Elastic Time Computation in QoS-Driven Hypermedia Presentations

Bruno Bachelet, Philippe Mahey, Rogério Rodrigues, Luiz Fernando Soares

Elastic Time Computation in QoS-Driven Hypermedia Presentations

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

Rogério Rodrigues and Luiz Fernando Soares [2]
Departamento de Informática, PUC-Rio,
Rio de Janeiro, Brazil.

Research Report LIMOS/RR04-16

[1] {bachelet,mahey}@isima.fr
[2] {rogerio,lfgs}@inf.puc-rio.br

Abstract

The development of hypermedia/multimedia systems requires the implementation of an element, usually known as formatter, which is in charge of receiving the specification of a document and controlling its presentation. In order to orchestrate the presentation, formatters should build a presentation plan that will contain the scheduling time for each document object and the inter media-object synchronization information, including those whose time of occurrence cannot be predicted, like relationships coming from user interaction. Besides orienting the presentation scheduling, the plan will guide prefetching, reservation and adaptation mechanisms in charge of maintaining the presentation quality of service. Adjustment in the duration of media objects is one of the most important adaptation techniques in order to maintain spatio-temporal relationships specified in a hypermedia document. Elastic time computation accomplishes this goal by stretching and shrinking the ideal duration of media objects. This paper presents new elastic time algorithms for adjusting the hypermedia document presentation in order to avoid temporal inconsistencies. The algorithms explore the flexibility offered by some hypermedia models in the definition of media object durations, choosing objects to be stretched or shrunk in order to obtain the best possible quality of presentation. Our proposals are based on the out-of-kilter and the cost-scaling methods for minimum-cost flow problems on temporal graphs. An aggregation procedure enhances the basic algorithm offering more flexibility in modeling real-life situations in comparison with other previous work based on linear programming.

Keywords: hypermedia presentation, elastic time computation, minimum-cost flow and tension.

Résumé

Le développement de systèmes hypermédia/multimédia requiert l'implémentation d'un élément, généralement connu sous le nom de formateur, chargé de recevoir les spécifications d'un document et de contrôler sa présentation. Afin d'orchestrer la présentation, les formateurs doivent construire un plan de la présentation qui contiendra la date planifiée de chaque objet du document et des informations de synchronisation inter-objet, incluant celles où la date d'occurrence ne peut pas être prévue, comme les relations issues d'interactions avec l'utilisateur. En plus d'orienter la planification de la présentation, le plan guidera les mécanismes de mise en cache, de réservation et d'adaptation en charge de maintenir la qualité de la présentation. L'ajustement de la durée des objets multimédia est l'une des plus importantes techniques d'adaptation pour assurer les relations spatio-temporelles spécifiées dans un document hypermédia. Le calcul de temps élastique réalise cet objectif en étirant ou contractant la durée idéale de présentation des objets multimédia. Cet article présente de nouveaux algorithmes de calcul de temps élastique pour ajuster la présentation de documents hypermédia, afin d'éviter les inconsistences temporelles. Les algorithmes explorent la flexibilité offerte par certains modèles hypermédia dans la définition de la durée des objets multimédia, choisissant les objets qui doivent être étirés ou contractés pour obtenir la meilleure qualité possible pour la présentation. Nos propositions reposent sur des méthodes de mise à conformité et de mise à l'échelle pour des problèmes de flot de coût minimum dans des graphes temporels. Une procédure d'agrégation améliore l'algorithme de base pour offrir plus de flexibilité dans la modélisation de situations réelles en comparaison avec de précédents travaux reposant sur la programmation linéaire.

Mots clés : présentation hypermédia, calcul de temps élastique, flot et tension de coût minimum.

Acknowledgements / Remerciements

This project was partially funded by France-Brazil cooperation project CAPES-COFECUB 398/02.

Elastic Time Computation in QoS-Driven Hypermedia Presentations

Bruno Bachelet[*] Philippe Mahey[*] Rogério Rodrigues[+] Luiz Fernando Soares[+]

[*]{bachelet, mahey}@isima.fr
LIMOS, CNRS
ISIMA, Université Blaise Pascal

[+]{rogerio, lfgs}@inf.puc-rio.br
Departamento de Informática
PUC-Rio

Abstract

The development of hypermedia/multimedia systems requires the implementation of an element, usually known as formatter, which is in charge of receiving the specification of a document and controlling its presentation. In order to orchestrate the presentation, formatters should build a presentation plan that will contain the scheduling time for each document object and the inter media-object synchronization information, including those whose time of occurrence cannot be predicted, like relationships coming from user interaction. Besides orienting the presentation scheduling, the presentation plan will guide pre-fetching, reservation and adaptation mechanisms in charge of maintaining the presentation quality of service. Adjustment in the duration of media objects is one of the most important adaptation techniques in order to maintain spatio-temporal relationships specified in a hypermedia document. Elastic time computation accomplishes this goal by stretching and shrinking the ideal duration of media objects.

This paper presents new elastic time algorithms for adjusting the hypermedia document presentation in order to avoid temporal inconsistencies. The algorithms explore the flexibility offered by some hypermedia models in the definition of media object durations, choosing objects to be stretched or shrunk in order to obtain the best possible quality of presentation. Our proposals are based on the "out-of-kilter" and the cost-scaling methods for minimum-cost flow problems on temporal graphs. An aggregation procedure enhances the basic algorithm offering more flexibility in modeling real-life situations in comparison with other previous work based on linear programming.

1. Introduction

The development of hypermedia/multimedia systems requires the implementation of a component, from here on called *hypermedia formatter*[1] [BuZe93a, SoRM00], which is in charge of receiving the specification of a document and controlling its presentation. Besides controlling content presentation of different media types, hypermedia formatters should try to guarantee that specified relationships among media objects are respected, assuring a document presentation of good quality.

The inter media-object synchronization issue — which is related with spatial and temporal relationships specified by the author, the duration assumed by each media object presentation, and the time that each presentation action occurs (play, stop, pause, etc.) — is a QoS (quality of service) problem, involving all service-offering-environment (SOE) components: client (formatter), servers and communication providers [RoSo03].

In order to orchestrate a presentation, the formatter should build a presentation plan that will contain the scheduling time for each document object and the inter media-object synchronization information, including those whose time of occurrence cannot be predicted, like relationships

[1] *Engine*, *player*, etc. are other names given in the literature for this component.

coming from user interaction. Besides orienting the presentation scheduling, the presentation plan, indeed a temporal graph, will guide pre-fetching, reservation and adaptation mechanisms, which are all important elements in the maintenance of the document presentation QoS.

Formatters should consider unpredictable factors related to a document presentation (unpredictable object duration, media object alternatives, unpredictable relationships, etc.) and should be able to adapt the document exhibition to improve its quality. Sometimes, formatters should be able to adapt the durations of some objects to maintain the document spatio-temporal consistency. One possibility is to perform adaptations before starting the presentation, when they are called compile-time adaptations. In this situation, adaptation techniques may take into account the user profile and preferences, and characteristics of the exhibition platform, including those of the transport network.

In order to minimize object presentation lags and to compensate content jitter caused by the service-offering-environment components, the incorporation of a prefetching/caching mechanism in hypermedia formatters is also desirable. In addition, prefetching is important to reduce the probability of needing adjustments during document runtime, helping the temporal synchronization maintenance. Prefetching tasks are simplified if the SOE provides QoS support [RoSo03].

Unfortunately, compile-time adaptations, prefetching and QoS provisioning are not sufficient to guarantee a good presentation. There are situations that may impair the document exhibition as planned by its author. For instance, in the prefetching, the cache memory may not have sufficient space to store the needed data; or in the QoS provisioning, the SOE platform (operating systems, networks, etc.) may not guarantee the requested QoS, or even offer only the best-effort service. In all these cases, presentation adjustments are needed, like shrinking or stretching the presentation durations (or presentation rates) of some objects, or replacing some objects by other ones with lesser QoS requirements, or even discarding some objects.

Therefore, runtime adaptation techniques may be necessary. Like at compile time, they may take into account the user profile, user preferences, and platform characteristics. However, different from the compile-time adaptation, which is typically based on context information that is static or does not change very much during the presentation, runtime adaptation techniques may also take into account dynamic behaviors, like user interactions and SOE performance parameter changes that may result in loss of synchronization.

Both at compile time and during runtime, adjustment in media object durations is one of the most important adaptation techniques in order to maintain spatio-temporal relationships specified in a document. Elastic time computation, as formerly named by [KiSo95], accomplishes this goal by stretching and shrinking the ideal duration of media objects. Of course, elastic time computation is only meaningful when hypermedia systems are based on document models that provide flexibility in the definition of media-object presentation characteristics. Usually, the specification of flexible durations contains some kind of cost information that gives the formatter a metric for maximizing the global quality of the presentation, while obeying other criterion, such as trying to minimize the number of objects that will be stretched or shrunk. Besides the difficulty of modeling such problem in real-life situations, where user interactions induces non-deterministic data, or where the definition of quality metrics depends on multi-criterion analysis, it must also be kept in mind that these kind of algorithms should run in very short execution time.

Some proposals for elastic time computation have been addressed in the literature [BuZe93b, KiSo95, LaSR02]. However, Firefly [BuZe93b] and Isis [KiSo95] have limited their study to simplified academic models, which lead to small-scale linear programs solved by the simplex method. Furthermore, their algorithms are suitable only for compile-time adaptation. Madeus formatter [LaSR02] presents a solution for runtime adjustment, but it is based on a greedy approach,

ignoring any cost specification to improve the document quality. In this paper, new algorithms are proposed. The first one is based on the "out-of-kilter" method for minimum-cost tension problems [Pla71], which allows much more flexibility to model real-life applications. The algorithm is validated in the deterministic case with general convex cost functions, being suitable for both compile time and runtime. Another approach, the "dual cost-scaling" is also proposed, which converts the problem into a minimum-cost flow problem solved with the "cost-scaling" method [AhHO99]. This method is better adapted for big graphs at compile time. A new method, based on the specific structure of temporal graphs and called "aggregation", is also presented. It concentrates on the sequential and parallel aspects of temporal constraints, providing efficient resolution time. This makes the method well suited to developing runtime heuristics. The two last approaches work with piecewise linear convex cost functions only.

This paper is based on the HyperProp formatter, which was used to validate all ideas proposed here. The HyperProp formatter is a specialization of the formatter framework [BSRS04] developed at the TeleMidia Laboratory of the Catholic University of Rio de Janeiro. The framework provides support for context adaptations; inter and intra media-object QoS provisioning; and support for several hypermedia models and languages, through a collection of converters. The system already has converters developed to read SMIL [W3C01] and NCL/NCM[2] [Much03, SoRM00] documents. HyperProp formatter is implemented in Java and is available in http://www.telemidia.puc-rio.br/formatter/. The next sections of the paper are organized as follows. Section 2 introduces some document-model entities that are necessary when adaptations are available. The section highlights that media-objects should have "elastic duration" in order to favor the maintenance of the document temporal consistency. Compile-time adaptations, that is, those made before the start of the document presentation, are discussed in Section 3, with special emphasis to elastic time adaptation. Elastic time computation is the matter of Section 4, where the algorithms proposed in this paper are presented. Two algorithms are extensively compared. The first one is an adaptation of the out-of-kilter method to minimum-cost tension problems, while the second one is a dual cost-scaling algorithm. Both outperform the simplex method when dealing with piecewise linear costs. The first algorithm is more general, as it can deal with general convex costs; in addition, it is also more flexible, as it can be adapted to runtime issues. The second method is better adapted for big graphs at compile time. Another approach is also presented, an aggregation algorithm that offers similar performances and proposes to concentrate on the sequential and parallel temporal constraints, opening new possibilities for runtime computation. Section 5 discusses how pre-fetching and QoS provisioning may be used to avoid, or to reduce, adaptations at runtime phase. Section 6 goes back to the adaptation matter, but now during runtime, leaving for Section 7 the discussion about runtime elastic time computation. Related work is presented and compared in Section 8. Finally, Section 9 is left for the conclusions and future work.

2. Adaptive Hypermedia Document

The specification of a hypermedia document is accomplished by the definition of its basic elements (text, audio, video, etc.) and their spatial and temporal relationships.

The basic elements, usually called *media objects* or *content nodes*, have as main attributes the content to be presented (often a reference to the content) and some additional parameters that define the object presentation behavior (e.g., duration, spatial characteristics, anchors, etc.).

[2] NCM (Nested Context Model) is the document model of NCL (Nested Context Language).

Anchors are a subset of the information units of a media object and define events. As stated in [PéLi96], an *event* is an occurrence in time that can be instantaneous or can occur during some time period. Hypermedia models define several types of events, among them: *presentation event*, which is defined by the presentation of an anchor of an object; *selection event*, which is defined by the selection of an anchor of an object; and *attribution event*, which is defined by a change in the value of an attribute of an object.

Each event defines a state machine [SoRM00] that must be maintained by the formatter. Figure 1 shows the NCM presentation-event state machine, as an example.

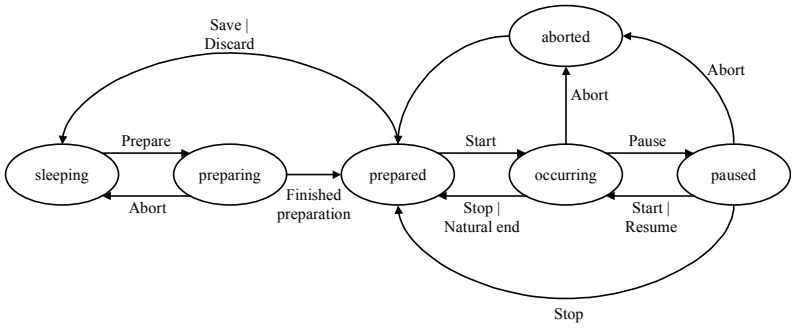

Figure 1 – NCM presentation-event state machine.

A presentation event can change from *occurring* to *prepared* in two situations: as a consequence of the natural end of the presentation duration, or due to an action that stops the event. The duration of an event is the time it remains in the occurring state. This duration can be intrinsic to the media object or specified by an author, like in SMIL and NCL/NCM. Synchronizations between media objects are indeed specified as relationships between event state machines, as will be seen further on.

Some hypermedia models allow the definition of sets of alternative objects for presentation. Each alternative object should be associated to an exhibition rule. When a rule is satisfied, the associate object is chosen to be presented. Rules can evaluate user preferences or platform conditions (available exhibition tools, network bandwidth, whether resource reservation is available, etc.). Some kind of context aware mechanism (internal or external to the formatter) should be looked up to support the evaluation [DeSA01].

Likewise, some models allow the definition of sets of alternatives of exhibition for the same content. For example, a text object could have two presentation alternatives: one as a formatted text and another one as a synthesized and synchronized audio and video[3]. Actually, exhibition alternatives always can be specified as alternative objects (different media objects with different presentation information attributes), but it is better for reuse purposes if the document model separates media object content from presentation characteristics, enabling each part to be independently adapted.

Of course selection rules can be composed. For example, from the user preferences, the formatter may determine which text idiom to be selected. However, upon discovering that the user is driving a

[3] HyperProp formatter has a player, called "Expressive Talking Heads" [Rodr04], which is able to receive a text as input and to generate a synthesized audio, synchronized with an animated graph as output, representing a human face enunciating the text.

car (exhibition platform information), and being unable to read at that moment, the formatter can also choose to synthesize the selected text.

SMIL formatters (called *players*) [W3C01], NCL/NCM formatters and the Cardio-OP application [KlGF99] are examples of tools able to make presentation adaptation based on media-object alternatives (*switches*) and context information. In Cardio-OP, a cross-media strategy [BoKW99] allows dynamic creation of alternatives. NCL/NCM formatters also make presentation adaptations based on exhibition alternatives for the same content (*descriptor switches*).

Relationships among events of media objects are usually specified by *links*, *compositions*, or both, depending on the conceptual model in use. Examples of composition usage can be found in parallel and sequential elements of SMIL [W3C01], in Isis primitives [KiSo95] and in templates of NCL [MuSo02]. Examples of link usage can be found in HTML hyperlinks, in NCL spatio-temporal links [SoRM00], in timed Petri net transitions of OCPN [LiGh90] and TSPN [DiSé94], and in Firefly temporal point nets [BuZe93a].

Depending on how they are specified, temporal relationships can be classified as causal or constraint relationships. *Causal relationships* are based on conditions and actions. They define actions that must be fired when some conditions are satisfied. An example of causal relationship is: "when video *A* presentation finishes, stop playing audio *B*". *Constraint relationships* specify restrictions that must be satisfied during a presentation, without any related causality. An example of constraint relationship is: "video *A* and audio *B* must finish their presentations at the same time". Although similar, the former example (causal relationship) does not ensure that both media objects will be entirely played. If *A* finishes its presentation first, *B* presentation will be interrupted. On the other hand, the latter example (constraint relationship) specifies that the presentations should finish at the same time and in a way that both media objects have all their content presented. It is worth mentioning that this constraint must be satisfied if and only if both media objects are presented simultaneously (i.e., only the presentation of *A* or only the presentation of *B* also satisfies the constraint relationship).

Both kinds of relations are important to hypermedia presentation modeling. Models based only on temporal constraints do not capture the causality intrinsic to hypermedia scenarios and can lead to specification ambiguities [DuKe95]. For instance, let us consider the *meet* constraint relation of Allen [Alle83], exemplified in Figure 2. Just specifying that a media object *X* meets another media object *Y* does not let one know if the end of *X* causes the start of *Y*, if the start of *Y* causes the end of *X*, or if the end of *X* should coincide with the start of *Y*, the last one without any causal relationship. On the other side, models purely based on causal relations do not allow an author to specify, for example, that two media objects should finish their presentation simultaneously, both having their content entirely presented, as previously exemplified.

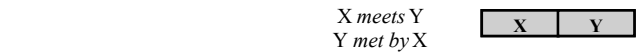

Figure 2 – Allen *meets* relation.

Models based only on causal relationships cannot generate temporal inconsistencies. However, they can lead to deadlocks (objects whose presentation depends on impossible conditions) or spatially inconsistent specifications, for example, resource contention (two objects that need to be presented at the same position/resource and at the same time). On the other hand, models with constraint relationships can have inconsistent temporal specifications. For these models, it is important to verify whether all temporal constraints can be satisfied.

Regarding temporal consistency checking, it is desirable to have models that allow the specification of a flexible duration for media object presentation (and thus its events), taking advantage of the fact that several media types can be temporally stretched or shrunk, without impairing human comprehension. Instead of taking an inflexible ideal duration specification that could lead to inconsistencies, it would be possible to present a consistent document, even with some loss of presentation quality. However, with the introduction of flexibility, besides verifying the temporal consistency of a document, it will be desirable to find the event duration values that satisfy all given temporal constraints and leads to the best possible presentation quality of the whole document. Obviously, temporally inconsistent documents may still exist, since the range of adjustments is usually limited.

Models that support presentation flexibility should specify durations using cost functions. The idea is to allow an author to specify a function "cost *versus* duration", for each presentation event that constitutes the object presentation. Figure 3 shows examples of cost functions. Intuitively, these functions give the cost for shrinking or stretching the duration from an ideal value, where the ideal value is given by the function minimum points. It is also important to allow durations to be specified not only by real values, but also by unpredictable values. Media objects that are presented until some external action interrupt them and media objects that finish their presentations by themselves, but in some instant not known a priori, are examples of events with unpredictable duration.

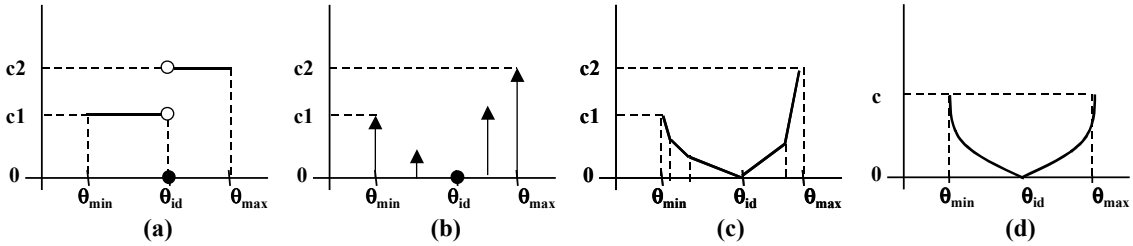

Figure 3 – Examples of cost functions. a) Step function; b) Simple discrete function; c) Continuous linear piecewise function; d) Continuous non-linear function.

3. Compile-Time Adaptation

As aforementioned, there are two moments when formatters can apply adaptation strategies: at compile time, when building the presentation plan (object scheduling plan for presentation); and in runtime, when adjustments of the presentation plan are needed. Both compile-time and runtime adaptations may take into account the user profile and preferences, user interactions and performance parameters of the SOE (exhibition platform, servers and the transport network). This section discusses compile-time adaptations, leaving the discussion of runtime adaptations to Section 6.

Usually, before starting a document presentation, a hypermedia formatter first resolves the rules associated to sets of alternative objects and to sets of alternatives of exhibition for the same content[4]. This will identify which objects will take part in the presentation plan and how they will be presented. Afterwards the formatter may apply any necessary cross-media adaptation, like transforming text into synthesized speech. Only then, elastic time adjustments may be applied in

[4] Some document models, as NCL/NCM and SMIL, also allow specifying rules directly associated to single objects, without component alternatives. When a rule of this type is evaluated as false, the respective object is simply ignored during the presentation.

order to maintain the temporal synchronization defined in the document. The expected duration of each event is then established, as will be discussed in the next section. The elastic time adjustment can also take into account external constraints, as for example, the total document presentation duration.

Static media object duration (like logos and captions) can be easily adjusted. Continuous media objects derived from the synthesis of a static content usually are also easy to have their duration adjusted [Rodr04]. On the other hand, durations of continuous media objects are difficult to adjust, except by the simple cut or repetition of their last samples.

At compile time, several unpredictable factors may prevent building the whole presentation plan. For example: there may be rules to chose alternatives that can only be evaluated during runtime; or the time an object starts may depend on a user interaction over an anchor; or there may be a live object with unknown duration making part of the presentation.

It should be noted that the presentation plan is not necessarily a timeline. A presentation-plan data structure based on time chains of events was proposed in Firefly [BuZe93a, BuZe93b]. One main temporal graph[5] is built corresponding to a sequence of predictable presentation events, initiated by the event that corresponds to the beginning of the document presentation. A predictable event is one that can have its time of occurrence previewed, based on the occurrence time of another event. For example, if the duration of an object is known, the event corresponding to the end of its exhibition can be predicted from the event corresponding to the start of the exhibition. Another example of predictable event occurs when there is a causal relationship between two objects specifying, for instance, that the end of the first object exhibition must trigger the starting of the second object exhibition.

Many multimedia documents have only the main temporal graph, but hypermedia documents usually have one or more auxiliary temporal graphs of events, each one being initiated by an unpredictable event; like a user anchor selection, the end of a live object with unpredictable duration, etc. All events of an auxiliary temporal graph must be predictable in relation to another event present in the same auxiliary graph.

During runtime, each auxiliary graph is joined to the main temporal graph, as soon as its initial unpredicted event happens. Therefore, only at the time the last unpredicted event of the document happens, can the whole temporal graph be obtained.

At compile time, it is only possible to maintain temporal consistency within a temporal graph. Hence, elastic time computation is evaluated for each one of them, independently.

4. Elastic Time Computation at Compile Time

Algorithmic issues related to the computation of optimal elastic times at compile phase are analyzed in this section. As explained in the preceding section, a temporal graph associated with the temporal relationships between events will be built and used to model the optimization problem. It will be assumed that each event has a variable duration which should be computed within a given elastic time interval, so that all synchronization relationships between events are satisfied, and a given criterion measuring the overall cost of the adjustments of object durations is minimized. Several

[5] Timed Petri Nets and other models can be used in the temporal graph specification, which will be more precisely defined in Section 4.

problems can be defined for each type of criterion and for each additional constraint introduced to model some specific situation. The commonly used objective functions are (see Figure 3):

1. Minimize the weighted sum of absolute deviations from ideal duration values.
2. Minimize the maximal absolute deviation from ideal duration values.
3. Minimize the number of objects with at least one distorted event.
4. Minimize or maximize the total duration of the presentation (or of any subset of objects contained in the presentation).

Observe that the first two objectives are particular cases of metric functions that measure the distance to an ideal situation. Any positive convex function that is equal to zero when all events stay at their ideal duration could be used. The case of euclidean norms (leading to least-square minimizations) is quite interesting for optimization purpose as it ensures unicity of the optimal solution, even if it yields a non-linear model. The choice 1 will tend to introduce many alternative decisions and, at the opposite, the choice of criterion 2 will tend to spread the deviations on many objects (see Annex 1 for a complete study of an academic model). A compromise between criteria 1 and 3 should be ideal but it is very difficult to be properly implemented, as the third objective function is purely combinatorial.

Additional constraints can be added in the model, like fixing the duration of the presentation, or forcing the deviations to lie in a discrete set. The first one is very easy to consider in the model, as it is a simple synchronization relation of a dummy event associated with the whole presentation. The second one will lead to mixed-integer mathematical programs, which are very hard to solve.

In the sequel, criterion 1 is chosen for consideration. The main algorithm for continuous convex cost functions with no additional constraints is analyzed, based on the out-of-kilter method for minimum-cost flow problems [Fulk61] adapted to temporal graphs (where the flows are induced by the optimality conditions as dual objects with respect to durations, as explained below). A second algorithm based on cost-scaling [AhHO99] will also be presented. Temporal graphs are mainly built from sequential and synchronization constraints and may correspond to a very specific class of graphs, the series-parallel graphs [BaMa03a], which are usually much easier to treat computationally. Unfortunately, not all temporal constraints can be modeled by a series-parallel structure, which leads to consider quasi series-parallel graphs. A third method is then proposed, called “reconstruction”, which combines the out-of-kilter with the “aggregation” technique [BaMa03a].

The numerical study will not only rely on the computational performance of the algorithms, but will show that the methods should be as simple as possible in its formal adaptation to the real-life cases. The methods should be scalable, in the sense that the elastic time computation algorithms should take profit of the results computed before any mismatch[6] happens, in order to improve the on-the-fly calculus performance. Discussion on these runtime issues continues in Section 7.

However, the final goal of this study is to provide methods for elastic time computation that take into account other criteria than the objective function 1 introduced above. It is also important to be able to:

1. Minimize the number of objects that will have events with expected duration different from ideal duration (if there is more than one optimal choice with the objective function 1).

[6] We say that a mismatch occurs when the expected time for an event occurrence, defined by the formatter algorithm, is no longer satisfied.

2. Choose the fairest solution that spreads changes in the ideal duration across the objects (in the case that there is more than one solution that satisfies the synchronization constraints).
3. Minimize the total cost of shrinking and stretching, given a fixed duration for the whole document presentation.
4. Find the shortest and longest possible duration for the document.
5. Evaluate new ranges for which the temporal consistency of the document is valid, when a media object begins to be presented. This will be very important to help the formatter in adjusting object duration on-the-fly.

4.1. Temporal Graph and Optimality Conditions

The basic model is a directed graph $G=(X;U)$, where X is the set of nodes and U is the set of arcs (let $n=|X|$ and $m=|U|$). Nodes represent event-state transitions (the start or end of a presentation event), and arcs represent the precedence relationships between these events. *Head* and *tail* nodes are added to connect all nodes with no predecessor or successor, respectively, in order to represent the start and the end times of the temporal scenario. Except for the head outgoing arcs (and the tail incoming arcs), each arc is assigned a cost function. The function defines a feasibility interval $[\theta_{ii}^{\min},\theta_{ii}^{\max}]$, containing an ideal length $\hat{\theta}_u$, and is supposed to be a positive convex and real function on the feasibility interval.

When two or more arcs are incident to a common node, synchronization constraints occur, which correspond in terms of graph theory to the existence of cycles (e.g. [AhMO93]) made of two directed paths. A graph that represents these synchronization constraints will be called a *temporal graph*.

4.1.1. Modeling

We must first observe that, with the temporal graph, we do not intend to model the whole author's specification, but only to represent the main part of the temporal constraints. Here are some examples of how to model author specifications by the temporal graph:

- Let i be a presentation event, as defined in Section 2 and presented in Figure 4a. In the temporal graph, i is modeled by two nodes, s_i and e_i, which represent the start (s_i) and end (e_i) of event i occurrence. Note that some actions, like the preparation (pre-fetch) of the event, are not considered here. An arc u_i is also created to model the duration of event i occurrence. The cost function related to this duration is then assigned to u_i.
- Consider now two presentation events, i and j, which must start at the same time, as shown in Figure 4b. In the temporal graph, i and j are modeled as above, but the nodes s_i and s_j are merged to be a single node s_{ij}.
- Similarly, if i and j are two presentation events that must end at the same time, as illustrated in Figure 4c, the nodes e_i and e_j are merged to be a single node e_{ij}.

Note that the resultant graph can contain multiple arcs. Indeed, two events specified with the same starting and ending time will be represented by two arcs with the same source and target nodes. Temporal graphs are thus by nature multigraphs.

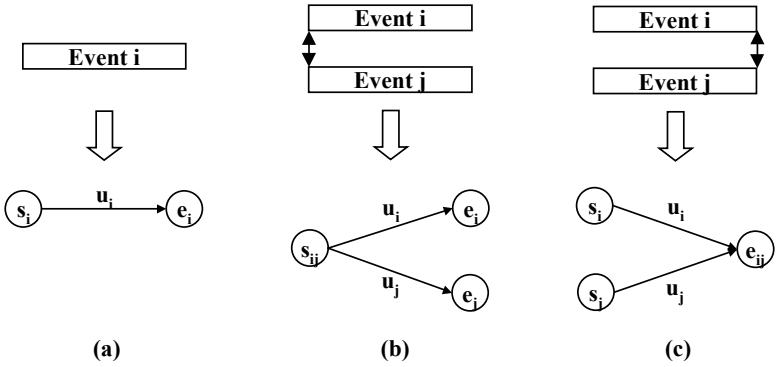

Figure 4 – Temporal graph representation for author specification of hypermedia event relationships.

4.1.2. Definitions

Some definitions that are useful for the description of the mathematical model must be recalled (all related notions can be found in [AhMO93]). Given a graph:

- A *cycle* is a chain of arcs, in an arbitrary direction, with a start node coinciding with the end node. For instance, an elementary cycle is the one formed by two paths with the same source and target nodes. A cycle *v* can be seen as an *m*-vector, where its elements v_u are equal to *+1* (if the arc *u* belongs to the cycle in a given direction), *-1* (if the arc *u* belongs to the cycle in the other direction) or *0* (if the arc *u* does not belong to the cycle).
- A *cut* is the set of arcs that connects one subset of nodes to its complement. Like a cycle, a cut *w* can be seen as an *m*-vector, where its elements w_u are equal to *+1* (if the arc *u* belongs to the cut in a given direction), *-1* (if the arc *u* belongs to the cut in the other direction) or *0* (if the arc *u* does not belong to the cut).
- A *flow* is a vector of values assigned to each arc, such that the flow conservation law is satisfied at each node (thus, the algebraic sum of the flow values on a cut is necessarily zero).
- A *potential* is a value assigned to a node.
- A *tension* is a vector of values assigned to each arc, such that each tension value of an arc is the difference between the potentials of its end nodes (thus, the algebraic sum of the tension values on a cycle must be necessarily zero).

4.1.3. Minimum-Cost Tension Problem

Former proposals for elastic time computation have been addressed with linear programming. These models use piecewise linear convex cost functions: [BuZe93b] defined a model based on potentials, whereas [KiSo95] defined a model based on cycles. We will extend the models to deal with general convex cost functions and give a complete description of the optimality conditions.

Back to the temporal graph, if an initial date π_0 is arbitrarily assigned to the head node, we can assign a date π_i to each node *i*, which we call the potential of the node. We define π as an *n*-vector with components π_i for each node *i*. The duration of an object associated to a given arc *u=(i,j)* can be seen as the difference of the potentials of its end points. This is indeed the definition of a tension vector θ on the graph, such that $\theta_u = \pi_j - \pi_i$ on each arc *u*. The problem is then a minimum-cost tension problem on the temporal graph [BeGh62]. Using vectorial notation and the node-arc incidence matrix of the graph, i.e., the $(m \times n)$ matrix *A*, with elements a_{iu} equal to *-1* (if *u* goes out of node *i*), *+1* (if *u* goes into node *i*) and *0* (for all other cases), the problem is simply:

Minimize $\sum_{u \in U} c_u(\theta_u)$

Subject to $\theta = A^T \pi,\ \theta^{\min} \le \theta \le \theta^{\max}$

Note that the problem can be modeled without introducing potential variables, but we need to know in advance all synchronization cycles. For any cycle, the sum of the arc durations in one direction must be equal to the sum in the other direction. If S is the arc-cycle incidence matrix, where each column is a cycle vector (Section 4.1.2), then $\theta = A^T \pi \Leftrightarrow S^T \theta = 0$. The problem can thus be modeled as:

Minimize $\sum_{u \in U} c_u(\theta_u)$

Subject to $S^T \theta = 0,\ \theta^{\min} \le \theta \le \theta^{\max}$

One may ask how to build the cycle matrix S. Obviously, it is not possible to enumerate all the cycles of a graph. However, it has been proved that only a subset of cycles, called the *cycle basis*, is enough to ensure the tension property. How to find such a cycle basis is presented in [Bach03].

4.1.4. Optimality Conditions

The optimality conditions for convex costs have been derived by Rockafellar for piecewise smooth convex functions [Rock84] as a generalization of the primal-dual optimality conditions for the linear case [Fulk61]. They are based on the existence of dual multipliers for tension problems that define a flow vector φ (indeed, the multiplier vector must be orthogonal to the tension space defined by $S^T \theta = 0$, which implies that it is a linear combination of cycle vectors, i.e., a flow). Then, for each arc u, the pair (θ_u, φ_u) of tension and flow values must lie on the so-called *kilter* curve, defined by the graph of the subdifferential of the cost function. In other words, let $c'(\theta_u)$ denotes a subgradient (left or right derivative) of the cost function c, so that we have:

1. $c'_u(\theta_u) = \varphi_u$ on each arc, such that $\theta_u^{\min} < \theta_u < \theta_u^{\max}$,
2. $c'_u(\theta_u) - \varphi_u \ge 0$ on each arc, such that $\theta_u = \theta_u^{\min}$,
3. $c'_u(\theta_u) - \varphi_u \le 0$ on each arc, such that $\theta_u = \theta_u^{\max}$.

Thus, the flow value is the marginal cost of the decision on the arc (positive for stretching, negative for shrinking and zero for keeping the object duration with its ideal value). An immediate consequence is that, if two arcs belong to the same path in the same cycle, i.e., if they belong to a common sequence of presentation with no synchronization constraint between them, they have the same flow (due to the conservation law), so the same marginal cost. Indeed, the flow conservation implies that the flow values are equal on both arcs.

4.1.5. Feasibility Issues

In the deterministic case, a feasible schedule is characterized by the following conditions. For each synchronization cycle consisting of two paths p and p', we have:

$$\begin{cases} \sum_{u \in p} \theta_u^{\min} \le \sum_{u \in p'} \theta_u^{\max} \\ \sum_{u \in p'} \theta_u^{\min} \le \sum_{u \in p} \theta_u^{\max} \end{cases} \quad (*)$$

Consequently, the flexibility interval for a given cycle *(p,p')* is:

$$\left[\max\left\{\theta_p^{\min},\theta_{p'}^{\min}\right\},\min\left\{\theta_p^{\max},\theta_{p'}^{\max}\right\}\right] \quad (**)$$

where θ_p stands for the total path duration. Hence, conditions *(*)* (that ensure that a feasible schedule exists) also guarantee that the intervals *(**)* are not empty. Finding out an initial feasible tension is by itself an interesting issue that should be solved at real-time, when some unspecified event affects the predefined schedule during the document presentation.

Cycle Increase Method

A very simple and quite efficient algorithm can be used to solve the problem. It starts with a null tension. For each arc u, where $\theta_u \leq \theta_u^{\min}$, a cut containing u can be found, for which the tension can be increased without making any other arc incompatible. The maximum increasing value is determined and applied to this cut. If θ_u is still inferior to $\theta_u^{\min}$, another cut is searched to increase the tension of u, else it goes on with another incompatible arc [Hadj96].

Shortest Path Method

Another algorithm, less intuitive, uses the shortest path method. The idea is to consider a potential π_i as the shortest distance between the head node and the node i. It is known that, if the potential on the whole graph represents shortest distances, the following property is verified for all arc $u=(i;j)\in U$: $\pi_j - \pi_i \leq d_{(i;j)}$, where $d_{(i;j)}$ is the length of an arc *(***)*.

In order to find a feasible tension, we have to look for a potential such that $\theta_u^{\min} \leq \pi_j - \pi_i \leq \theta_u^{\max}$. Going back to the optimality conditions *(***)*, we would like to have an arc where $\pi_j - \pi_i \leq d_{(i;j)} = \theta_u^{\max}$ and another arc (the opposite one) where $\pi_i - \pi_j \leq d_{(j;i)} = -\theta_u^{\min}$. Thus, the idea is to search for the shortest distances on the new graph made from the original graph, where all arcs *(i;j)* are doubled: when the existing arc *(i;j)* has the distance $\theta_u^{\max}$, a new arc *(j;i)* is added with the distance $-\theta_u^{\min}$ (cf. Figure 5). Using, for instance, the algorithm of Bellman and Ford [AhMO93] to find the shortest distances, we get a potential that defines a feasible tension. This method is faster than the previous one (it was shown to be 5 to 50 times faster, depending on the size of the graph [Bach03]). However, with the previous solution, it is possible to start from an "almost feasible" tension, what may be interesting during runtime.

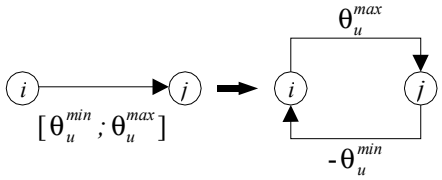

Figure 5 – Transformation to a shortest path problem.

4.2. The Out-of-Kilter Method

The first method proposed in this paper to satisfy the criterion 1 of Section 4 (minimize the weighted sum of absolute deviations from ideal event duration values) is an adaptation of the out-of-kilter method [Fulk61], first applied to minimum-cost tension problems [Pla71] in the linear case. Polynomial and strongly polynomial algorithms have been recently studied [Hadj96], but their performance in the convex case are still too slow for our purpose. The out-of-kilter algorithm is a

primal-dual method, which updates tensions and flows, iteratively, until optimality is reached. The main advantage is that the computations are distributed among the arcs, allowing much more flexibility than direct methods. The key tool is the kilter curve, which represents the local optimality condition on a specific arc. We consider two kinds of costs: piecewise linear costs (Figure 6a1) and convex smooth costs (Figure 6a2).

Consider now the kilter curve of an arc u relating its tension θ_u with its flow φ_u. For the linear cost, the kilter curve is represented by Figure 6b1, and for the convex cost, it is shown in Figure 6b2. In both cases, it is said that an arc u is *in-kilter* if the pair $(\theta_u ; \varphi_u)$ lies on the curve, otherwise the arc is said *out-of-kilter*. We can easily prove that if all arcs of the graph are in-kilter, then the tension is minimum (see the optimality conditions in Section 4.1.4). So, all difficulty is to bring all arcs on their kilter curve. We first explain how to perform that for piecewise linear costs. Then, we discuss the convex costs case. The solution strategy is inspired from [Pla71] and [Hadj96] in solving the minimum-cost tension problem with linear costs.

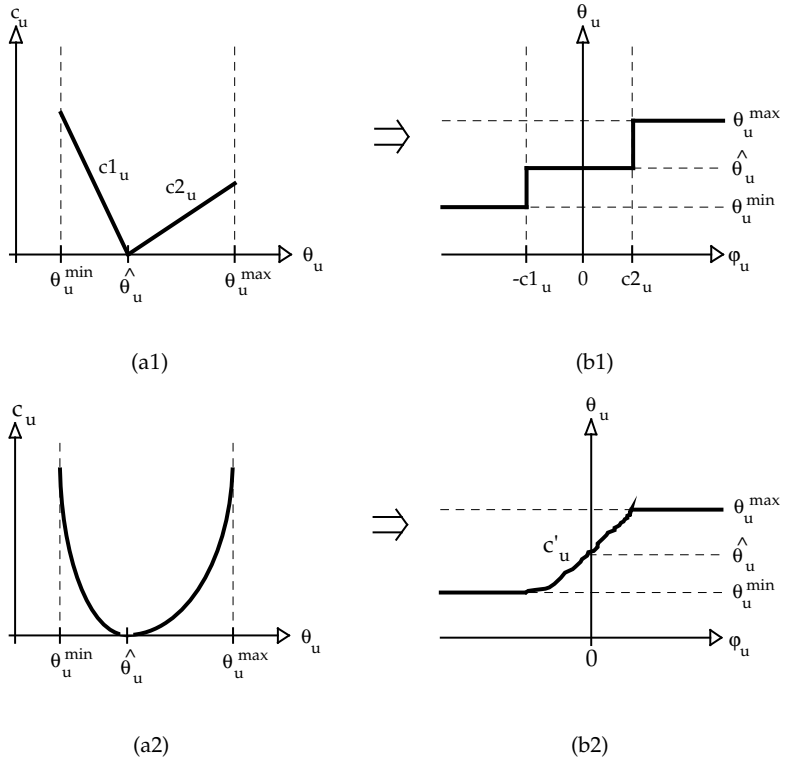

Figure 6 – a*) Cost functions. b*) Kilter curves. *1) Piecewise linear. *2) Convex smooth.

4.2.1. Piecewise-Linear Costs

First, we classify the arcs of the graph into 4 categories:

1. arcs below the kilter curve,
2. arcs above the kilter curve,
3. arcs on a horizontal part of the kilter curve,
4. arcs on a vertical part of the kilter curve.

For arcs in category 1, their tension must be increased or their flow must be decreased in order to improve their position towards the kilter curve. For arcs in category 2, their tension must be decreased or their flow must be increased. In categories 3 and 4, arcs are in-kilter. For arcs in category 3, only their flow can be modified without making them out-of-kilter, while for those in category 4, only their tension can be changed.

Following the seminal work of Fulkerson [Fulk61], we can set that:

- for an arc u in category 1, we can find either a cycle containing u for which the flow can be decreased, or a cut containing u for which the tension can be increased;
- for an arc u in category 2, we can find either a cycle containing u for which the flow can be increased, or a cut containing u for which the tension can be decreased.

These statements allow us to present the following algorithm that improves the position of an arc to its kilter curve:

```
algorithm improveArc:

if (u is below the curve) then
 find a cycle γ to decrease the flow of u
 or a cut ϖ to increase the tension of u;

 if (cycle found) then
  find λ the maximum value the flow on γ can be decreased;
  φ = φ – λγ;
 else (cut found)
  find μ the maximum value the tension on ϖ can be increased;
  θ = θ + μϖ;
 end if;

else /* u is above the curve */
 find a cycle γ to increase the flow of u
 or a cut ϖ to decrease the tension of u;

 if (cycle found) then
  find λ the maximum value the flow on γ can be increased;
  φ = φ + λγ;
 else (cut found)
  find μ the maximum value the tension on ϖ can be decreased;
  θ = θ - μϖ;
 end if;
end if;
```

Thus we can propose the following method to make all arcs in-kilter, determining the minimum-cost tension. The method just applies the previous algorithm successively and repeatedly upon each arc of the graph until all arcs are in-kilter:

```
algorithm linearInKilter:

make θ compatible;
φ = 0;

while (an arc u is out-of-kilter) do
 for all arc u do
  if (u out-of-kilter) then improveArc(u);
 end for;
end while;
```

4.2.2. Convex Costs

The method is slightly different in the case of general convex costs. Indeed, as previously discussed, to move an arc u closer to its kilter curve, a cycle or a cut (containing u) must be found and the flow (or the tension) on this cycle (or this cut) must be modified. The flow (or the tension) of all arcs in the cycle (or the cut) is then modified. However, the flow and the tension are modified one at a time; they cannot be modified together in a same iteration. If the kilter curve is made of horizontal and vertical lines (Figure 6b1) like with piecewise linear costs, it is possible to move an in-kilter arc (by changing either its flow, or its tension) without leaving its kilter curve. In the case of general convex costs, the kilter curve is no more made of horizontal and vertical lines (Figure 6b2), which

means that changing either the flow or the tension of an in-kilter arc will automatically make it out-of-kilter. In order to avoid this problem and still use the out-of-kilter technique, we approximate the kilter curve by a staircase curve. We then use the method presented for the linear costs to bring all arcs on their approximate kilter curve and after that we improve the approximation of the curve. The procedure is repeated until we obtain the desired precision. It is recommended to start with a weak approximation and then progressively improve it, instead of directly starting with a sharp approximation. We outline below the algorithm for this method, where *p* is the precision:

```
algorithm convexInKilter:

ε = 0.1 sup{θ_u^max − θ_u^min , u in U};

while (ε > p) do
 make all arcs in-kilter with precision ε; /* Uses algorithm linearInKilter. */
 ε = sup{0.1ε,p};
end while;
```

4.3. The Dual Cost-Scaling Method

The second method proposed has the same goal as the previous one: to bring all arcs on their kilter curve. However, the way arcs are moved is quite different. It is based on an algorithm called *cost-scaling* for the minimum-cost flow problem [AhMO93], and has been proposed in [AhHO99].

A tension is said *ε-optimal* if all arcs of the graph are less than ε far from their kilter curve on the tension component, i.e., on the curve or in the gray area in Figure 7.

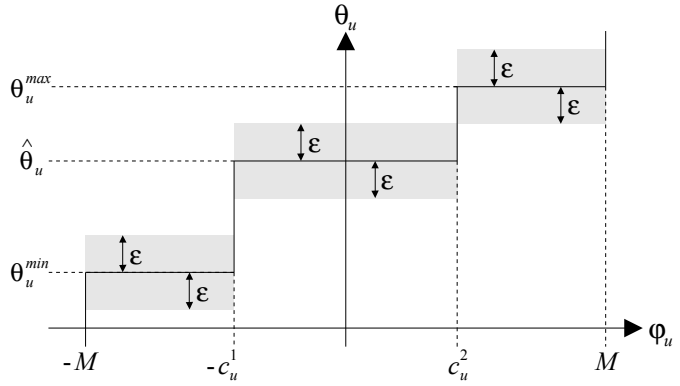

Figure 7 – *ε-kilter curve.*

The algorithm starts with a high value of ε, so that any feasible tension is obviously ε-optimal. It has been proved that $\varepsilon = A = \max_{u \in U}(\theta_u^{\min}; \theta_u^{\max})$ is such a value [AhMO93]. It has been also proved that if $\varepsilon < 1/n$ (where n is the number of nodes in the graph), then any ε-optimal tension is optimal [AhMO93]. The algorithm reduces the ε value, in each step, making the tension $\varepsilon/2$-optimal from an ε-optimal tension. Note that, for the purpose of the algorithm, it is required to bound the flow in an interval *[-M;M]*, where *M* is chosen to be the cost of any feasible tension [Bach03]. The main procedure of the algorithm can be stated as follows:

```
algorithm dualCostScaling:

π = 0;
ε = A;
φ = 0;

while (ε ≥ 1/n) do
```

```
 make π ε-optimal;
 ε = ε/2;
end while;
```

Making the tension $\varepsilon/2$-optimal from an ε-optimal tension is defined as follows. First, a pseudo-flow is built, i.e., the flow values of the arcs are put directly on the curve, on the nearest point (see Figure 8a). Hence, the flow conservation law is not satisfied anymore (i.e., for any node i, $\sum_{u=(i;j)}\varphi_u - \sum_{u=(j;i)}\varphi_u = 0$ is not always verified).

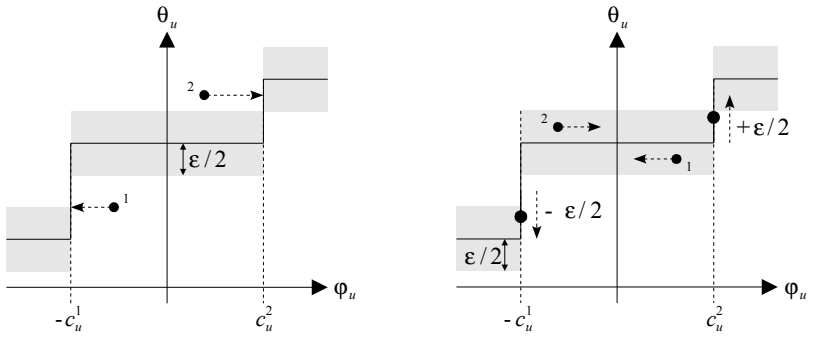

Figure 8 – *Pseudo-flow.*

Then, without moving any arc from the $\varepsilon/2$-kilter curve, the algorithm attempts to satisfy the flow conservation law. The process consists in finding an excess node i, i.e., a node with $\sum_{u=(j;i)}\varphi_u - \sum_{u=(i;j)}\varphi_u > 0$. The excess of flow is then withdrawn from the adjacent arcs. In other words, we search for incoming arcs whose flow can be reduced or outgoing arcs whose flow can be increased. To avoid any loop in the algorithm, the flow of an arc is increased/decreased only if it is above/below the curve [AhMO93]. If the flow cannot be entirely evacuated, the node potential is decreased of $\varepsilon/2$ (Figure 8b), therefore tensions of the incoming arcs decrease and tensions of the outgoing arcs increase. Thus, these arcs are moved to potentially provide flow capacity to evacuate the excess of flow. Note that this operation does not move any arc from its kilter curve. The whole procedure to make a tension ε-optimal can be stated as follows:

```
algorithm makeEpsilonOptimal:

build pseudo-flow φ;

while (an excess node i exists) do
 q = Σ_{u=(j;i)} φu − Σ_{u=(i;j)} φu ;

 while (q>0) do
  if (an arc u=(i;j) can increase its flow without leaving its kilter curve) then
   increase φu to its maximum;
  else if (an arc u=(j;i) can decrease its flow without leaving its kilter curve) then
   decrease φu to its minimum;
  else πi = πi - ε/2;
 end while;
end while;
```

4.4. The Reconstruction Method

Although the previous two methods work on any graph, we propose to investigate the class of *series-parallel* graphs (or *SP-graphs*), because they bear many common features with hypermedia

temporal graphs. Indeed, these latter graphs get their particularities from the existence of parallel paths (representing parallel presentations) between two nodes (events that synchronize the parallel presentations), and the cycles formed by these paths are nothing but elementary series-parallel graphs.

However, hypermedia temporal graphs are not exactly series-parallel. We explain later that, for instance, the *overlaps* temporal relation is modeled with a graph that is not series-parallel, but is very close. Such graphs are called quasi series-parallel graphs, or QSP-graphs. We propose first a method, called aggregation, to solve the tension problem on SP-graphs, and then we explain how to extend the technique to QSP-graphs.

4.4.1. Introduction to Series-Parallel Graphs

Formally, an SP-graph is a graph that cannot be reduced to a complete graph of 4 nodes (the K4 clique). More intuitively, they are graphs that can be constructed by iterative application of the two basic operations:

- Sequential (or series) splitting: an arc $u=(x;y)$ is separated into two consecutive arcs, $u_1=(x;z)$ and $u_2=(z;y)$, by the introduction of an intermediate node z (Figure 9a). The *series* relation that binds u_1 and u_2 is denoted u_1+u_2.
- Parallel splitting: an arc $u=(x;y)$ is substituted by two parallel arcs, $u_1=(x;y)$ and $u_2=(x;y)$, with the same end points (Figure 9b). The *parallel* relation that binds u_1 and u_2 is denoted $u_1//u_2$.

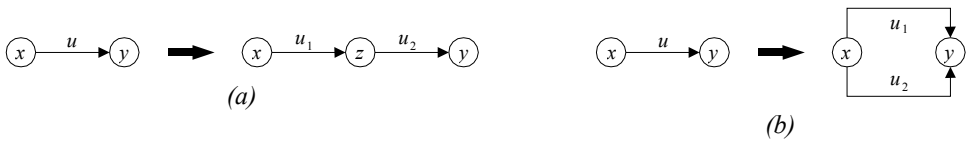

Figure 9 – *SP-relations.*

Series and parallel relations are gathered under the term *SP-relations*. The SP-relations are binary operations, so we can represent an SP-graph by a binary tree called *SP-tree*. Figure 10 shows an SP-tree of an SP-graph. Reference [BaMa03a] explains how to find such a tree in linear time.

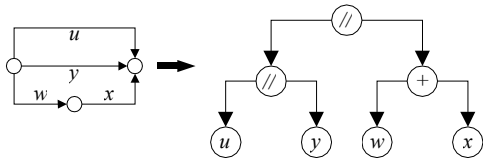

Figure 10 – *Example of SP-tree.*

As shown by Figure 11, most of Allen's temporal relations are series-parallel. Only the *overlaps* relation is not series-parallel: the arc linking the start of object *B* with the end of object *A* breaks the series-parallel property.

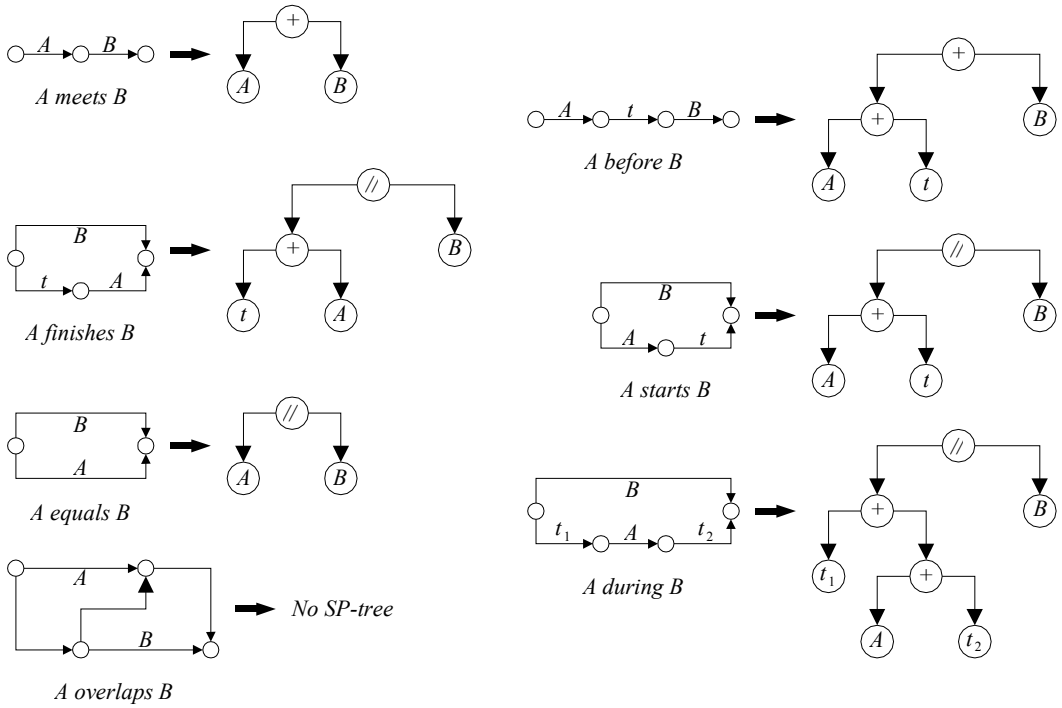

Figure 11 – *Allen relations and series-parallel property.*

4.4.2. The Aggregation Technique

The aggregation method, which allows solving the minimum-cost tension problem only on SP-graphs, has been introduced in [BaMa03a]. The algorithm works on an SP-tree T derived from an SP-graph G, recursively: considering an SP-relation, the algorithm 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 an SP-graph, it is obvious that an SP-graph has only one head node and only one tail node. Therefore, the *main tension* $\overline{\theta}$ of an SP-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 an SP-graph G must be defined. This function represents the cost of the optimal tension, where the main tension is forced to a given value:

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

As each function c_u is convex, the minimum cost function is indeed convex. We try now to find a procedure to build this function for an SP-relation. Let us consider two series-parallel subgraphs G_1 and G_2 , and suppose that their minimum cost functions C_{G_1} and C_{G_2} , respectively, are known. The minimum cost function $C_{G_1+G_2}$ of the SP-relation G_1+G_2 is given by:

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

As explained in [BaMa03a], this function can be built in linear time, needing precisely *O(p)* operations, where p is the sum of the pieces of both functions C_{G_1} and C_{G_2} (Figure 12a).

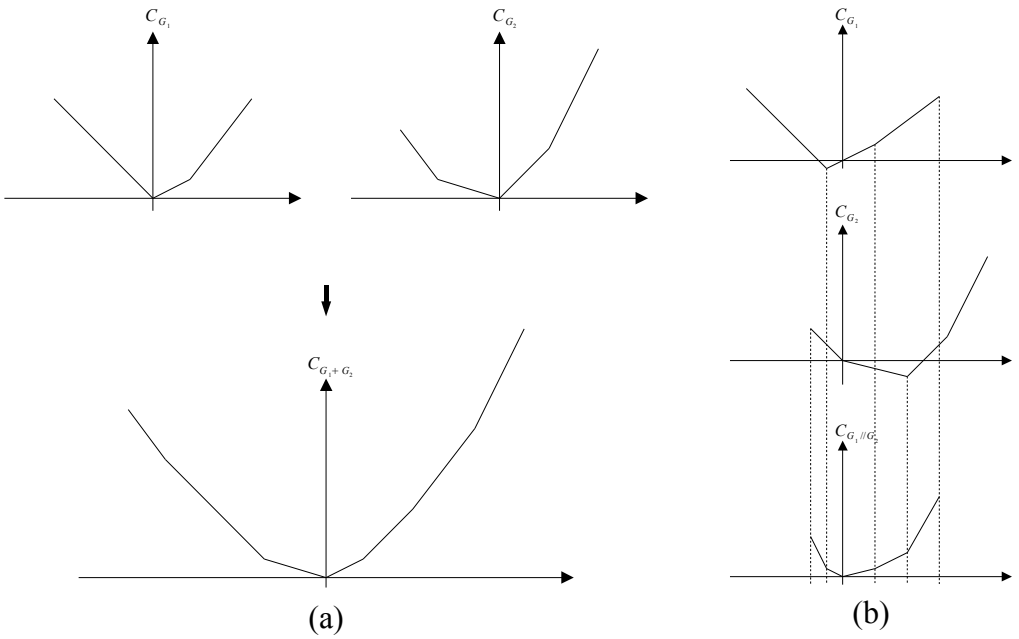

Figure 12 – *Aggregation technique.*

The minimum cost function $C_{G_1//G_2}$ of the SP-relation $G_1//G_2$ is:

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

This is a simple sum of functions that can be built also needing *O(p)* operations (Figure 12b). The full details of the aggregation technique are proposed in [BaMa03a]. However, the main procedure consists in starting from the leaves with a single SP-relation between two arcs, and building the minimum cost function associated with the SP-relation u_1+u_2 (or $u_1//u_2$). The procedure then goes on with another node of the tree, i.e., with another SP-relation, until it reaches the root node. At this moment, the minimum cost function of the whole graph is known, making it easy to identify the minimum cost value and thus to build the associate minimum cost tension.

4.4.3. Decomposition and Reconstruction

Hypermedia temporal graphs are very close to SP-graphs (e.g. the *overlaps* relation in Figure 11). However, they are quite more complex. Most of their structure is series-parallel, and a few constraints are added that break this property. We propose the notion of SP-perturbation, to measure how far they really are from the series-parallel graphs.

A *quasi SP-graph* or *QSP-graph* $G=(X;U)$ is such that the removal of a minimal subset of arcs, $U' \subset U$, from G makes the remaining graph $G'=(X;U\backslash U')$ series-parallel. The ratio $|U'|/|U|$ is called the *series-parallel perturbation*, or *SP-perturbation*, of graph G. This value indicates how many arcs of G are disturbing the series-parallel property of G. From this definition, any connected graph is a QSP-graph, but we prefer to use this term for graphs with a small SP-perturbation (in applications issued from the hypermedia field, 10% seems a satisfying threshold).

As the aggregation method cannot be used “as it is” on QSP-graphs, and the dual cost-scaling approach proves to be less efficient than the out-of-kilter on SP-graphs (see Table 1), we propose a method called reconstruction to solve the minimum-cost tension problem on QSP-graphs. This new approach combines the aggregation and the out-of-kilter techniques based on an SP-decomposition of the graph.

Decomposition Phase

We call *series-parallel component* or *SP-component* of a graph G an SP-subgraph of G. A *series-parallel decomposition*, or *SP-decomposition*, of a graph G is a set of arc-disjoint SP-components, whose union is G. [BaMa03b] discusses ways to obtain a "good" SP-decomposition that fits the reconstruction process.

The method starts with the search for an SP-decomposition D of a graph G. Then the minimum-cost tension problem is solved on each SP-component of D with the aggregation method. Thus, for each component D_u in D, its minimum cost function C_u and its optimal tension θ_u^* are known, so D_u can be seen as a single aggregated arc u with a convex piecewise linear cost function $c_u=C_u$ and a tension θ_u^*.

Reconstruction Phase

The method attempts next to put the SP-components back together. An iterative process consists in adding one by one the aggregated arcs into a new graph, rebuilding the original graph G. Starting with $H^0=(X^0;U^0)=(\varnothing;\varnothing)$, at each step an aggregated arc $u=(x;y)$ is added, i.e., $H^k=(X^{k-1}\cup\{x;y\};U^{k-1}\cup\{u\})$.

The newly added arc u may be out-of-kilter (and it is the only one) because even if its tension is optimal, its flow must be set to *0* in order to keep the flow conservation law on the whole graph H^k. Fortunately, as this arc is the only one out-of-kilter, repetitive search for a cycle or a cut to modify, respectively, either the flow or the tension of the arc u can be performed with few operations [BaMa03b]. Once the arc is in-kilter, the whole tension on H^k is optimal and another aggregated arc v can be added.

Tension Adjustment

However, adding an arc into the graph is not that obvious. Consider the aggregated arc $u=(x;y)$; the optimal tension θ_u of u may not be equal to the difference of potentials $\pi_y-\pi_x$. This can be achieved using the minimum cost function C_u of the arc u to optimally adapt the main tension of the SP-component aggregated behind u. This computation can be performed in linear time, as explained in [BaMa03b].

SP-Component Splitting

For the need of the reconstruction, we assume a partial order on the components of the SP-decomposition D, such that if the head node or the tail node of an SP-component D_u belong to an SP-component D_v, then $D_v<D_u$. Decomposition methods presented in [BaMa03b] prove that such an order exists for any graph. During the reconstruction phase, the SP-components are added following this partial order.

Another problem arises when adding an aggregated arc $u=(x;y)$ into the graph H^k: maybe the head node x and/or the tail node y of u are not present in H^k, because they are simply hidden in one of the aggregated arcs v of the graph H^k. Hence, the arc v should be replaced by the whole SP-component D_v it is hiding, making thus x and/or y visible. All arcs of the SP-component D_v are then brought back into the graph H^k (i.e., $H^{k+1}=H^k\cup D_v-\{v\}$). We call this operation the *splitting* of arc v.

The drawback of this technique is that the biggest SP-component, added behind a single aggregated arc at the first step, is certainly split at the second step, because nodes will obviously be needed for

the second added arc. This approach reveals to be extremely inefficient, as most of the arcs will be brought back in the graph only at the second step. Thus, we need a smarter way than totally splitting a SP-component. [BaMa03b] proposes a technique to split an SP-component into several pieces, reducing their numbers to the minimum necessary.

Conservation of the In-Kilter Property

After splitting an aggregated arc $u=(x;y)$, the resultant arcs must be in-kilter to keep the whole graph H^k optimal. Thus, knowing arc tensions, it is straightforward to find an interval in which arc flows must lie: indeed, with our assumptions, the kilter curve is a step function [BaMa03b].

For each arc v of the SP-component D_u, an interval $[e_v;f_v]$ can be defined. Let suppose an arc w from the head node x to the tail node y of the SP-component, with the capacity interval $[\varphi_u;\varphi_u]$, where φ_u is the flow of u in the whole graph H^k. Finding a flow φ_u for each arc v of the SP-component D_u turns to find a feasible flow φ in the SP-component (with the added loop arc w). Suitable methods to solve this classical network flow problem can be found in [AhMO93].

Algorithm

The following algorithm summarizes the whole process of the reconstruction method.

```
algorithm reconstruction:

find SP-decomposition D={D_u}_{u=1..k} ;
let H be an empty graph;

for all (SP-component D_u∈D ) do
 find minimum cost tension for D_u using aggregation;
 let u=(x;y) be the aggregated arc of D_u ;

 while (exists D_v with D_v<D_u and ( x∈D_v and y∈D_v )) do
  split D_v ;
  find tension for each arc of D_v using C_v ;
  find flow for each arc of D_v ;
 end while;

 add arc u in H;
 find minimum cost tension for H using out-of-kilter;
end for;

for all (aggregated arc u in H) do
 split D_u ;
 find tension for each arc of D_u using C_u ;
end for;
```

4.5. Computational Results

Extensive computational testing on medium- and large-scale temporal graphs has been performed to compare the behavior of the algorithms presented in this section for different objective functions. In the case of piecewise linear cost functions (which include the min-sum and min-max metric functions), the comparison could be extended to linear programming codes like GLPK[7]. Tables 1, 2

[7] http://www.gnu.org/software/glpk/glpk.html

and 3 present a practical comparison of methods with piecewise linear costs. The tables offer an idea of how the methods behave on SP-graphs (Table 1) and on QSP-graphs (Tables 2 and 3). Table 1 shows performances for SP-graphs when their size varies. Table 2 shows results when the size of the QSP-graphs varies. Table 3 points out the performances of the methods for various SP-perturbations.

Results are expressed in seconds on a Celeron 500 MHz processor under a Linux operating system. We used GNU C++ 2.95 compiler and its object-oriented features to implement the methods. The results are means of series of 10 tests on randomly generated graphs. Both $A = \max_{u \in U}(\theta_u^{\min}; \theta_u^{\max})$ and $B = \max_{u \in U}(c_u^1; c_u^2)$ are fixed to *1000*.

Nodes	Arcs	GLPK (s)	Kilter (s)	Dual (s)	Aggregation (s)
50	200	0.11	0.02	0.04	0.01
50	400	0.32	0.04	0.07	0.03
100	400	0.39	0.05	0.09	0.03
100	800	1.23	0.12	0.20	0.06
500	2000	9.66	0.91	1.55	0.15
500	4000	33.69	1.85	3.32	0.33
1000	4000	39.73	2.80	5.2	0.32
1000	8000	132.81	4.53	9.5	0.67

Table 1 – *Results on SP-graphs.*

On Table 1, the aggregation technique appears clearly to be more efficient than any of the other methods. However, the graphs are too ideal for real hypermedia synchronization cases.

Nodes	Arcs	GLPK (s)	Kilter (s)	Dual (s)	Reconstruction (s)
50	200	0.08	0.02	0.03	0.05
50	400	0.23	0.04	0.07	0.10
100	400	0.25	0.07	0.09	0.12
100	800	1.05	0.16	0.18	0.24
500	2000	7.56	1.48	1.57	0.95
500	4000	34.79	3.21	3.24	1.97
1000	4000	35.51	5.27	4.54	2.69
1000	8000	249.61	11.43	8.70	5.62

Table 2 – *Results on QSP-graphs, graph size influence (SP-perturbation=4%).*

On Table 2, with a SP-perturbation set to 4%, the reconstruction method appears to be the most efficient on large graphs (over 500 nodes and 2000 arcs), whereas the single out-of-kilter method is faster on small ones (below 500 nodes and 2000 arcs). The way the graph is decomposed in SP-components and how the components are split during the reconstruction process is a key to good performances. Further information can be found in [BaMa03b].

SP-Perturbation (%)	GLPK (s)	Kilter (s)	Dual (s)	Reconstruction (s)
2	12.58	1.85	2.28	1.13
4	17.65	2.21	2.45	1.45
6	21.63	2.63	2.27	1.78
7	26.50	2.80	2.11	2.03
8	25.17	2.83	2.12	2.26
9	28.51	3.07	2.11	2.42

10	28.32	3.27	2.12	2.78
15	37.38	3.86	2.06	4.41
20	45.05	4.47	2.02	6.43

Table 3 – *Results on QSP-graphs, SP-perturbation influence (500 nodes, 3000 arcs).*

Only the out-of-kilter method can deal with convex differentiable costs. Table 4 shows its behavior when the objective function is a least squares function (see Annex 1) and the precision of the results varies. The performances indicate that such costs can only be considered at compile time.

Precision 1/n	**Kilter (s)**
1	1.89
10	2.61
100	3.43
1000	5.18
10000	12.25

Table 4 – *Results for convex costs (100 nodes, 400 arcs).*

5. Prefetching and Reservation Issues

At compile time, mismatches can be smoothed out by adaptation mechanisms, improving the resultant document appearance. However, unpredictable behavior can be handled only during runtime. Unfortunately, during runtime, mismatches (event occurrence different from its expected time in the presentation plan) are detected only when part of the document has already been presented. This delay in detection limits the formatter ability to smooth out mismatches. Even though the synchronization can be recovered, part of the presentation will lose quality. These are reasons why a formatter needs not only supporting both compile-time and runtime adjustments, but also having some mechanism that anticipates actions specified in the document, reducing the probability of adjustments during runtime.

The main idea is to build a prefetching plan. Based on this plan, document objects (their contents) are prefetched before they are requested to be played, even in the case of random requests. The addition of a prefetching/caching mechanism in hypermedia formatters is important not only to reduce the probability of adjustments during document runtime, but also to minimize object presentation lags and compensate content jitter.

The prefetching plan can be built based only on the presentation plan information. However, it is important that it also takes into account context aware information, such as the performance parameters of the Hypermedia System SOE − service offering environment (Section 1) − (delays and jitters imposed by the network, operating systems, media players, and other components; available bandwidth; etc.), history of past requests, object lengths, availability for resource reservation, etc. Note that the prefetching plan may indeed be a timeline as opposed to the presentation plan.

SOE resource reservation information can be obtained from in advance QoS negotiation mechanisms. Resource reservation in advance will return the best moment to accomplish the prefetch and to reserve resources in the Hypermedia System SOE. However, neither network signaling protocols nor operating systems do this kind of reservation in current implementations. Thus, heuristics must be employed to build the pre-fetching plan, based only on assumptions about the SOE performance parameters. Several strategies for building the plan were proposed in the literature [JeHK97, KhTa01].

Analogous to the presentation plan, the prefetching mechanism can be divided into two phases: compilation and execution. At compile time, the prefetching plan is generated. During execution phase, the prefetching plan actions are scheduled and monitored, triggering adjustments when possible mismatches occur.

If the prefetching plan is built without resource reservation in advance, a QoS request for each object must be made based on the time specified for the object prefetch. If a QoS request cannot be provided, or if any violation on the QoS established contract occurs, the prefetching plan should be corrected.

Some Hypermedia SOEs, however, do not allow QoS negotiation on all their resources (or none of them); for example, a hypermedia system using a communication provider implemented by a network without QoS guarantees. In this case, the real prefetching parameters for each object should be monitored. If a mismatch between these parameters and those used to estimate the prefetch time occurs, the prefetching plan should also be corrected.

In any case demanding corrections, if they are possible or not, the presentation plan must be adjusted accordingly. On the other hand, at any time and for any reason, once the presentation plan is modified, the prefetching plan must be adjusted.

6. Execution Time Adaptation

As aforementioned, during compile time, several unpredictable factors may prevent building the whole presentation plan, such as rules to choose alternatives that must be evaluated during runtime, object start time that depends on a user interaction, live object with unpredictable duration, etc.

During runtime, context information is in continuous change, especially those associated to SOE performance parameters, which may cause changes in the presentation plan.

As pointed out in the previous section, the presentation plan can also be adjusted due to necessary adjustments in the prefetching plan. If in any moment the loading of document objects cannot be guaranteed in restricted accordance to the presentation plan, this plan should be adjusted.

Presentation events coming from exhibition tools (video, audio, text and other content players) should also be monitored and compared with their predicted time as established by the presentation plan, in order to prevent loss of synchronization among document objects.

In general, if any change or violation that impairs the presentation is detected, the presentation plan should be adjusted. Every time the presentation plan changes, adaptations may be required, particularly those to maintain the temporal consistency of the document. Note that in this case, adaptations are made on the fly, and must be rendered as fast as possible. Sometimes, elastic time computation may be necessary to resolve or minimize the problem (modifying content presentation bit rate or duration).

7. Elastic Time Computation During Execution Time

7.1. Mismatches

As aforementioned, during runtime, mismatches are detected only when a portion of the document has already been presented. This delay in detection limits the ability of the formatter to smooth out mismatches. During compile time, the possible mismatches can be smoothed out improving the

appearance of the resultant document. However, only at runtime, unpredictable behavior can be handled. These are the reasons why a formatter needs to support both compile-time and runtime adjustments.

When runtime mismatches cause adjustments in event duration, the expected duration of the subsequent events should be recomputed, since the calculus made at compile time may be no more valid. The formatter should run again the elastic time algorithm, now considering the exact duration of the already presented events and the real-time computation requirement. Thus, the elastic time computation should be scalable, in the sense that the algorithm should take profit of the results computed before any mismatch happens, in order to improve the calculus performance that must be done on-the-fly.

Depending on the severity of the mismatches, the real-time calculus may require heuristics, since the optimum calculus can be very time expensive. However, looking back at the results of Section 4, it appears that for common sizes of hypermedia documents (less than 100 temporal constraints), the compile-time algorithms are fast enough to be used in real-time.

Nevertheless, there are situations where it is possible to use smarter techniques to keep the synchronization optimal and avoid recomputing the whole schedule. Let us consider the temporal graph of Figure 13, and suppose that an optimal tension has already been computed for that graph. We identify three different cases of mismatches that can occur during the presentation:

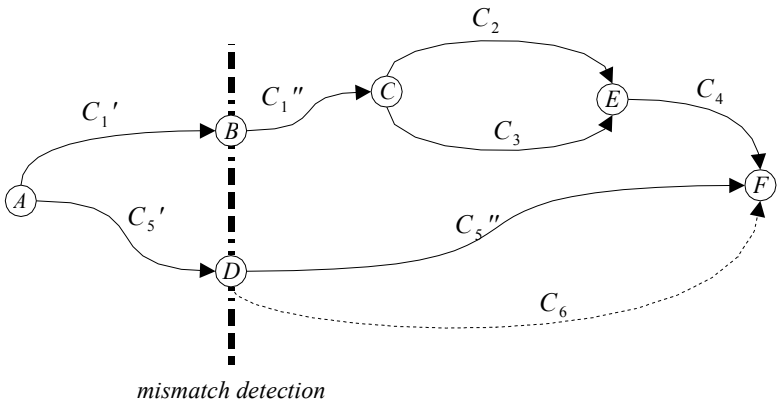

Figure 13 – *Mismatch examples.*

- A delay appears in the presentation of one media object, modifying the length of one active event, for instance in chain C_1 in Figure 13.
- An unpredictable event appears, for instance the user clicks on a button that starts a new chain of presentation, for example C_6 in Figure 13.
- The user decides to stop a chain, e.g. C_5 in Figure 13.

7.2. Recovering an Optimal Schedule

Note that the out-of-kilter method is scalable. Let consider a single delay in chain C_1, and suppose the detection of the delay appears at the time events B and D should start. That means chain $C_1''+C_2//C_3+C_4$ must be shrunk, and/or chain C_5'' stretched. We choose an arc u, for instance the first arc of C_1'' and decrease its tension so the tension of $C_1''+C_2//C_3+C_4$ equals the tension of C_5''. Thus the arc u is not in-kilter anymore, and maybe is not even compatible (i.e., out of its tension bounds). The idea is to use the improvement procedure of the out-of-kilter method (cf. Section 4) and apply it to arc u until it becomes in-kilter. For arcs that have already been presented, the tension

bounds are fixed to their past duration. As no other arc of the graph is out-of-kilter, at the end of the procedure, either the arc is still not compatible, which means that no solution exists for the set of synchronization constraints, or the arc is on its kilter curve and the synchronization schedule is optimal. Only a few iterations of the improvement procedure are requested (which is a fast procedure in *O(Km)* operations, where *m* is the number of arcs in the graph and *K* is a constant established from the data of the synchronization constraints [Bach03]).

In the case of an unpredictable event that adds a chain C_6 in the temporal graph at event *D*, the strategy will be close to the former one. An arc *u* of chain C_6 is chosen and its tension adapted so the tension of C_6 equals the tension of C_5''. The arc *u* is still the only one not in-kilter because, at compile time, an optimal schedule of C_6 has been determined, totally independent of the other chains of the temporal graph. In this situation too, if a synchronization is possible, an optimal schedule is found.

If the user decides to stop chain C_5 at event *D*, the remaining presentation will be correctly, but not optimally, presented, according to the new temporal graph. The chain C_5'' is removed, but the flow conservation law is not satisfied anymore for the nodes *D* and *F*. To recover the correct balance, a fictitious arc *u* is added between *D* and *F* with a flow equal to the flow of C_5'', a null cost and no tension bounds. The only out-of-kilter arc is *u*, but in opposition to the other cases, a synchronization is always possible. Like in the previous situations, an optimal schedule can be found with a few iterations of the improvement procedure applied on arc *u*.

7.3. Series-Parallel Chains and Aggregation

The study on SP-graphs of Section 4 can be interesting for the real-time computation of an optimal schedule. As explained in that section, the aggregation technique, which solves the minimum-cost tension problem on a SP-graph, does not only provide an optimal tension, but also provides a minimum cost function and associate information that allows, in linear time, to adjust the presentation of a whole SP-graph optimally. To sum up, a series-parallel chain can be replaced by a single arc (of course, with a much more complex cost function, but still a step function fully adapted to the out-of-kilter technique), as presented for the reconstruction technique in Section 4.

Therefore, it can be considered to use the aggregation technique at compile time to compute the minimum cost function of all the elementary series-parallel chains of the temporal graph, and try to use them during runtime to speed-up the improvement process of the out-of-kilter. This technique seems promising but needs to be thought through to identify situations where the gain is significant.

8. Related Work

The Firefly system formatter [BuZe93a, BuZe93b] has introduced the idea of temporal chains (one main schedule and zero or more auxiliary schedules). It has also separated the presentation control in two phases, compilation and execution. One of the main drawbacks in Firefly is that its execution plan (temporal graph) is a simple timeline of presentation actions, making difficult the implementation of runtime adjustments.

Elastic time computation has been addressed in Firefly and Isis [KiSo95]. However, they only consider models with piecewise linear convex cost functions (Firefly model is based on potentials, whereas Isis model is based on cycles). They propose to solve them with the simplex method, which is suitable only for compile-time adaptation. As discussed in Section 4, our proposal extended these models to general convex cost functions and gave a complete description of the optimality conditions. Section 4 and 7 also discussed how our proposals can be used in runtime computations.

Moreover, Firefly and Isis formatter architectures do not preview the existence of prefetching mechanisms.

Madeus [LaSR02] is an authoring and presentation system for interactive multimedia documents based on spatio-temporal constraints among media objects. The system also offers temporal flexibility through media object duration specifications, which can be defined as an interval, establishing a lower and an upper limit value.

In its compile phase, Madeus formatter computes for the predictable (or *controllable*) objects their expected duration (the *nominal duration*) and builds a temporal graph, named HSTP (*Hypergraph of Simple Temporal Problems*). This graph can be considered as the Madeus presentation plan.

During the execution phase, the formatter monitors the effective duration of each media object and compares them with the expected values. When any deviation that can cause a future synchronization loss is perceived, the Madeus formatter applies an adjustment algorithm, but without considering optimization metrics. The main goal is to quickly find a new presentation arrangement that satisfies the duration interval constraints. The Madeus formatter also monitors the duration of unpredictable events in order to verify if their presentation end require duration adjustments.

The Madeus formatter does not deal with prefetching and context-oriented adaptation issues. Although offering an algorithm for elastic time adaptation on the fly, the solution uses a greedy approach, without concerning about the qualitative result.

Similar to the other formatters, CMIFed formatter (named CMIFed Player) [Ross93] builds a graph of temporal dependencies to identify the moments to fire the presentation actions. Basis for GRiNS[8] player implementation for SMIL [W3C01] document presentation, the system can perform some context-based adaptations, but no elastic time adjustment is mentioned. In SMIL version 2.0 the introduction of intervals for defining object duration allows elastic time computations, but like Madeus, no metrics are available for measuring the adapted presentation quality. Actually, we are now investigating how to extend SMIL to accommodate elastic time computation algorithms.

Jeong et al. [JeHK97] developed a mechanism for pre-scheduling multimedia presentations. However, the proposal only considers documents exclusively based on predictable relationships and predictable media object durations. During document runtime phase, there is a monitor that compares the object actual prefetching duration with the expected duration previewed in the plan. If the actual duration overcomes the predicted one, the system runs an instant scheduling algorithm for recalculating object presentation durations, in order to maintain the temporal segment synchronization. When necessary, the algorithm sacrifices static media objects, shrinking their durations. No elastic time computation based on metrics are available for measuring the adapted presentation quality.

Finally, it is important to comment that although some proposals discussed in this section deal with the elastic time computation, none of them comments about how to apply these adjustments in continuous-media contents, except by the simple cut or repetition of their last samples. HyperProp players, exemplified by the “Expressive Talking Heads” [Rodr04] is a step towards this goal.

[8] http://www.oratrix.com

9. Final Comments and Future Work

This paper proposes new algorithms for elastic time computation. One of them is based on the "out-of-kilter" method for minimum-cost tension problems, which allows much more flexibility to model real-life situations. The algorithm is general, in the sense that it can deal with general convex cost functions.

Another approach, the "dual cost-scaling" is also proposed, which converts the problem into a minimum-cost flow problem solved with the well-known "cost-scaling" method. This method is better adapted for big graphs at compile time.

Both algorithms are extensively compared and both outperform the simplex method, used in previous work, when dealing with piecewise linear costs.

In order to be used during runtime, the elastic time computation should be scalable, in the sense that the algorithm should take profit of the results computed before any mismatch happens, in order to improve the calculus performance that must be done on-the-fly. This is the case of the out-of-kilter method, which can be adapted to runtime issues, as discussed in Section 7.

A new method offering similar performances and based on specific structure of temporal graphs, called "aggregation", is also presented. Working only with piecewise linear convex cost functions, the aggregation-and-reconstruction technique manages the series-parallel aspect of the temporal constraints. The technique provides an efficient resolution time that makes the method not only suitable to be used in the compile-time algorithm, but also to develop runtime heuristics, opening new possibilities for runtime computation.

Turning back to the five goals presented in the beginning of Section 4, it is possible to discuss directions for future work.

The first goal is an NP-Complete problem, requiring the use of heuristics and approximations, which is a research already under development [MeRS02]. A solution for series-parallel graphs has also been developed, using the aggregation technique [Bach04]. Although the theoretical complexity of the method is exponential, results show a good practical behavior: for a 50 nodes and 100 arcs graph, the problem is solved in 0.5 second on a 1800 XP+ processor.

The results presented in this paper allow to conclude that the second goal can be easily reached for continuous cost functions. However, for discrete functions it is also an NP-Complete problem that must be investigated.

The third goal is naturally reached by the proposed temporal graph, by adding two nodes, N_1 and N_2, and three arcs: the first linking N_1 to N_2 and having the desired presentation duration, and the other two linking the head node to N_1 and N_2 to the tail node, respectively. The fourth goal was also solved by the temporal graph. To obtain the minimum and maximum possible duration for a given presentation, the idea is similar to the previous one, but now, instead of fixing the N_1 to N_2 arc duration, one should set its feasibility as indeterminate.

Goals concerning on-the-fly issues are discussed in Section 7, in special using series-parallel chains and aggregation. It can be considered to use the aggregation technique at compile time to compute the minimum-cost function of all the elementary series-parallel chains of the temporal graph, and try to use them during runtime to speed-up the improvement process of the out-of-kilter. This technique seems promising but needs to be thought through to identify situations where the gain is significant.

Regarding cost function specification, SMIL 2.0 introduces intervals for defining object duration, allowing elastic time computations, but no metrics are available for measuring the adapted presentation quality. We are now investigating how to extend SMIL to accommodate elastic time computation algorithms.

Finally, static media object duration can be easily adjusted but the same is not true with continuous media. We are also concerned in how to apply duration adjustments in continuous-media contents, besides the simple cut or repetition of their last samples. HyperProp tools for synthesized audio and video, exemplified by the ETHs [Rodr04] is a step towards this goal, as aforementioned. We are now working on multimedia content analysis for content adaptation for MPEG audio and video, and audio analysis for MPEG video adaptation.

The final goal is to integrate the elastic time computation algorithms with players that are able to perform the calculated adjustments without perceptual losing (or losing a minimum) of presentation quality.

References

[AhMO93] Ahuja R.K., Magnanti T.L., Orlin J.B. Network Flows, Prentice Hall, 1993.

[AhHO99] Ahuja R.K., Hochbaum D.S., Orlin J.B. Solving the Convex Cost Integer Dual Network Flow Problem. *Lecture Notes in Computer Science*, volume 1610, 1999, pp. 31-44.

[Alle83] Allen J.F. "Maintaining Knowlegde about Temporal Intervals". *Communications of the ACM*, 26(11), November 1983, pp. 832-843.

[Bach03] Bachelet B. Modélisation et optimisation de problèmes de synchronisation dans les documents hypermédia. PhD Thesis, Université Blaise Pascal, Clermont-Ferrand, France. *Available in http://frog.isima.fr/bruno/?doc=phd_thesis*.

[BaMa03a] Bachelet B., Mahey P. Minimum Convex-Cost Tension Problems on Series-Parallel Graphs. *RAIRO Operations Research*, volume 37, 2003, pp. 221-234.

[BaMa03b] Bachelet B., Mahey P. Minimum Convex Piecewise Linear Cost Tension Problem on Quasi Series-Parallel Graphs. Research Report RR-03-19, LIMOS, Blaise-Pascal University, Clermont-Ferrand, France, 2003. *Available in http://www.isima.fr/limos/publi/paper/2003/RR0319.pdf.*

[Bach04] Bachelet B. Aggregation Approach for the Minimum Binary Cost Tension Problem. Research Report RR-04-08, LIMOS, Blaise-Pascal University, Clermont-Ferrand, France, 2004. *Available in http://www.isima.fr/limos/publi/paper/2004/RR0408.pdf.*

[BeGh62] Berge C., Ghoula-Houri A. Programmes, jeux et réseaux de transport. Dunod, 1962.

[BoKW99] Boll S., Klas W., Wandel J. "A Cross-Media Adaptation Strategy for Multimedia Presentation". *ACM Multimedia*, Orlando, USA, October 1999.

[BuZe93a] Buchanan M.C., Zellweger P.T. "Automatic Temporal Layout Mechanisms", *ACM International Conference on Multimedia*, California, USA, 1993, pp. 341-350.

[BuZe93b] Buchanan M.C., Zellweger P.T. "Automatically Generating Consistent Schedules for Multimedia Documents". *ACM Multimedia Systems Journal*, Springer-Verlag, 1(2), September 1993, pp. 55-67.

[BSRS04] Bulterman D., Soares L.F.G., Rodrigues R.F., Soares Neto C.S, "QoS Provisioning in Adaptive Hypermedia Formatters", Technical Report of TeleMidia Lab, PUC-Rio, Rio de Janeiro, Brazil, May 2004.

[DeSA01] Dey A.K., Salber D., Abowd G.D. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications". *Human-Computer Interaction (HCI) Journal - special issue on Context-Aware Computing*, 16(2-4), 2001, pp. 97-166.

[DiSé94] Diaz M., Sénac P. "Time Stream Petri Nets, a Model for Timed Multimedia Information", *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, Zaragoza, 1994, pp. 219-238.

[DuKe95] Duda A., Keramane C. "Structured Temporal Composition of Multimedia Data". *Proceedings of the IEEE International Workshop on Multimedia Database Management Systems*, Blue Mountain Lake, USA, August 1995.

[Fulk61] Fulkerson D.R. "An Out-of-Kilter Method for Minimal Cost Flow Problems". *SIAM Journal on Applied Math.*, vol. 9, 1961, pp. 18-27.

[Hadj96] Hadjiat M. Problèmes de tension sur un graphe – Algorithmes et complexité. PhD Thesis, Université de la Méditerrannée, Marseille, France, 1996.

[JeHK97] Jeong T., Ham J., Kim S. "A Pre-scheduling Mechanism for Multimedia Presentation Synchronization". *IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Canada, 1997, pp. 379-386.

[KhTa01] Khan J., Tao Q. "Prefetch Scheduling for Composite Hypermedia", *IEEE International Conference on Communications - ICC 2001*, Helsinki, Finland, June 2001.

[KiSo95] Kim M., Song J. "Multimedia Documents with Elastic Time". *Proceedings of ACM Multimedia'95*, São Francisco, USA, November 1995.

[KlGF99] Klas W., Greiner C., Friedl R. "Cardio-OP – Gallery of Cardiac Surgery". *IEEE International Conference on Multimedia Computing and Systems (ICMS'99)*, Florence, Italy, June 1999.

[LaSR02] Layaïda N., Sabry-Ismail L., Roisin C. "Dealing with uncertain durations in synchronized multimedia presentations". *Multimedia Tools and Applications Journal*, Kluwer Academic Publishers, 18(3), December 2002, pp. 213-231.

[LiGh90] Little T., Ghafoor A. "Synchronization and Storage Models for Multimedia Objects". *IEEE Journal Selected Areas of Communications*, 8(3), April 1990, pp. 413-427.

[MeRS02] Medina M.T., Ribeiro C.C., Soares L.F.G. "Automatic Scheduling of Hypermedia Documents with Elastic Times". *Parallel Processing Letters*, 2002, pp. 45-59.

[MuSo02] Muchaluat-Saade D.C., Soares L.F.G. "XConnector & XTemplate: Improving the Expressiveness and Reuse in Web Authoring Languages", *The New Review of Hypermedia and Multimedia Journal*, Taylor Graham, vol. 8, 2002, pp. 139-169.

[Much03] Muchaluat-Saade D.C. "Relations in Hypermedia Authoring Languages: Improving Reuse and Expressiveness", PhD Thesis, Informatics Department, PUC-Rio, Rio de Janeiro, Brazil, March 2003. *Available in ftp://ftp.telemidia.puc- rio.br/pub/docs/theses/2003_03_muchaluat_ingles.pdf*

[PéLi96] Pérez-Luque M.J., Little T.D.C. "A Temporal Reference Framework for Multimedia Synchronization". *IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication)*, 14(1), January 1996, pp. 36-51.

[Pla71] Pla J.M. An Out-Of-Kilter Algorithm for Solving Minimum Cost Potential Problems. *Mathematical Programming*, 1, 1971, pp. 275-290

[Rock84] Rockafellar R.T. Network Flows and Monotropic Optimization. J. Wiley. 1984.

[RoSo03] Rodrigues R.F., Soares L.F.G "Inter and Intra Media-Object QoS Provisioning in Adaptive Formatters", *ACM Symposium on Document Engineering*, Grenoble, France, November 2003.

[Rodr04] Rodrigues R.F., Lucena-Rodrigues P.S., Feijó B., Velho L., Soares L.F.G. "Cross-Media and Elastic Time Adaptive Presentations: the Integration of a Talking Head Tool with a Hypermedia Formatter", Technical Report of TeleMidia Lab, PUC-Rio, Rio de Janeiro, Brazil, March 2004.

[Ross93] van Rossum G., Jansen J., Mullender K.S., Bulterman D. "CMIFed: A Presentation Environment for Portable Hypermedia Documents." ACM Multimedia, Anaheim, USA, August 1993.

[SoRM00] Soares L.F.G., Rodrigues R.F., Muchaluat-Saade D.C. “Modeling, Authoring and Formatting Hypermedia Documents in the HyperProp System”, *ACM Multimedia Systems Journal*, Springer-Verlag, 8(2), March 2000, pp. 118-134.

[W3C01] World-Wide Web Consortium (W3C). “Synchronized Multimedia Integration Language (SMIL 2.0) Specification”. *W3C Recommendation*, August 2001. *Available in http://www.w3.org/TR/smil20.*

Annex A

The theoretical aspects of the minimum-cost tension problem described in Section 4 are illustrated here on a very simple example, corresponding to the synchronization of two events. The three basic metrics (min-sum, min-max and least-squares) will be compared for the following model in R^2:

$$\text{Minimize} \quad c_1(\theta_1)+c_2(\theta_2)$$

$$\text{Subject to} \quad \theta_1-\theta_2=0$$

$$\theta_1^{\min} \le \theta_1 \le \theta_1^{\max}$$

$$\theta_2^{\min} \le \theta_2 \le \theta_2^{\max}$$

We suppose $\hat{\theta}_1 < \hat{\theta}_2$, where $\hat{\theta}_1$ and $\hat{\theta}_2$ are the ideal durations, and let $e = \hat{\theta}_2 - \hat{\theta}_1$.

A.1. Min-Sum Criterion

Lets consider the objective function $\min\left|\hat{\theta}_1-\theta_1\right|+\left|\hat{\theta}_2-\theta_2\right|$. It represents the l_1-distance between the ideal solution $I=(\hat{\theta}_1;\hat{\theta}_2)$ and the constraint $\theta_1-\theta_2=0$. Looking at Figure 14, it appears that the optimal solution is not unique, but a continuous set of possibilities. In fact, fixing the tension θ_1 between $\hat{\theta}_1$ and $\hat{\theta}_2$, the tension θ_2 is also deduced between $\hat{\theta}_1$ and $\hat{\theta}_2$. In Figure 14, the solutions are represented by the full segment $[(\hat{\theta}_1;\hat{\theta}_1);(\hat{\theta}_2;\hat{\theta}_2)]$.

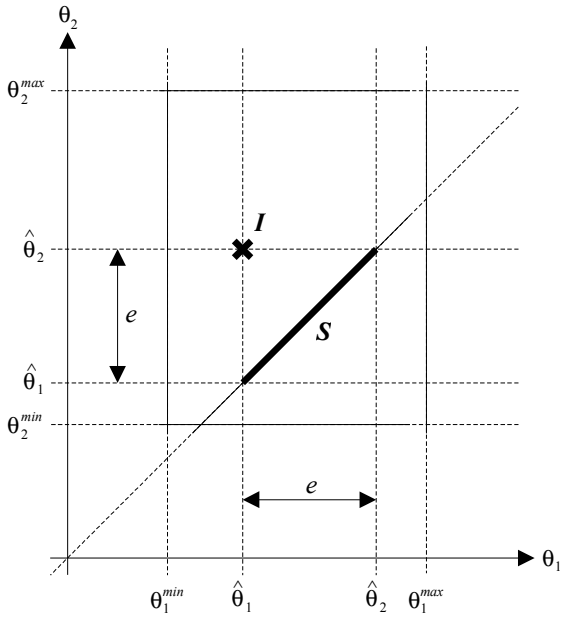

Figure 14 – *Optimal solution example, with the min-sum criterion.*

Observe that solutions where only one object is distorted exist, for instance $\theta_1 = \hat{\theta}_1, \theta_2 = \hat{\theta}_2 - e = \hat{\theta}_1$.

A.2. Least-Squares Criterion

Lets consider the objective function $\min\left(\hat{\theta}_1-\theta_1\right)^2+\left(\hat{\theta}_2-\theta_2\right)^2$. It represents the euclidean distance (norm l_2) between the ideal solution $I=(\hat{\theta}_1;\hat{\theta}_2)$ and the constraint $\theta_1-\theta_2=0$. As that norm is strictly convex, the optimal solution is always unique. It is the orthogonal projection of $(\hat{\theta}_1;\hat{\theta}_2)$ on the constraint $\theta_1-\theta_2=0$. In Figure 15, the solution is represented by point *S*, where $\theta_1=\hat{\theta}_1+e/2, \theta_2=\hat{\theta}_2-e/2$. Note also that both objects are equally distorted.

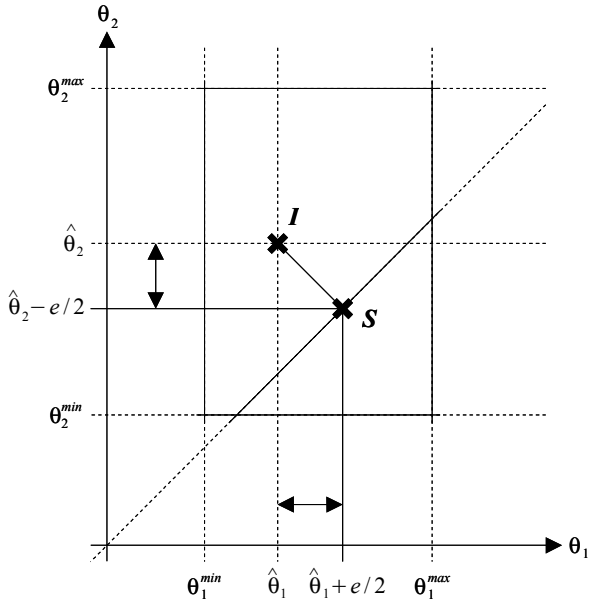

Figure 15 – *Optimum solution example, with the least-squares criterion.*

A.3. Min-Max Criterion

Let consider the objective function $\min\max\left\{\left|\hat{\theta}_1-\theta_1\right|;\left|\hat{\theta}_2-\theta_2\right|\right\}$. Here we try to minimize the maximal absolute distance (norm l_∞) between the ideal solution $I=(\hat{\theta}_1;\hat{\theta}_2)$ and the constraint $\theta_1-\theta_2=0$. It appears that the optimal solution is unique (in this example, but not in the general case) and it is the same as for the least-squares criterion, i.e. point *S* in Figure 15.