Enforcing Structure on Temporal Sequences: the Allen Constraint

Pierre Roy, Guillaume Perez, Jean-Charles Régin, Alexandre Papadopoulos, François Pachet, Marco Marchini

Sony CSL Paris
6, rue Amyot
75 005 Paris, France roypie@gmail.com

Abstract. Recent applications of constraint programming to entertainment, *e.g.,* music or video, call for global constraints describing the structure of temporal sequences. A typical constraint approach is to model each temporal event in the sequence with one variable, and to state constraints on these indexed variables. However, this approach hampers the statement of constraints involving events based on temporal position, since the position depends on preceding events rather than on the index. We introduce ALLEN, a global constraint relating event indexes with temporal positions. ALLEN maintains two set-variables: the set of events occurring at a position defined by an Allen relation, and the set of their indexes. These variables enable defining structural and temporal synchronization properties that cannot be stated on indexed variables. We show that a model based on a local scheduling approach does not solve the problem, even for very small instances, highlighting the need for complex filtering. We present a model that uses Multi-valued Decision Diagrams (MDDs) to compile the ALLEN constraint. We show that this model can be used to state and solve two complex musical tasks: audio track synchronization and musical score generation.

Keywords: Global constraints, temporal sequences, music, MDD

1 Introduction

Many difficult combinatorial problems consist in arranging sequences of events in time, subject to *horizontal* and *vertical* constraints. These constraints are expressed on the *temporal position* of events. Horizontal constraints relate events in the same sequence, but occurring at different positions. Vertical constraints relate events occurring simultaneously, *i.e.,* at the same position in different sequences. This is similar to shceduling problems, such as job-shop scheduling, in which tasks are performed on machines according to sequencial and resource constraints. The combination of horizontal and vertical constraints make these problems extremely difficult to solve: the job-shop scheduling problem is notoriously among the hardest combinatorial problems.

A typical constraint programming approach to generating such sequences is to define a variable for each item of the sequence, and to post constraints on these variables. Temporal sequences challenge this model, since the position of an event is determined by the duration of all the preceding events, and so is only weakly dependent on its index. It is therefore difficult, if not impossible, to express temporal properties using constraints on item variables.

This problem appears naturally in application domains related to entertainment [1–4]. Structural properties usually involve long-range dependencies between events. Deep learning approaches attempt precisely at capturing these dependencies in a statistical model, to reproduce them during classification or sampling. However, the representations of structure in statistical models is not explicit, making them inappropriate for specifying hard constraints on sequences. In the next section, we describe a typical example of a structural constraint occurring in the generation of lead sheets, a type of musical notation.

Lead Sheet Generation

A lead sheet is a representation of a musical piece commonly used in popular music and consisting of a melody with chord labels on top, as shown on Figure 1 (extract of *A Day in the Life* by Lennon / McCartney).

An important aspect of this lead sheet is that melodic patterns are distributed according to a temporal structure. For example, the pattern of bars 1-2 is repeated at bars 5-6. This type of structures is commonplace in popular music. To generate lead sheets with a similar temporal structure similar, a standard CP approach is to define one variable per note. However, notes have a different duration: bar 1 contains eight short notes (including the rest) and bar 2 contains only one long note. Consequently, there is no direct correspondence between the index of a note and its temporal position. This makes it hard to post constraints stating that the first two bars should be repeated two bars later, regardless of their number of notes; or any other constraint of the same kind. In Section 6, we show that our approach yields a practical solution to this problem.

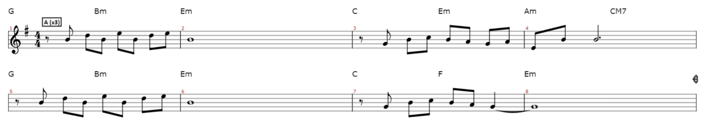


Fig. 1. The first 8 bars of *A Day in the Life* by John Lennon and Paul McCartney

Automatic Accompaniment Generation

We describe and evaluate our technical contribution on the generation of musical accompaniment from audio multitrack recordings. We chose this example as it has immediate applications for computer generated musical improvisation or accompaniment generation, as mentioned in Section 5.1.

The task consists in generating a new multi-track audio accompaniment by reusing an existing multi-track recording. The original tracks are segmented into chunks, using an onset detector [5]. Then new tracks are generated by recombining chunks, using concatenative synthesis [6]. Chunks in the generated tracks may appear in a different order than in the original track and may be used any number of times. Such a scheme involves several types of constraints: On the one hand, we have to constrain each track to avoid awkward chunk transitions, by allowing transitions that are similar to the transitions in the training corpus. The similarity is measured using acoustic features, see Section 5.1. On the other hand, to prevent the tracks from "drifting" from one another, *e.g.,* one track becomes increasingly louder while another track fades out, we have to *synchronize* the tracks at regular points in time, for instance at the onset of every bar.

This problem raises the same issue as lead sheet generation. The chunks have different durations, therefore the index of a chunk and its temporal position are not directly depending on one another.

Approaches

In the examples above and, more generally, in many interactive or content generation applications, we need to specify sequences with structural properties that cannot be inferred using statistical models. In the first example, the temporal structure in a lead sheet is not a statistical property that can be inferred from a set of examples, but is rather explicitly imposed, for instance by a user.

Constraint programming provides an ideal way of enforcing structure on sequences. However, as we highlighted earlier, we cannot state structural constraints on events based on their index alone.

Adopting a position-based model, in which variables represent events of smaller, *atomic duration* whereby longer objects are made up of several consecutive variables, solves this issue. For a given total duration, a fixed number of variables are defined and therefore indexes correspond to temporal positions. This requires discretizing time into a grid of equal-duration slices, small enough so that all events are aligned with the grid. In this model, the number of variables is considerably larger than the number of events in the generated sequences: if durations are expressed as fractions of the longest event, the atomic duration decreases with the *least common multiple* of the denominators, whose growth is exponential [7]. Hence, the size of the grid may be exponentially smaller than the event lengths, creating an intractable number of variables. Moreover, the position-based model requires additional horizontal constraints to aggregate atomic events to form longer objects. These constraints are not easy to specify in general. This approach is therefore not applicable in many real problems.

Several frameworks using constraint propagation make inferences about temporal relations from a qualitative [8] or quantitative standpoint [9]. The computational efficiency of these approaches is very limited in the general case, but they offer a precise and powerful representation of relations between times events.

Our Contribution

Allen [8] introduced an algebra with 13 binary relations between time intervals for temporal reasoning. We use this algebra as a language to express temporal positions and introduce the Allen global constraint, which defines variables corresponding to a given Allen relation. Technically, for a given time interval t and a given Allen relation $\mathcal{R}$, Allen maintains two set-variables: the set of events and the set of variable indexes satisfying $\mathcal{R}$ for t. Temporal properties of the sequence are represented by constraints defined on these set-variables. We present two models implementing Allen: the first model is based on a classical scheduling approach and the second model uses Multi-valued Decision Diagrams (MDDs). We show that the MDD-based approach, which performs tighter pruning, is much more efficient on the multitrack generation problem. The MDD representation and the associated filtering procedures are complex, which is why we present the first, simpler model. The two approaches are also used to run experimental comparisons showing that the MDD approach is necessary to solve actual synchronization problems.

A constraint based on Allen's algebra [1] takes a set of tasks, a set of Allen relations, a set of intervals, and checks that every task satisfies at least one relation for one interval. They apply this work to the generation of video summaries. In their approach, the checks for every task are independent from one another. On the contrary, in our approach, we use Allen to enforce *explicit* structural temporal properties, defined by Allen relations. Moreover, we take all specified properties into account in a single, global constraint.

The Meter constraint [10] provides control on the duration and on various temporal properties of sequences, but is limited to the definition of *unary* constraints on events defined by their temporal position. It does not provide actual variables representing these events, and cannot be used, for instance, to state equality or difference constraints between events. Allen generalizes Meter by defining two additional set-variables that can be used to state arbitrary, *e.g., binary*, constraints enforcing temporal structures on sequences.

Allen is designed to model and solve problems of temporal sequence synchronization and structure. Previous work, including our own on using Regular [11] for music generation, never addressed this aspect.

2 Constraining Contiguous Temporal Sequences (CTS)

A *temporal event* e is a symbol with a duration $d(e)$. A *contiguous temporal sequence*, *CTS* for short, is a finite sequence of temporal events $(e_1, \ldots, e_n)$. A CTS is basically a concatenation of events: two consecutive events in a CTS are considered contiguous. Therefore, for a CTS $S = (e_1, \ldots, e_n)$, the *duration* $d(S)$ of S is the sum of the duration of the events contained in S, defined by $d(S) = \sum_{i=1}^{n} d(e_i)$. The *absolute temporal position*, or *starting time* of an event e_p in S is defined by $s(e_p) = \sum_{i=1}^{p-1} d(e_i)$. Note that the absolute temporal position is not an intrinsic property of a temporal event, it is a property of a temporal event with respect to a CTS. A same temporal event may appear several times in a same CTS at different starting times.

In this article, we consider only temporal events with integer duration and, therefore, we address CTS in which all events have integer temporal positions.

Given a set E of temporal events, a model for the generation of CTS is to represent a CTS containing n temporal events of E as a sequence of n constrained variables $(X_1, \ldots, X_n)$, each with domain $dom(X_i) = E$. With this model, it is easy to state constraints relating events based on their index in the sequence, such as $X_1 = X_n$, or $X_i \neq X_{i+1}$. However, the absolute temporal position of an event in a CTS is not directly related to its index as it depends on the duration of all preceding events. There is therefore no straightforward way of constraining the elements of the sequence based on their absolute temporal position.

3 The Global Allen Constraint

The idea behind the Allen constraint is to use Allen relations between temporal intervals to specify some temporal element(s) of a CTS (the 13 atomic relations of Allen are given on Table 1). Let S be a CTS $(e_1, \ldots e_n)$. An Allen relation $\mathcal{R}$ and a temporal interval t specify a subsequence of S. For instance, if $\mathcal{R}$ is **d**, *i.e.,* the relation "during", and $t = [a, b]$, then $\mathcal{R}$ and t specify the subsequence of S containing the events which start after a and end before b.

Let E be a set of temporal events and let $X_1, \ldots, X_n$ be n constrained variables, each with domain E. The X_is are the *sequence* variables. Let t be a temporal interval and let $\mathcal{R}$ be a relation of Allen between temporal intervals (see Table 1). Let $\mathcal{I}$ be a set-variable, with domain $\{1, \ldots, n\}$ and $\mathcal{E}$ be a set-variable with domain E. The Allen constraint

$$\textsc{Allen}_{\mathcal{R},t}(X_1, \ldots, X_n, \mathcal{I}, \mathcal{E}) \tag{1}$$

ensures that $\mathcal{I}$ contains the *indexes* of all sequence variables X_i belonging to the subsequence of $(X_1, \ldots, X_n)$ specified by Allen relation $\mathcal{R}$ and temporal interval t. Similarly, the constraint (1) ensures that $\mathcal{E}$ contains the *values* of all sequence variables X_i belonging to the subsequence of $(X_1, \ldots, X_n)$ specified by $\mathcal{R}$ and t.

The Allen constraint defined above is satisfied if and only if

$$\mathcal{I} = \{i \in \{1, \ldots, n\} \mid [s(X_i), s(X_{i+1})] \, \mathcal{R} \, t\} \text{ and } \mathcal{E} = \{X_i \mid i \in \mathcal{I}\}$$

For example, the melody on Figure 1 contains 36 notes (including rests). We represent each note as a temporal event and the melody as a CTS $M = (n_1, \ldots, n_{36})$, where n_i is the i-th note of the melody. Writing events as (note, duration) ordered pairs, with duration 1 for eighth-notes, we have:

$n_1 = (rest, 1); n_2 = (B4, 1); n_3 = (D5, 1); n_4 = (B4, 1); n_5 = (E5, 1);$
$n_6 = (B4, 1); n_7 = (D5, 1); n_8 = (E5, 1); n_9 = (B4, 8); \ldots$

where $D4$ denotes pitch D and octave 4. Note that $n_2 = n_4 = n_6$, $n_3 = n_7$, and $n_5 = n_8$.

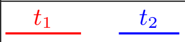
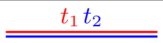
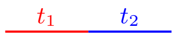
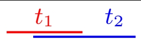
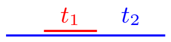
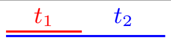
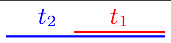
Relation	Symbol	Example	Semantics	Inverse
t_1 before t_2	$<$		$t_1+ < t_2-$	$>$
t_1 equal t_2	**eq**		$t_1- = t_2-$ and $t_1+ = t_2+$	**eq**
t_1 meets t_2	**m**		$t_1+ = t_2-$	**mi**
t_1 overlaps t_2	**o**		$t_1- < t_2-$ and $t_2- < t_1+ < t_2+$	**oi**
t_1 during t_2	**d**		$t_1- > t_2-$ and $t_1+ < t_2+$	**di**
t_1 starts t_2	**s**		$t_1- = t_2-$ and $t_1+ < t_2+$	**si**
t_1 finishes t_2	**f**		$t_1- > t_2-$ and $t_1+ = t_2+$	**fi**

Table 1. The 13 atomic relations of Allen. The lower bound of a time interval t_i is denoted by t_i- and the upper bound by t_i+.

Consider the relation of Allen "during" and the time interval defined by the first bar, *i.e.,* the interval starting at temporal position 1 and ending at temporal position 8 (as we count 1 for an eighth-note). The Allen constraint

$$\textsc{Allen}_{\mathbf{d}[1,8]}(n_1, \ldots, n_{36}, \mathcal{I}, \mathcal{E})$$

is satisfied if, and only if $\mathcal{I} = \{1, 2, \ldots, 8\}$ and $\mathcal{E} = \{(rest, 1), (B4, 1), (D5, 1), (E5, 1)\}$. Note that although the note $(B4, 1)$ appears three times in the first bar, it appears only once in $\mathcal{E}$, as temporal events do not have a starting time.

4 Implementing the Allen Constraint

We now describe two implementations of the Allen constraint. The first one is a simple model, based on scheduling, and performing only local propagations. We show in Section 5 that this model performs poorly. This justifies the need for a more elaborated model, using an MDD to represent the sequences explicitly, which makes it possible to prune more values during the search. This second model is presented in Section 4.2.

In both models, the sequence variables $X_1, \ldots, X_n$ take temporal event values.

4.1 A First Model

The Allen constraint can be seen as a non-preemptive scheduling problem with unary resources where variables correspond to activities having a variable duration. In this model, each variable X_i is associated with two variables S_i and D_i. Variable S_i represents the absolute temporal position of X_i in the CTS, and D_i represents the duration of X_i. The start and duration variables are related via a set of constraints

$$S_{i+1} = S_i + D_i, \forall i = 1, \ldots, n-1 \qquad (2)$$

with $S_1 = 0$.

In order to define the propagation rules, we will use the following five predicates:

- $\textsc{Holds}_{\mathcal{R},t}(s,d) \overset{def}{\iff} [s, s+d]\,\mathcal{R}\,t$, where s is a start time (i.e., absolute temporal position) and d a duration
- $\textsc{Possible}_{\mathcal{R},t}(i,e) \overset{def}{\iff} e \in dom(X_i)$ and $\exists s \in dom(S_i)$, $\textsc{Holds}_{\mathcal{R},t}(s, d(e))$
- $\textsc{Possible}_{\mathcal{R},t}(i) \overset{def}{\iff} \exists e \in dom(X_i)$ such that $\textsc{Possible}_{\mathcal{R},t}(i,e)$
- $\textsc{Required}_{\mathcal{R},t}(i,e) \overset{def}{\iff} X_i = e$ and $\forall s \in dom(S_i)$, $\textsc{Holds}_{\mathcal{R},t}(s, d(e))$
- $\textsc{Required}_{\mathcal{R},t}(i) \overset{def}{\iff} \forall e \in dom(X_i), \forall s \in dom(S_i)$, $\textsc{Hold}_{\mathcal{R},t}(s, d(e))$

Variables $\mathcal{I}$ and $\mathcal{E}$ are set-variables [12]. We will use the notation $lb(.)$ for the lower-bound of a set-variable domain and $ub(.)$ for its upper-bound. Intuitively, during the filtering procedure, the lower-bound $lb(\mathcal{I})$ (resp., $lb(\mathcal{E})$) is the set of required values for $\mathcal{I}$ (resp., $\mathcal{E}$). Similarly, the upper-bound $ub(\mathcal{I})$ (resp., $ub(\mathcal{E})$) is the set of possible values for $\mathcal{I}$ (resp., $\mathcal{E}$). The filtering rules presented below rely on the equivalences:

$$\begin{aligned} i \in ub(\mathcal{I}) &\iff \textsc{Possible}_{\mathcal{R},t}(i) && (3)\\ i \in lb(\mathcal{I}) &\iff \textsc{Required}_{\mathcal{R},t}(i) && (4)\\ e \in ub(\mathcal{E}) &\iff \exists i, \textsc{Possible}_{\mathcal{R},t}(i,e) && (5)\end{aligned}$$

Note that $e \in lb(\mathcal{E})$ is more difficult to express in terms of the predicates, which is why Rule (15) is more complex. In fact, reasoning on $lb(\mathcal{E})$ is the most complex operation for maintaining the consistency between the sequence variables and the set-variables. In the next section, we use an MDD model, which is sufficiently rich to infer the exact lower-bound $lb(\mathcal{E})$.

The consistency between the event, start, and duration variables, and the lower and upper bounds of the ALLEN set-variables, is maintained with a set of filtering rules.

When S_i is modified, *i.e.,* a value was removed from its domain, the following rules may apply:

$$\begin{aligned} i \in ub(\mathcal{I}) :\ & \neg\textsc{Possible}_{\mathcal{R},t}(i) \Rightarrow i \notin ub(\mathcal{I}) \\ & \textsc{Required}_{\mathcal{R},t}(i) \Rightarrow i \in lb(\mathcal{I}) && (6)\\ i \in lb(\mathcal{I}) :\ & e \in dom(X_i) \wedge (\forall s \in dom(S_i), \neg\textsc{Holds}_{\mathcal{R},t}(s, d(e))) \\ & \Rightarrow e \notin dom(X_i) && (7)\\ i \notin ub(\mathcal{I}) :\ & e \in dom(X_i) \wedge (\forall s \in dom(S_i), \textsc{Holds}_{\mathcal{R},t}(s, d(e))) \\ & \Rightarrow e \notin dom(X_i) && (8)\\ e \in ub(\mathcal{E}) :\ & \nexists j, \textsc{Possible}_{\mathcal{R},t}(j,e) \Rightarrow e \notin ub(\mathcal{E}) && (9)\\ e \notin ub(\mathcal{E}) :\ & (\forall s \in dom(S_i), \textsc{Holds}_{\mathcal{R},t}(s, d(e))) \\ & \Rightarrow e \notin dom(X_i) && (10)\end{aligned}$$

Let us explain the first rule, Rule (6), in detail. Rule (6) is applied when a value is removed from the domain of S_i and if $i \in ub(\mathcal{I})$. The predicate $\textsc{Possible}_{\mathcal{R},t}(i)$ is evaluated, and if it does not hold true, index i is removed from $ub(\mathcal{I})$. The predicate $\textsc{Required}_{\mathcal{R},t}(i)$ is also evaluated, and if it holds true, index i is added to $lb(\mathcal{I})$. The variable S_i represents the starting times of the i-th event in the CTS. The property $i \in ub(\mathcal{I})$ means exactly that predicate $\textsc{Possible}_{\mathcal{R},t}(i)$ holds true (by Equivalence (3)). A possible consequence of removing a value from the domain of S_i is that there may be no more starting time s in S_i such that $[s, s+d(e)]\,\mathcal{R}\,t$. Therefore, we reevaluate $\textsc{Possible}_{\mathcal{R},t}(i)$, and if it does not hold true anymore, we remove i from $ub(\mathcal{I})$. Another possible consequence of removing a value from S_i is that all remaining values $s \in dom(S_i)$ are such that $[s, s+d(e)]\,\mathcal{R}\,t$ for any event $e \in dom(X_i)$, which means that $\textsc{Required}_{\mathcal{R},t}(i)$ holds true. Equivalence (4), we add index i to $lb(\mathcal{I})$.

When X_i is modified, we apply the following rules:

$$
\begin{aligned}
i \in ub(\mathcal{I}) :\ & \neg\textsc{Possible}_{\mathcal{R},t}(i) \Rightarrow i \notin ub(\mathcal{I}) \\
& \textsc{Required}_{\mathcal{R},t}(i) \Rightarrow i \in lb(\mathcal{I}) && (11)\\
i \in lb(\mathcal{I}) :\ & dom(X_i) = \{e\} \Rightarrow e \in lb(\mathcal{E}) \\
& s \in dom(S_i) \wedge (\forall e \in dom(X_i), \neg\textsc{Holds}_{\mathcal{R},t}(s, d(e))) \\
& \ \Rightarrow s \notin dom(S_i) && (12)\\
i \notin ub(\mathcal{I}) :\ & s \in dom(X_i) \wedge (\forall e \in dom(X_i), \textsc{Holds}_{\mathcal{R},t}(s, d(e))) \\
& \ \Rightarrow s \notin dom(S_i) && (13)\\
e \in ub(\mathcal{E}) :\ & \nexists j, \textsc{Possible}_{\mathcal{R},t}(j, e) \Rightarrow e \notin ub(\mathcal{E}) && (14)\\
e \in lb(\mathcal{E}) :\ & \exists i \in ub(\mathcal{I}) \text{ s.t.} \\
& \qquad \textsc{Possible}_{\mathcal{R},t}(i, e) \wedge \\
& \qquad \forall j \in ub(\mathcal{I}) \text{ s.t. } j \neq i, (e \notin dom(X_j) \vee \neg\textsc{Possible}_{\mathcal{R},t}(j, e)) \\
& \ \Rightarrow dom(X_i) = \{e\} \wedge i \in lb(\mathcal{I}) && (15)
\end{aligned}
$$

When $\mathcal{I}$ is modified: if $i \in lb(\mathcal{I})$, apply Rule (7) and Rule (12); if $i \notin ub(\mathcal{I})$ apply Rule (8) and Rule (13). When $\mathcal{E}$ is modified: if $e \in lb(\mathcal{E})$ apply Rule (15); if $e \notin ub(\mathcal{E})$, $\forall i \in ub(\mathcal{I})$, apply Rule (10).

Most of those rules are straightforward implications of the predicate definitions, except rule (15). The first line of Rule (15) says that it is possible to have value e in the sequence. The following lines express the fact that if a only one variable X_i may take value e, we perform the assignment $X_i \leftarrow e$. We can easily verify that no rule removes any consistent value, *i.e.,* the rules are sound. However, this model does not remove all inconsistent values, *i.e.,* it does not achieve arc-consistency for Allen.

4.2 MDD-Based Model

This model uses an MDD constraint to represent the extension of the Allen constraint. By using propagators for MDDs, we can therefore achieve arc-consistency of the whole Allen constraint. In fact, we show that we can even combine several Allen constraints into a single MDD and thus achieve arc-consistency for a set of Allen constraints. An MDD, in constraint programming, is a directed acyclic graph with one layer of nodes per constrained variable plus a final layer containing the single *true* node [13]. Defining the Allen constraint with MDDs can be decomposed into two steps:

– We first represent temporal constraint by a transition function computing the set of all temporal positions reachable from a given temporal position. The MDD is defined, starting from a root node associated to position 0, by successive applications of the transition function to determine all reachable temporal positions. An arc is associated to a duration, *i.e.,* time difference between the temporal position of its ending node and its origin node, *i.e.,* for and arc $a = (i, j)$, we have $t(j) = t(i) + d(a)$, where $t(.)$ denotes the temporal position of a node. The MDD constructed this way simply represents a sum function. We use the MDD Pattern construction defined in [14], which allow us to build an MDD based on a function of the node. Here the function node is the transition function between durations. Events are introduced in the MDD as follows: for each arc associated to a duration, we create as many arcs as there are events with this duration. Each arc in the resulting MDD is therefore labelled with a couple (event, duration).
– Then, for a given Allen relation, we identify all the arcs in the MDD that satisfy this relation. We can do this by noting that an arc $a = (i, j)$ occupies the temporal interval $[t(i), t(j)]$. These are the red arcs in Figure 2.

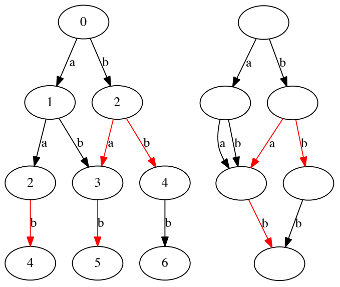


Fig. 2. The graph (left) and MDD (right) representations of the constraint $\text{ALLEN}_{\mathbf{d}\vee\mathbf{s}\vee\mathbf{fi}\vee\mathbf{eq}[2,5]}$ (see Table 2). Red labels correspond to values satisfying the constraint. Numbers in the graph on the left represent the temporal position.

To illustrate this with an example, consider two events a and b with $d(a) = 1$ and $d(b) = 2$ and a sequence of three variables X_1, X_2, X_3 with domains $dom(X_1) = dom(X_2) = \{a, b\}$ and $dom(X_3) = \{b\}$. Let R denote the relation $\mathbf{d} \vee \mathbf{s} \vee \mathbf{f} \vee \mathbf{eq}$, which is similar to $\mathbf{d}$ except it is not strict. The extension of $\text{ALLEN}_{R[2,5]}([X_1, X_2, X_3], \mathcal{I}, \mathcal{E})$ is shown in Table 2, where events that occur, not strictly, during [2,5] are in red.

X_1	X_2	X_3	$\mathcal{I}$	$\mathcal{E}$
a	a	b	$\{3\}$	$\{b\}$
a	b	b	$\{3\}$	$\{b\}$
b	a	b	$\{2,3\}$	$\{a,b\}$
b	b	b	$\{2\}$	$\{b\}$

Table 2. The extension of $\text{ALLEN}_{R[2,5]}$ for the example. Events that are, not strictly, during $[2,5]$ are in red.

The list of valid sequences of the constraint may be represented by the graph in Figure 2 (left). Each layer represents one sequence variable (X_1 is the top layer, X_2 is the middle layer, and X_3 the bottom layer). Node labels represent start times and edge labels are events. Edges corresponding to events satisfying $\text{ALLEN}_{R[2,5]}$ are in red.

Note that the Allen relation does not change during search. As a consequence, one can ignore the temporal information in the nodes and apply the MDD reduction operation to the graph. This yields the reduced MDD in Figure 2 (right). Note that the reduction distinguishes between black and red labels.

The algorithm presented in [15] is used to filter the domains of the sequence variables in the constraint represented by the MDD. The set-variables $\mathcal{I}$ and $\mathcal{E}$ must satisfy the following properties:

1. if $\forall a \in A_i$, a is red, then $i \in lb(\mathcal{I})$;
2. if $\exists a \in A_i$ such that a is red, then $i \in ub(\mathcal{I})$ and $label(a) \in ub(\mathcal{E})$.

where A_i denotes the i-th layer of the MDD, and a denotes an arc. By using the arcs deleted from the MDD, we maintain these properties incrementally.

When an event e is added to $lb(\mathcal{E})$, it means that all complete path in the MDD contains at least one red arc labelled with e. We modify the current MDD, notated MDDc, to integrate this new information. This is done by generating a new MDD, MDD(-e) containing all paths of MDDc that do not go through a red arc labelled with e. Then, we substract MDD(-e) from MDDc. This is efficient as MDD(-e) is a subgraph of MDDc. To generate MDD(-e), we duplicate MDDc, suppress all red arcs labelled with e, and propagate the suppression until every node belongs to a complete path (from the root node to a terminal node). This procedure is described in an article by Perez and Régin [16]. The complexity of these filtering operations is bounded by the size of the MDD.

In practice, the MDD representation is efficient because the bottom layers are highly compressed. This approach solves problems with up to 150 variables, which is enough for the targeted applications (see Section 5.3).

An important aspect of this approach it that we represent *several* ALLEN constraints stated on the same sequence in a *single* MDD. Then, we implement channeling relations between the MDD and the set-variables for each relation. This allows us to achieve arc-consistent of the conjunction of all the ALLEN constraints. Note that integrating the set-variables in the MDD would require the definition of one MDD per Allen relation, and would sacrifice compression without improving filtering.

5 Evaluation

We evaluated the two models on an instance of the audio multitrack synchronization problem described in the introduction. The evaluation was implemented using the OR Tools [1] solver. Benchmark data will be made available online.

5.1 Description of the Benchmark

We use a three-track recording (guitar, bass, and drum) of the first 32 bars of song *Prayer in C* (Lilly Wood & The Prick). Each track is segmented using standard onset detection and quantized to 1/24th of a beat (see Figure 3).

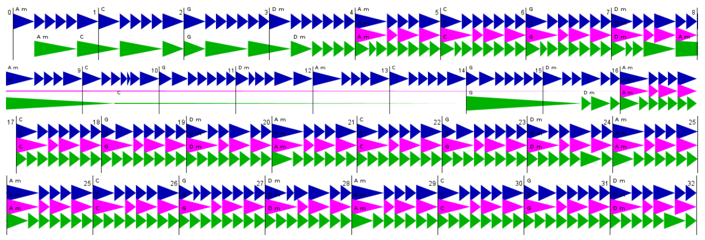


Fig. 3. A graphical representation of the guitar (top), bass (center), and drum (bottom) tracks of *Prayer in C*. Each track contains 32 bars and each triangle represents a chunk. Vertical lines indicate bar separations.

Chunks are categorized into *clusters* according to harmonic similarity (for pitched instruments) and timbre similarity for drums. The timbre is represented by Mel Frequency Cepstral Coefficients (MFCC) with 13 coefficients; the harmonic similarity is computed using the Harmonic Pitch-Class Profile (HPCP) with 36 divisions of the octave [17].

The guitar track contains 128 chunks with duration ranging from an eighth-note (half a beat) to a dotted quarter-note (1.5 beats), categorized into 13 harmonic clusters. The bass track contains 81 chunks (duration from half a beat to 1.5 beats) categorized into 9 bass clusters (harmonic similarity). The drum track contains 94 chunks with duration from half a beat to 20/3 beats, that is a full bar plus two thirds of a bar. There are 40 timbre-based drum clusters.

We state the problem of creating new multitracks as the generation of three sequences of chunks, each with an imposed total duration of n bars. Each bar has four beats, and the duration of the shortest chunks is $^1/_8$ of a bar, therefore each sequence contains at most $p = 32n$ chunks.

[1] OR Tools is open source and available at https://github.com/google/or-tools

We define a sequence of p chunk variables: $G_1, \ldots, G_p$ (guitar), $B_1, \ldots, B_p$ (bass), and $D_1, \ldots, D_p$ (drums). The domain of each variable is the set of chunks in the corresponding recorded track, plus a dummy chunk with duration 0, called the *padding element*, which we explain below.

For each track, all chunk transitions, *e.g.,* $G_i \to G_{i+1}$, are such that the associated cluster transition exists in the original track. Additionally, we synchronize the tracks together at the beginning of every bar. More precisely, let G_i, B_j, D_k be the three chunks playing at the beginning of bar b, and $C(G_I), C(B_j), C(D_k)$ be the corresponding clusters. We enforce that the same cluster "signature" exists somewhere in the original multitrack, not necessarily at the beginning of a bar. The underlying idea is that the cluster signatures of the original track are musically acceptable. Intuitively, this constraint imposes that the generated multitrack uses acceptable chunk signatures at the beginning of every bar but can "invent" new cluster signatures (new sounds) between bar lines.

It is easy to impose the total duration of $4n$ to each track by simply removing all nodes of the graph (see Figure 2, left) whose label is greater than $4n$ and every node of the final layer whose label is different from $4n$. Every node with label $4n$ that is not in the final layer receives a new arc, labeled with the padding element, going to the next layer to a new node with the same label $4n$, since the padding element has a 0 duration. We repeat this process to the final layer. This allows us to generate sequences with fewer than p "actual" variables, the padding value being assigned to the "extra" variables.

The variables are subject to the binary constraints on chunk cluster transitions. These constraints are expressed as simple table constraints between consecutive chunk variables in each track. For example, $(C(G_i) \to C(G_{i+1})) \in \mathcal{C}_g$, where $\mathcal{C}_g$ is the set of all cluster transitions in the original guitar track. The same applies to the two other instruments.

The vertical synchronization constraints are represented by an ALLEN constraint for each track and for each bar. To specify the events that are playing at the beginning of bar i, we use the Allen relation $\mathbf{o} \vee \mathbf{s}$ applied to the time interval $[4(i-1), +\infty)$. In our context, this relation, one of the 2^{13} combinations of Allen relations, specifies exactly the intervals which "contain" the temporal point $4(i-1)$, the onset of bar i.

The ALLEN constraints for the guitar track are

$$\textsc{Allen}_{\mathbf{o}\vee\mathbf{s}\,[4(i-1),\infty)}([C^g_1, \ldots, C^g_p], \mathcal{I}^g_i, \mathcal{E}^g_i)$$

where C^g_i is the variable cluster(G_i). The synchronization itself is enforced by an *ad hoc* table constraint between $\mathcal{E}^g_i$, $\mathcal{E}^b_i$, and $\mathcal{E}^d_i$, where accepted triplets are cluster signatures of the original multitrack.

This approach applies to the generation of automatic accompaniment of an imposed melody in a given style. A demo video[2] is available online. Note that in this case, we enforce additional harmonic constraints to match a target chord sequence, in the case of the video above, the *Ode to joy.*

[2] Video available online, https://www.youtube.com/watch?v=buXqNqBFd6E, examples at at seconds 140, 176, and 216.

5.2 Evaluation of the First Model

We evaluate two implementations of the scheduling model, depending on how we implement constraint (2), which links start times and durations (defined in Section 4.1). First, we enforce arc-consistency on this ternary sum constraint. The model solves the problem for two bars in 8.4 seconds. It does not solve the problem for more than two bars in less that 30 minutes, which we consider a timeout. It is interesting to observe that the set of ternary sum constraints models a Regular constraint enforcing the graph on Figure 2 (left). Furthermore, by enforcing AC for the ternary sum constraints, we also achieve AC for this Regular, since we obtain a model equivalent to the Berge acyclic decomposition of Regular [18]. The more complex MDD-based model differs from this approach by the compression applied to this graph, and its exploitation to obtain a tighter filtering on the set-variables.

We also implemented a lighter version where the ternary sums constraints only perform bound-consistency, based on the intuition that propagating information about the bounds of event duration offers a good trade off between simplicity and pruning. This model solves the problem for two bars in 5.4 seconds, but does not scale either to larger instances.

5.3 Evaluation of the MDD-Based Model

We use MDD4R [15] to perform the operations on the MDD constraint representing each Allen constraint. The code is implemented using the OR Tools solver. Note that, as said in Section 4.2, all the Allen constraints for a same track are represented by a single MDD.

n	MDD size (#Vertices, #Edges)						Time
	Guitar		Bass		Drum		(ms)
6	2382	41k	848	13667	1864	73k	2301
8	4199	74k	1493	24k	3817	156k	7219
10	6530	117k	2388	39k	6513	275k	23k
12	9374	169k	3623	61k	9957	429k	57k
14	12k	231k	5085	87k	14k	617k	112k

Table 3. The size of the MDDs and the execution time to find 5 solutions for various multitrack lengths

.

The comparison with the performance of the simple model for Allen is clearly in favor of the MDD approach (see Table 3). The simple model does not solve problems longer than two bars in less than 30 minutes. In contrast, the MDD-based model solves the 14-bar problem in less than 2 minutes. The extra cost of performing the MDD construction and operations is more than compensated for by the higher pruning offered by this model, especially regarding the treatment of the set-variable $\mathcal{E}$.

6 Generation of Lead Sheets

To complete our presentation, we now address the problem of generating a melody, given a chord sequence, using Allen. This task, involving a single sequence, is a particular application of a general framework for lead sheet composition that we have developed. In this section, we sketch out the problem and how to state it using Allen. We do not report computation times as they vary considerably depending on the corpus and on the imposed chord sequence.

The melody is defined as a sequence of note variables, where a note is an event with a pitch and a duration, subject to constraints enforcing: (1) the duration of the melody matches that of the chord sequence, (2) the note transitions form acceptable musical intervals, (3) the notes match the current harmony (the chord), and (4) the melody satisfies an imposed pattern structure. We explain these constraints in detail on the example on Figure 4.

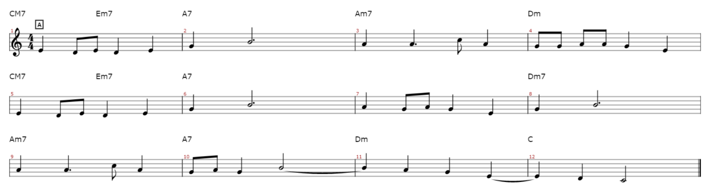


Fig. 4. A 12-bar lead sheet generated using an Allen constraint.

For a 12-bar melody with a $^4/_4$ time signature, we define a sequence of note variables $[N] = [N_1, \ldots, N_p]$, with $p = 96$. The total duration is $48 = 12 \times 4$ beats. The melodic interval constraints are stated as binary table constraints between the pitch of any two consecutive notes. The allowed pitch intervals are collected from a corpus of 12000 lead sheets of popular music.

To enforce the harmonic constraints, we state an Allen constraint for each chord. For instance, on Figure 4, the first chord, CM7, occupies the time interval [1,3] (half of the first bar). We define $\textsc{Allen}_{\mathbf{s}\vee\mathbf{d}\vee\mathbf{eq}\vee\mathbf{f}[1,3]}(\mathcal{E}_{\text{CM7}}, \mathcal{I}_{\text{CM7}})$, and use a unary constraint on $\mathcal{E}_{\text{CM7}}$, restricting its set-domain to the notes whose pitch is harmonically compatible with CM7. The allowed pitch for a given chord are extracted from our corpus.

Figure 4 shows a 12-bar lead sheet generated using an Allen constraint enforcing the equality between the patterns of bars 1-2 and bars 5-6, and between bars 3 and 9. Let P_1 denote the pattern of bars 1-2; P_2 that of bar 3; P_3 that of bars 5-6; and P_4 for bar 9. We state one Allen constraint for each pattern

- P_1 corresponds to $\textsc{Allen}_{\mathbf{s}\vee\mathbf{eq}\vee\mathbf{d}\vee\mathbf{f}[1,9]}([N], \mathcal{E}_1, \mathcal{I}_1)$,
- P_3 corresponds to $\textsc{Allen}_{\mathbf{s}\vee\mathbf{eq}\vee\mathbf{d}\vee\mathbf{f}[9,13]}([N], \mathcal{E}_2, \mathcal{I}_2)$,
- P_3 corresponds to $\textsc{Allen}_{\mathbf{s}\vee\mathbf{eq}\vee\mathbf{d}\vee\mathbf{f}[17,25]}([N], \mathcal{E}_3, \mathcal{I}_3)$,
- P_4 corresponds to $\textsc{Allen}_{\mathbf{s}\vee\mathbf{eq}\vee\mathbf{d}\vee\mathbf{f}[33,37]}([N], \mathcal{E}_4, \mathcal{I}_4)$.

However, the relations between patterns are relations between sub-sequences. They cannot be stated straightforwardly in terms of $\mathcal{E}_1$ and $\mathcal{E}_3$, as these do not maintain the order of the values.

For two index set-variables $\mathcal{I} = \{i_1, \ldots, i_k\}$ and $\mathcal{J} = \{j_1, \ldots, j_k\}$ with the indexes sorted by increasing order. We define the sub-sequence equality constraint

$$\textsc{SeqEq}([X_1, \ldots, X_n], \mathcal{I}, \mathcal{J}) \iff X_{i_1} = X_{j_1} \wedge \cdots \wedge X_{i_k} = X_{j_k}$$

This constraint is expensive to filter as the domains of $\mathcal{I}$ and $\mathcal{J}$ are not known in advance. We add a redundant constraint to speed up the propagation:

$$\textsc{Equal}(\mathcal{E}, \mathcal{F})$$

where $\mathcal{E}$ and $\mathcal{F}$ are the even set-variables corresponding to $\mathcal{I}$ and $\mathcal{J}$ respectively. The filtering procedure for the equality constraint between set-variables (Equal) is quite standard; the filtering procedure for SeqEq is not presented here. Basically, the filtering operation starts only once the first index in $\mathcal{I}$ and in $\mathcal{J}$ are known or, similarly, once the last index in $\mathcal{I}$ and in $\mathcal{J}$ are known.

7 Conclusion

We have presented the Allen global constraint. Allen maintains set-variables representing events in a temporal sequence in two ways: one variable is the set of events occurring at a given position, defined by an Allen relation with a reference time interval; the other variable is the set of indexes of these events. In practice, Allen offers the possibility to control the generation of temporal sequences by constraining events defined by their index *and* temporal position.

We proposed two models for Allen: a simple model using local propagation and a model based on MDDs and shown that the MDD representation, which achieves the global AC of the constraint, performs much better than the simple model on a temporal sequence synchronization problem.

Allen makes it possible to model and solve new types of problems involving structural constraints on patterns, represented by sub-sequences. We illustrated Allen on the task of synchronizing several audio tracks. Another application is the generation of structured lead sheets with pattern repetition. Such tasks could hardly be addressed using standard global constraints. More generally we believe that Allen addresses an increasing need for enforcing complex structural constraints in content generation for the entertainment domain.

Acknowledgment

This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 291156.

References

1. Derrien, A., Fages, J.G., Petit, T., Prud'homme, C.: A global constraint for a tractable class of temporal optimization problems. In: Principles and Practice of Constraint Programming – CP 2015, Springer (2015) 105–120
2. Galvane, Q., Christie, M., Lino, C., Ronfard, R.: Camera-on-rails: Automated Computation of Constrained Camera Paths. In: ACM SIGGRAPH Conference on Motion in Games, Paris, France (November 2015)
3. Galvane, Q., Ronfard, R., Lino, C., Christie, M.: Continuity Editing for 3D Animation. In: AAAI Conference on Artificial Intelligence. AAAI Press, Austin, Texas, United States (January 2015)
4. Berrani, S.A., Boukadida, M.H., Gros, P.: Constraint satisfaction programming for video summarization. In: IEEE International Symposium on Multimedia, Anaheim, California, United States, IEEE (December 2013)
5. Dixon, S.: Onset detection revisited. In: Proceedings of the 9th International Conference on Digital Audio Effects. Volume 120., Citeseer (2006) 133–137
6. Maestre, E., Ramírez, R., Kersten, S., Serra, X.: Expressive concatenative synthesis by reusing samples from real performance recordings. Comput. Music J. **33**(4) (December 2009) 23–42
7. Nair, M.: On chebyshev-type inequalities for primes. AMM **89** (1982) 126–129
8. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Commun. ACM **26**(11) (1983) 832–843
9. Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Networks. Artif. Intell. **49**(1-3) (1991) 61–95
10. Roy, P., Pachet, F.: Enforcing Meter in Finite-Length Markov Sequences. In desJardins, M., Littman, M.L., eds.: AAAI, AAAI Press (2013)
11. Papadopoulos, A., Pachet, F., Roy, P., Sakellariou, J.: Exact Sampling for Regular and Markov Constraints with Belief Propagation. In: CP. Volume 9255 of Lecture Notes in Computer Science., Springer (2015) 341–350
12. Puget, J.F.: PECOS: a High Level Constraint Programming Language. In: Proceedings of Singapore International Conference on Intelligent Systems. SPICIS'92 (1992) 137–142
13. Hoda, S., Van Hoeve, W.J., Hooker, J.N.: A systematic approach to mdd-based constraint programming. In: Principles and Practice of Constraint Programming–CP 2010, Springer (2010) 266–280
14. Perez, G., Régin, J.c., Antipolis, U.N.s., Umr, I.S.: Efficient Operations on MDDs for Building Constraint Programming Models. In: IJCAI International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina (2015) 374–380
15. Perez, G., Régin, J.C.: Improving GAC-4 for Table and MDD Constraints. In Publishing, S.I., ed.: Principles and Practice of Constraint Programming, Lyon, France (2014) 606–621
16. Perez, G., Régin, J.C.: Relations between MDDs and Tuples and Dynamic Modifications of MDDs based constraints. arXiv preprint arXiv:1505.02552 (2015)
17. Gómez, E.: Tonal Description of Music Audio Signals. PhD thesis, Universitat Pompeu Fabra (2006)
18. Quimper, C., Walsh, T.: Global grammar constraints. In Benhamou, F., ed.: Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings. Volume 4204 of Lecture Notes in Computer Science., Springer (2006) 751–755