Visualizing Timed, Hierarchical Code Structures in AscoGraph

Grigore Burloiu, Arshia Cont

Visualizing Timed, Hierarchical Code Structures in AscoGraph

Grigore Burloiu
Faculty of Electronics, Telecommunications
and Information Technology
University Politehnica of Bucharest
gburloiu@gmail.com

Arshia Cont
MuTant Team-Project
IRCAM STMS UMR, CNRS, INRIA, UPMC
cont@ircam.fr

***Abstract*—*Antescofo* is a state-of-the-art software package for mixed music authorship and performance. In this paper, we apply an information visualisation perspective to a set of revisions in the timeline-based representation of action items in *AscoGraph*, the dedicated user interface to *Antescofo*. Our contribution is two-fold: (a) a design study of the proposed new model, and (b) a technical, algorithmic component. In the former, we show how our model relates to principles of information coherence and clarity, facility of seeking and navigation, hierarchical distinction and explicit linking. In the latter, we frame the problem of arranging action rectangles in a 2D space as a *strip packing* problem, with the additional constraint that the (horizontal) time coordinates of each block are fixed. We introduce three algorithms of increasing complexity for automatic arrangement, estimate their packing performance and analyse their strengths and weaknesses. We evaluate the systemic improvements achieved and their applicability for other time-based datasets, while noting the limitations of the model and resulting directions for future research.**

I. Introduction

The aim of information visualisation is to organise heterogenous data graphically in a way that optimises human cognition, based on an inherent model. The model should address specific visualisation aims and yet be general enough to allow for wide public adoption and adaption to other problems from the domain. An interesting instance is the "standard" model of Western music notation: it developed over centuries to render pitch and rhythm readable, writable and playable by its users, while its semantics have been further extended to wider harmonic, instrumental and stylistic applications over the years, all the time keeping the basic framework. Still, the notation will never capture all the sonic aspects a musician might have in mind when authoring or reading a score. In this sense, any visualisation must strike a bargain between what is shown the user, and what remains hidden or implied. The major advantage of interactive models is the enabling of user-led exploration and focus, so that the visible domain becomes fluid, adapting to one's own needs and preferences.

This paper addresses the problem of ergonomically representing a set of timed hierarchical structures. Our use case is a reactive system for mixed music *Antescofo* [1] and its dedicated visual interface, *AscoGraph* [2]. *Antescofo* is an award-winning software[1] for authorship (programming) and execution (performance) of mixed music, defined as the live association of human musicians and reactive/interactive computer software. The system has been rapidly adopted by artists and is widely used in public concerts around the world.

Contributions of this work are however of interest to the wider domain of time-based information visualisation. Throughout this paper, we highlight aspects particular to *AscoGraph* and show how they relate to more general visualisation principles and techniques. The design study thus constructed represents the primary focus of this paper.

As a secondary contribution, we introduce three new algorithms developed for spatial arrangement of timed modules in *AscoGraph*. They are however not specific to our system, and might be applied in other situations involving timed data-block sets.

Other visual solutions for interactive media authorship exist. We note Iannix [3] and Iscore [4], fully-featured suites employing multiple timelines and drag-and-drop type functionality to manipulate actions and events. What distinguishes *Antescofo-AscoGraph* is the inherent system coupling of a *score follower* [1] with a *reactive action language* [5] which, unlike the alternatives, allows for unassisted live performance of mixed music which is permanently responsive to the musician. *AscoGraph* seamlessly integrates these two core functions into a coherent visual presentation.

Inside the *AscoGraph* environment (Fig. 1), the code for a mixed score is listed in the text editor on the right, whose contents are reflected in the graphical display on the left hand side. The two graphical views, an instrumental *piano roll* and an electronic section, are coupled in musical time along a common horizontal timeline. Each musical event in the top view can have one or several corresponding action blocks in the bottom view.

Our paper starts with a short discussion in section II of the role of time in *Antescofo*, after which in section III we delineate the problems tackled. We then present our revisions to *AscoGraph*'s electronic action view, as it was first described in [2], developed towards a clearer and more time-coherent

This project was partially funded by the French ANR INEDIT Project.

[1] http://repmus.ircam.fr/antescofo/

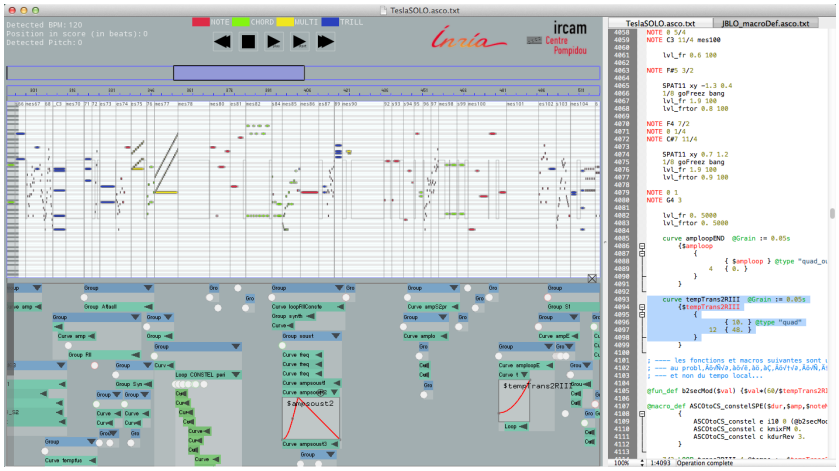


Figure 1: The *AscoGraph* visual editor for *Antescofo* mixed music scores. Right: the textual score describing acoustic events and connected electronic actions. Left, upper: the *piano roll* view of acoustic material. Left, lower: the electronic actions layout.

visualisation of *Antescofo* scores. Our algorithmic solution for efficiently positioning action blocks is explained in section IV. In section V we address the issue of coherence between block width and musical time. Sections VI and VII cover block filtering and colouring — two features meant to facilitate the navigation of the score dataset. We conclude the paper with an overall discussion of the present model and future perspectives.

II. Representing Time in *AscoGraph*

We present a brief overview of temporal concepts in the *Antescofo* action language. It is the final goal of *AscoGraph* to comprehensively and coherently visualise all instances of such concepts.

We distinguish between two types of actions: *atomic* and *compound* [6]. The former correspond to points in time, and the latter to temporal intervals. For instance, an atomic action might be a simple message trigger, or a specific computation. A compound action could be an automation Curve[2] unfolding over time, or a Loop construct that repeats certain atomic actions sequentially. Only compound actions can be said to have a duration (which can be dynamically determined), while atomic actions are by definition logically instantaneous.

Any combination of actions can be grouped together. Nesting compound actions within each other produces a deep hierarchy of *action groups*. Within a group, items can have different start times, so the ”parent” action group start time is by definition the start time of the first item.

The *Antescofo* language supports all three major types of temporal structures: *linear*, *cyclical* and *branching* [7]. However, the current iteration of *AscoGraph* is organised around a purely linear timeline. The reasons for this design choice are the following: (1) the fact that a majority of musical works are composed in linear time, (2) the analogy to classical music notation that most musicians are familiar with, (3) the similarity to commonly used timeline-based music software (e.g. ProTools, Ableton Live), and (4) the analogy to the linear nature of *Antescofo* code. A limitation of the model is that, while the Jump language feature allows for cyclical or branching transitions in time, this does not currently impact *AscoGraph*’s temporal organisation.

There remains a gap between the complexity of the *Antescofo* action language and *AscoGraph*’s representation capabilities. Along the development process, language concepts have been introduced that are sometimes difficult to readily represent visually. The next section details the problems that our current contribution has addressed, as a step towards bridging this gap.

III. Problem Definitions

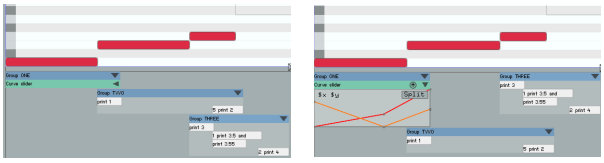
The following is a list of design requirements, to be tackled in subsequent sections. They are ordered by visual scale, from the level of action groups to the score as a whole:

- clearly represent action group content and duration. We frame this issue as a *strip packing* problem in section III-A, and we propose three new algorithms to solve it in section IV;
- ensure time coherence between atomic and compound action items. Section V covers this requirement;
- facilitate the locating of specific information. We achieve this through action filtering, presented in section VI;
- represent the hierarchical structure of the code. Sections V and VII show our advances in this regard;
- synthesise a useful overview of the score, showing the degree of electronic complexity over musical time. All the sections listed above contribute to this aspect, which is further discussed in the concluding section.

A. The Bin Packing Perspective

In the initial iteration of *AscoGraph*, it was common to have temporally overlapping action groups displayed on top of each other along the timeline. The resulted in a loss of coherence

[2]in *Antescofo*, Curves are interpreted as discrete sequences of actions, whose parameters contain variables that change over time.



(a) Curve block is closed (b) Curve block is open

Figure 2: Expanding an action block leaves room for another block to drop to the horizontal base.

and clarity, with the widths of some action blocks no longer corresponding to their programmed durations.

In order to rectify this, we amended the model by stacking action groups in downward non-overlapping order, similarly to how the elements *within* groups are arranged. In the design process, we faced two challenges: (1) managing the 2D space efficiently (due to the new tendency towards vertical growth) and (2) maximising readability and easy navigation. The solutions we propose all strike a certain balance between these parameters.

For arranging rectangular blocks in a two-dimensional space, we translate the issue to a *strip packing* problem. A subset of the *bin packing* approach, strip packing is used in areas ranging from optimizing cloth fabric usage to multiprocessor scheduling [8]. Algorithms seek to arrange a set of rectangles within a 2D space of fixed width and bottom base, and of infinite height. In our present case, the width of the strip corresponds to the total duration of the piece, and the rectangles to be placed are the action group blocks.

Bin packing is a rapidly expanding research area, with results often being difficult to track and compare [9]. A good measure of algorithm efficiency is the absolute approximation ratio: $\sup_I \text{ALG}(I)/\text{OPT}(I)$, where $\text{ALG}(I)$ is the strip height produced by the algorithm ALG on an input I, and OPT is the optimal algorithm. By calculating a superior bound for any input we obtain a useful value for any input size [10].

Unlike existing bin packing problems, all *AscoGraph* action blocks must retain their *X* coordinate along the time axis. This constraint alone distinguishes our problem from the rest of the bin packing literature. Even multiprocessor scheduling strategies involve expediting or delaying tasks in time [8]. Thus, relying on existing algorithms becomes impractical.

IV. Packing algorithms

We introduce three new algorithms for stacking action groups in *AscoGraph*'s graphical editor. The user can switch between each and the original display style through the application's *View* menu. The appropriate option will depend on score complexity and the user's personal taste.

The interactive nature of the software demands dynamic and periodic calls to block arrangement calculation and rendering. For instance, expanding an action group to reveal its contents might free up space next to it – see Fig. 2. In order to minimise user confusion and computer processing overhead, we take several measures. Firstly, the ordering is computed for all actions in the score, and not just the ones in a specific timeframe; that way, when scrolling through a score at a constant zoom level, recalculation is not necessary. Secondly, we defined a number $n = 10$ of user actions between recalculations. Finally, the most recently clicked action group always stays in place, no matter what reordering process is triggered otherwise.

It is important to note that following bin packing conventions, we shall consider boxes as being placed *on top of* the strip base. Naturally, in the *AscoGraph* environment the situation is mirrored and we build *downwards* starting from the upper border.

In this section, we define a set of rectangles $I = \{r_1 ... r_n\}$, with each r_i being determined by height h_i, width w_i and start time x_i.

A. First Fit (FF)

The first option is the trivial solution of placing the blocks in the first space they will fit, starting from the base. The main strengths of this algorithm are speed and predictability: blocks are placed in the order in which they appear in the source code text, which is also their scheduled temporal order. Since spatial position is in principle the strongest perceptual cue [11], if all groups are considered equally important then it makes sense to place the first occurring elements in the top positions. In section VI we propose filtering as a way to express precedence of certain blocks over others. Other options in this direction remain to be examined.

The downside is evident in Fig. 3a and 3b. Let us consider the worst-case scenario in order to calculate the absolute approximation ratio. Let $I' = \{r'_1 ... r'_n\}$ with increasing heights $h'_i = h'_{i-1} + \varepsilon_i$ and for simplicity, let all rectangle widths be equal. While FF would stack them on top of each other (see Fig. 3a), the optimal configuration would stack them two by two (Fig. 3b), so that the maximum height is given by the final two elements:

$$\frac{\text{FF}(I')}{\text{OPT}(I')} = \frac{\sum_{i=1}^{n} h'_i}{h'_{n-1} + h'_n} \tag{1}$$

It is obvious that a finite upper bound cannot be defined.

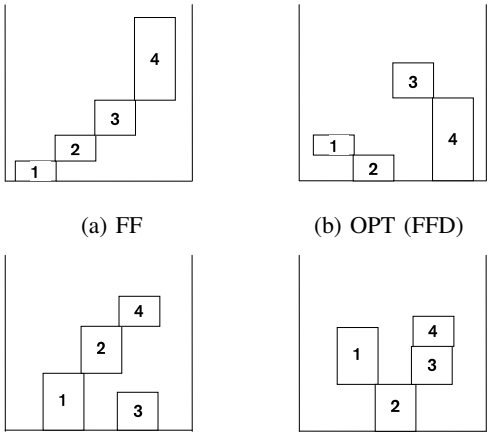
B. First Fit Decreasing (FFD)

Note that in the previous case, the optimal configuration could be reached by simply reordering the blocks by height. This leaves room for blocks of smaller height to drop to the baseline, as we saw in Figures 2 and 4b.

This insight lies at the root of the classic FFDH strip packing algorithm [12]. In our case, the FFD algorithm orders the blocks by non-increasing height, after which the First Fit process is applied.[3] Fig. 3c shows a basic example of an FFD arrangement, along with its optimal solution at Fig. 3d.

It is obvious that, for any $n = 4$ blocks, the above configuration is the one that produces the largest $\text{FFD}(I)/\text{OPT}(I)$ ratio,

[3]The difference to the classic FFDH algorithm is the absence of horizontal levels. New blocks are stacked at the minimum possible altitude rather than a common level.



(a) FF (b) OPT (FFD)

(c) FFD (d) OPT (FFDT)

Figure 3: Horizontally constrained strip packing. The rectangle set in (a) and (b) is first arranged by First Fit, and then optimised by FFD. The set in (c) and (d) is first stacked by FFD, then optimised by FFDT.

by means of the gap between blocks 3 and 4. Generalizing, we can state the following:

Definition 1. Given an FFD layout, a pair of non-adjacent blocks is called a *basic broken pillar*, if the horizontal projection of one block, traversing upwards, intersects the other block.

In Fig. 3c, blocks 2 and 3 form a basic broken pillar.

Definition 2. In an FFD layout, a *broken pillar* is a chain of basic broken pillars where the top block of one is the bottom block of the next.

Theorem 1. *For any configuration of blocks I,*

$$FFD(I) - OPT(I) \leq g, \tag{2}$$

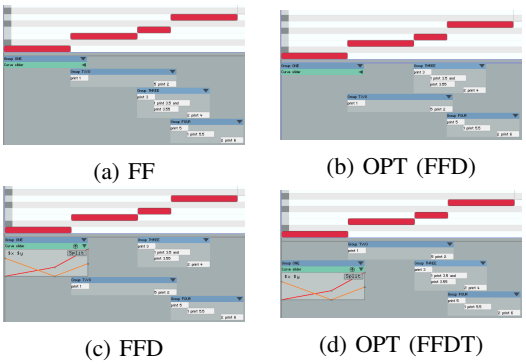
where g is the accumulated vertical gap inside the tallest broken pillar.

Proof: Since, for a basic broken pillar, the reduction to be made by layout optimisation is evidently limited by the basic pillar's internal gap, we can proceed along the chain to obtain the overall maximum reduction. ■

Corollary 1. *For any $n > 4$, the configuration that maximises $FFD(I)/OPT(I)$ is generated by sequentially adding to a broken pillar exclusively.*

Proof: When placing each block, if it is not adding to a broken pillar, then the difference between $\mathrm{FFD}(I)$ and $\mathrm{OPT}(I)$ remains constant. The goal must be to maximise the accumulated gap for the given set of blocks. ■

Corollary 2. *In a broken pillar, if the accumulated vertical gap g is greater than the accumulated height of the member blocks, then g will not contribute to layout optimisation.*



(a) FF (b) OPT (FFD)

(c) FFD (d) OPT (FFDT)

Figure 4: The three *AscoGraph* packing options: (a) FF - action group blocks are placed as close to the horizontal timeline as possible, in the order in which they appear in the code. (b, c) FFD - blocks are first ordered by height before being placed as in FF. (d) FFDT - blocks are first ordered according to a gap-minimisation heuristic before being placed on the timeline as in FFD.

Proof: For a basic broken pillar that satisfies these conditions[4] the conclusion is obvious. Proceeding along the chain, each gap's contribution will continue to be annulled by the stack on the opposite side. ■

It is now possible to present an upper bound to the absolute approximation ratio.

Theorem 2. *The absolute approximation ratio of the FFD algorithm is lower than 2.*

Proof:

$$\sup_I \frac{\mathrm{FFD}(I)}{\mathrm{OPT}(I)} \leq \frac{h_{pillar} + g}{h_{pillar}} = 1 + \frac{g}{h_{pillar}} \tag{3}$$

where g is the accumulated gap inside the tallest broken pillar. If $g \geq h_{pillar}$, then its contribution is annulled and we must look to the second tallest broken pillar. And so on, until the fraction becomes subunitary. ■

C. First Fit Decreasing Towers (FFDT)

As before, the optimal configuration in the previous example hints at the following algorithm. We propose a greedy heuristic that builds upon FFD while tackling situations common in *AscoGraph*, like one action block sharing time with several blocks on both sides of it. (e.g. Fig. 3c and d). The basic goal is to minimise the gaps in broken pillars.

The FFDT algorithm first orders all blocks as in FFD. Then, action group *towers* are defined at the time-axis intersections between two or more group blocks. Their height is equal to the sum of the heights of their component blocks. For instance, in Fig. 3c and d the rectangle 2 is part of four towers: $\mathsf{T}\{r_1, r_2\}$, $\mathsf{T}\{r_2\}$[5], $\mathsf{T}\{r_2, r_3\}$ and $\mathsf{T}\{r_2, r_3, r_4\}$.

[4]The minimum number of blocks to build a basic broken pillar is 5. 2 blocks form the broken pillar itself, and at least 3 blocks are needed (under FFD conditions) to build a large enough gap.

[5]The minimal tower is one that is identical to an (isolated) action block.

The entire width being now split along these virtual vertical strips we call towers, we are now able to refine the ordering of the blocks. Alg. 1 below lists all the steps.

Algorithm 1 FFDT

(1) Compute all tower assignments and heights.
(2) For each block r_i, compute:
- MTH_i = the maximum tower height among the towers containing the block;
- NT_i = the number of towers containing the block;

(3) Order all action group blocks as follows:
(3a) by decreasing MTH_i
if equal **then**
  (3b) by decreasing NT_i;
  if equal **then**
    (3c) by non-increasing block height (as in FFD).
  end if
end if
(4) Place blocks using FF.

The maximum tower height is a definite lower bound of any *AscoGraph* strip packing configuration. Therefore, in the FFDT heuristic the top tower will always be placed first, in an attempt not to overshoot this lower bound. Among its component blocks, the ones that are shared with many other towers are then given preference – the intention again being to maintain tower integrity as much as possible. Lastly, as with FFD, tall blocks are prioritised so as to fill gaps efficiently.

We assert that the FFDT algorithm's absolute approximation ratio is lower than 2 (similarly to FFD), and it is safe to assume it is much closer to 1.

V. Time Coherence of Atomic Actions

As we have shown in section II, atomic actions are instantaneous and ideally should not take up horizontal space. Moreover, as seen in Figures 2 and 4, they introduce unnecessary clutter in the workspace.

Our solution is to group all instances from a specific hierarchical level and display them on a single line as small circles, or conceptual *points*. When the mouse hovers over such a point, a list of the messages it contains is shown. Fig. 5 shows the expanded list for Group THREE; the messages are set at 3 different points in time, which is why 3 points are present in the message line.

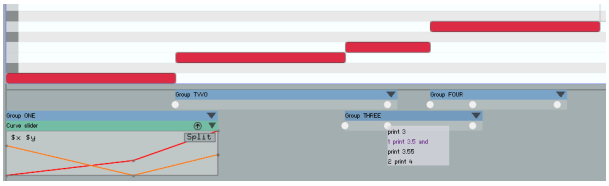
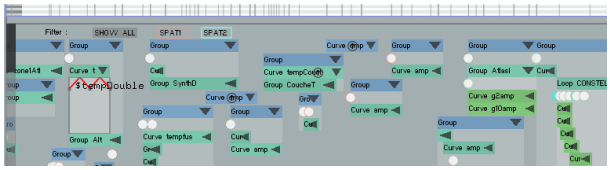


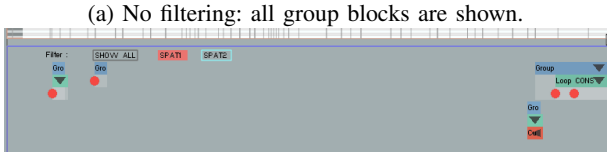
Figure 5: Time-coherent message circles display

This feature also harnesses the visualisation principle of explicit linking [13]. When the user clicks on a certain message in the expanded list, the corresponding line of code is highlighted. Thus, we capitalise on the coupling between adjoining views and their value is enhanced. This feature was already present in *AscoGraph* among other elements: action blocks, musical events in the piano roll, and specific lines of code are all linked and clickable.

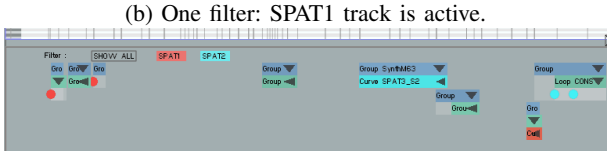
Our new model is fully time coherent and considerably clearer than before. The user experience improvement over the classic model becomes most obvious when dealing with complex scores with many messages.



(a) No filtering: all group blocks are shown.



(b) One filter: SPAT1 track is active.



(c) Two filters: SPAT1 and SPAT2 tracks are active.

Figure 6: Track filtering in *AscoGraph*: A line is added just below the timeline showing all available track names which the user can filter by.

VI. Track Filtering

AscoGraph implements the "visual information seeking mantra" – overview first, zoom and filter, details on demand [14]. While zooming the timeline and expanding blocks to reveal their contents have already been mentioned, filtering is another function essential to information seeking and score navigation.

In *Antescofo*, filters are defined in the score code through regular expressions. The resulting set of objects that match a certain user-defined expression is referred to as a *track* [6]. The user can define up to 32 such tracks.

Fig. 6 illustrates the use of filters. Note that any number of tracks can combine into a filter. At any point, the user can click the SHOW ALL button to return to the initial view.

VII. Action Block Colours

In choosing the colour scheme for *AscoGraph*, we observed the guidelines of Tufte [15] that large areas should use desaturated colour and that high contrast between foreground and background improves readability. We present an additional contribution over the previous version of *AscoGraph*, in which each action block header had a different, randomly assigned colour.

In our new model, header colours signify (1) hierarchy and (2) filters. Unless a block header's colour is overridden by a filter highlight, it will correspond to the block's hierarchical level. First-order action groups have blue headers; afterwards we vary the hue angle in 45 degree increments. Filter highlight tones are purer and more saturated in order to clearly distinguish them from default hierarchical colours.

Message circles are by default transparent white, regardless of hierarchy. Once allocated into tracks, their colours also change accordingly.

VIII. Conclusions and Future Work

We evaluated the current design informally over two phases. During the planning and implementation of the new features, we consulted with the internal community of composers and musicians using *Antescofo* at IRCAM. Their suggestions proved useful in guiding our design. In terms of feedback, there was a lot of enthusiasm over the space saving and clarity provided by the new atomic action display mode, whereas appreciation of the difference between stacking algorithms was generally more gradual. This supports the common insight that algorithm complexity is often unrelated to design impact.

The second phase started with the announcement of the new features on the public *Antescofo* forum[6]. This allowed the improvements to be also informally validated by the *AscoGraph* user-base.

In general, we have achieved increased visual integrity and coherence through the stacking of action group blocks. The most basic stacking method, First Fit, is the most easily readable option for scores of moderate depth. We also proposed two increasingly efficient stacking algorithms, FFD and FFDT, for scores containing larger concentrations of actions per time unit where vertical growth is a concern. While superior algorithms for space optimisation are technically conceivable (possibly a metaheuristic scheme built on top of FFDT), the present options were deemed appropriate for the practical use and the allowed processing overhead of the *AscoGraph* software.

We have also introduced a method of displaying related messages on a single line, preserving group hierarchy. The main advantages are time coherence and vertical compactness. Finally, through the filtering function and the hierarchical colour scheme, navigation through large scores has been found to be drastically improved. All the while, view linking ensures that the navigation process correlates smoothly over all three main views: score code, piano roll, and action blocks.

In the future, we consider employing more techniques to further refine the action view. At wide zoom levels, we might implement block aggregation of neighbouring groups instead of drawing them separately. Semantic zooming might also be of help: in the broad view it is pointless to burden the scene with textual detail.

[6]http://forumnet.ircam.fr/user-groups/antescofo/forum/topic/new-ascograph-display-features-2/

Despite improvements, the *AscoGraph* model is not yet able to visualise some highly dynamic structures in an *Antescofo* score, such as dynamic durations, processes and more as briefly discussed in section II. This points us away from *AscoGraph*'s linear timeline, towards a more flexible model that would accommodate all dynamic constructs in the action language.

References

[1] A. Cont, "Antescofo: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music." in *International Computer Music Conference (ICMC)*, Belfast, Ireland, Aug. 2008, pp. 33–40. [Online]. Available: http://hal.inria.fr/hal-00694803

[2] T. Coffy, J.-L. Giavitto, and A. Cont, "AscoGraph: A User Interface for Sequencing and Score Following for Interactive Music," in *ICMC 2014 - 40th International Computer Music Conference*, Athens, Greece, Sep. 2014. [Online]. Available: https://hal.inria.fr/hal-01024865

[3] M. R. G. Jacquemin, T. Coduys, "Iannix 0.8," in *Actes des Journées d'Informatique Musicale (JIM 2012)*, 2012, pp. 107–115.

[4] G. A. A. Allombert, Myriam Desainte-Catherine, "Iscore: A system for writing interaction," in *DIMEA '08*, 2008.

[5] J. Echeveste, A. Cont, J.-L. Giavitto, and F. Jacquemard, "Operational semantics of a domain specific language for real time musician-computer interaction," *Discrete Event Dynamic Systems*, vol. 23, no. 4, pp. 343–383, Aug. 2013. [Online]. Available: http://hal.inria.fr/hal-00854719

[6] J.-L. Giavitto, A. Cont, and J. Echeveste. Antescofo a not-so-short introduction to version 0. x. [Online]. Available: http://support.ircam.fr/docs/Antescofo/AntescofoReference.pdf

[7] W. Aigner, "Visualizing time-oriented data: A systematic view." *Computers and Graphics*, vol. 31, pp. 401 – 409, 2007.

[8] R. Thöle, "Approximation algorithms for packing and scheduling problems," Ph.D. dissertation, Christian-Albrechts-Universität zu Kiel, 2008.

[9] M. C. Riff, X. Bonnaire, and B. Neveu, "A revision of recent approaches for two-dimensional strip-packing problems," *Eng. Appl. Artif. Intell.*, vol. 22, no. 4-5, pp. 833–837, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.engappai.2008.10.025

[10] R. Harren and R. van Stee, "Improved absolute approximation ratios for two-dimensional packing problems," in *PROCEEDINGS OF THE 12-TH INTERNATIONAL WORKSHOP ON APPROXIMATION ALGORITHMS FOR COMBINATORIAL OPTIMIZATION PROBLEMS (APPROX'09), 177–189*, 2009.

[11] J. Mackinlay, "Automating the design of graphical presentations of relational information," *Acm Transactions On Graphics (Tog)*, vol. 5, no. 2, pp. 110–141, 1986.

[12] J. Coffman, E. G., M. R. Garey, D. S. Johnson, and R. E. Tarjan, "Performance bounds for level-oriented two-dimensional packing algorithms," *SIAM J. Comput.*, no. 9, pp. 808–826, 1980.

[13] J. C. Roberts, "State of the art: Coordinated and multiple views in exploratory visualization," in *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. IEEE, 2007, pp. 61–71.

[14] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *IEEE Symposium on Visual Languages*. IEEE, 1996, pp. 336 – 343.

[15] E. Tufte, *Envisioning Information*. Graphics Press, 1990.