Visualization of Splitting and Merging Processes

K. A. Robbins, C. Jeffery and S. Robbins
Division of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249-0667

Department of Computer Science
University of Nevada Las Vegas
Las Vegas, NV 89120

Abstract

Information about objects that split or merge is often generated dynamically as a by-product of computation or in the observation of real-world behavior. Visualization tools for such processes must not only reveal temporal patterns and spatial organization but should also accommodate on-the-fly generation of split-merge information. This paper develops a formal structure for split-merge processes that provides a suitable underpinning for dynamic visualization tools. The structure allows split-merge processes to be constructed dynamically and supports operations such as concatenation and partitioning while maintaining the underlying split-merge structure. The paper also proposes a method for abstracting split-merge substructure to assist in the visualization of large systems. The abstractions can be used to produce visual simplification and to guide layout heuristics in producing better visualizations. The implications of visualizing split-merge processes with time lines are explored. Auxiliary visualization strategies based on dynamic boxes and tree rings are introduced to enhance information content of focus areas in larger time-line visualizations. The concepts are illustrated using data from timings of a network protocol and from the detection of patterns in scientific video data.

1 Introduction

Objects that split or merge occur both in models of real-world behavior and as a by-product of computational processes. Split-merge information is often generated on the fly, so there may be no information about the scope and complexity of the observed process when visualization begins. These dynamic aspects of split-merge processes are central to the development of appropriate visualization tools. Three common forms of split-merge problems are trees, bundles and proximity merges.

Tree problems may have arbitrary combinations of splitting nodes, but no merging. Examples of tree processes include cell mitosis, the Unix `fork` without a `wait` and broadcast communication with no acknowledgments. In cell mitosis, cells split in a branching process and die at the leaves. After a split, the original cell is no longer identifiable, and two new cells are created. A Unix process can create a duplicate of itself by calling `fork`. Unlike splitting cells in mitosis, the original Unix process retains its identity after the split. Broadcast communication with no acknowledgments can also be represented by a tree. The root node, corresponding to the source message, splits into multiple nodes as the message is received by different processors. Additional splits occur as the message is forwarded.

A second common class of split-merge problems has a bundle structure. In bundles, merging occurs only among descendents of special nodes designated as *bundle roots*, and all descendents subsequently merge back into a single node. Bundle structure is apparent for threads or processes that must eventually rejoin with a parent. Some parallel algorithms such as the computation of a sum on a ring or other regular structure have a bundle structure [6]. Splitting models the distribution of data among the processors, while merging represents the consolidation of data as the computation completes. Multicast and broadcast communication with acknowledgments are other examples of bundle structures [2]. The initial send can be modeled as a multi-way split. Acknowledgments can be modeled by merges of the acknowledging receivers with the sender.

A third class of split-merge problems involves proximity merging. Proximity systems only allow objects that are close enough to merge. Proximity may be applied to spatial position or to other attributes that have a distance measure. If the objects are in motion, their proximity to other objects changes with time. One of the case studies that motivated the present paper involved splitting and merging of cells in a two-dimensional

flame front [7]. The cells could not merge unless they were physically adjacent. Other examples of proximity merging include systems that support group communication with dynamic groups [1] and parallel algorithms that exploit proximity, e.g. algorithms in which processors exchange boundary information with their nearest neighbors [9].

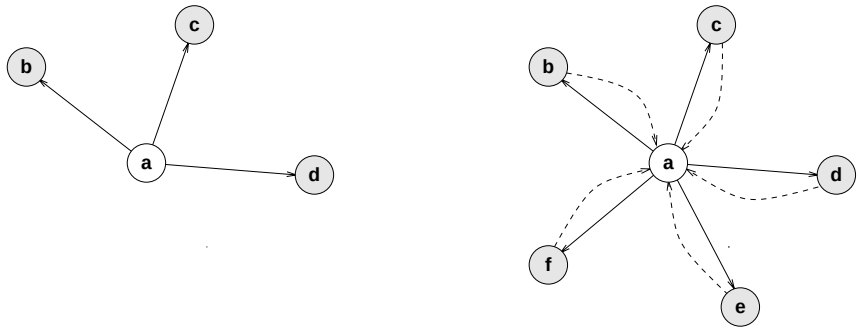
Split-merge processes are a subclass of a large class of problems that can be represented graphically. The primary time axis imposes some hierarchical structure, because splitting and merging can only occur among objects that are alive at the same time. Common hierarchical graph applications include the representation of programs in CASE tools [13], modeling of industrial or other processes [5, 21], routing problems for networks [4] and scheduling algorithms in hardware design [23]. Much of the work on hierarchical graph visualization has focused on clean layout of graphs that are relatively static [3, 8, 10, 15, 17, 22]. However, by their nature, split-merge processes are dynamic and visualizations should emphasize temporal relationships and patterns.

Visualization of splitting and merging processes starts with techniques used to visualize ordinary graphs and trees but must then incorporate the time model associated with the system. Our development is guided by the visualization requirements of several real applications. Operations are defined for concatenating records and for extracting parts of records in order to support panning and zooming visualizations. Section 3 develops a description of time-line visualizations. Section 4 introduces an abstraction operation that visually simplifies complex split-merge processes to allow viewing on different scales. Section 5 presents additional visualizations of history from the viewpoint of individual nodes. These visualizations can enhance information of focus areas in a larger visualization. Section 6 offers some concluding remarks.

2 Modeling of Split-Merge Processes

The difficulties that arise in visualizing split-merge processes can be illustrated by a simple broadcast protocol designed to support certain types of parallel computation on workstations [18]. The data sets are divided into fixed-size packets and broadcast using UDP. Receivers monitor a broadcast port and request retransmission of any packets that they miss. The protocol does not provide for acknowledgment of individual packets, and receivers return an acknowledgment only after they have received the entire data set. Figure 1 shows a simple visual model of the broadcast as a split-merge process. Node a is the broadcast initiator,

and nodes b through f are receivers. The solid arcs connecting a to the other nodes represent the initiation of the broadcast communication. A dotted arc from a receiving node to the initiator indicates that node has acknowledged the completion of the completion of the communication. Figure 1a shows the state after nodes b, c and d have received the initiation. Figure 1b shows the completed communication. Figure 1 does not convey the dynamic nature of the splitting and merging such as the relative ordering of the split events or the time scale over which the split and merge events occur.



a) b)

Figure 1: A simple visual model of a broadcast communication with acknowledgments. Node a is the sender. The remaining nodes are receivers. a) The state after b, c and d have received the initiation. b) The state after the communication has completed.

Time plays an important role in split-merge processes. Usually the data is acquired by making a recording or observation of a physical process over an interval of time. The resulting collection of information is called a *record*. The nodes represent entities that only exist for certain intervals of time. Similarly, the arcs that represent splitting and merging are associated with the time of the corresponding event. We formalize these notions in order to define the data set operations necessary to support visualization primitives such as panning and zooming.

Each node (system object) has a creation time, t_c, and a destruction time, t_d. If node v was created as the result of a split or a merge, then $t_c(v)$ represents the time at which that split or merge takes place. If the node was not created from a split or merge, $t_c(v)$ represents the birth time. If node v exists when the recording of data starts, then $t_c(v) = 0$. If node v was destroyed by a split or merge, then $t_d(v)$ represents the time of that split or merge. A node may also die by transformation (splitting or merging into a single new node) or

by dissipation. If node v persists until the end of the record, then $t_d(v) = \infty$.

Formally, we define a split-merge process $G = (V, E_s, E_m, t_c, t_d)$ as a quintuple, where V is a nonempty finite set, E_s and E_m are subsets of $V \times V$, and t_c and t_d are nonnegative functions on V. G satisfies the following conditions:

1. If $v \in V$ then $t_c(v) < t_d(v)$.
2. If $(v, w) \in E_s$ then $t_c(v) < t_c(w) \leq t_d(v)$.
3. If $(v, w) \in E_m$ then $t_c(w) \leq t_d(v) < t_d(w)$.
4. $E_m \cap E_s = \emptyset$.

The functions t_c and t_d are the node creation and destruction times. Condition (1) states that a node must be born before it is destroyed. E_s contains edges that represent splits (i.e. if $(v, w) \in E_s$, then node v undergoes a split at time $t_c(w)$ that results in node w). The time of the split is taken to be $t_c(w)$. The inequality $t_c(v) < t_c(w)$ in condition (2) states v must be created before it can split into node w. This inequality implies that (V, E_s) is a DAG (directed acyclic graph). The inequality $t_c(w) \leq t_d(v)$ states that v must exist until the split occurs. If $t_c(w) = t_d(v)$, node v is destroyed by the split operation, while if $t_c(w) < t_d(v)$, v splits into w and the nodes coexist after the split.

E_m contains edges that represent merges (i.e. if $(v, w) \in E_m$, then node v merges into w at time $t_d(v)$). The time of the merge is taken to be $t_d(v)$. The inequality $t_d(v) < t_d(w)$ in condition (3) states that the resulting node, w, must be alive after the merge. This inequality implies that (V, E_m) is a DAG. The inequality $t_c(w) \leq t_d(v)$ states that w must exist at the time that the merge occurs. If $t_c(w) = t_d(v)$, node w was created by the merge operation, while if $t_c(w) < t_d(v)$, w existed before the merge. Conditions (2) and (3) together imply that no split or merge edge introduces a cycle. These conditions also imply that $(v, v) \notin E_m$ and $(v, v) \notin E_s$ for all $v \in V$.

Physical systems represented as split-merge processes have edges that correspond to physical transitions. A given edge might satisfy the inequalities in both (2) and (3) of the definition. In this case the researcher may classify the transition as either a split or a merge. The determination should be based on whether the

splitting or the merging is the focus of attention. For example, if merging is more important, then classifying ambiguous arcs as split edges may be preferable.

2.1 On-the-Fly Generation

Because split-merge processes are dynamic, the definition must accommodate on-the-fly generation. An object that exists at the start of observation begins with a creation time of 0 and a destruction time of ∞. An object that exists at the end of the interval observed thus far is assigned a destruction time of ∞.

When node v splits and w is created as a result, an arc (v, w) is inserted into E_s and the node w is inserted in V. The value of $t_c(w)$ is set to the time of the split, and $t_d(w)$ is set to ∞. If v is destroyed by the split, $t_d(v)$ is also set to the time of the split.

Similarly, when v merges into node w, an arc (v, w) is inserted into E_m, and $t_d(v)$ is set to the time of the merge. If w is created by the merge, it is inserted into V. In this case $t_c(w)$ is set to the time of the merge, and $t_d(w)$ is set to ∞. At each instant of time, the structure constructed in this way is a split-merge process.

2.2 Concatenation and Partitioning

In this section, we define concatenation and partition operations that are analogous to the familiar concatenation and substring operations on strings. These operations are necessary for pan and zoom operations in visualization since they allow split-merge processes to to be combined or divided into pieces for closer examination. The following definitions are helpful.

The *start set*, $S(G)$, of a split-merge process $G = (V, E_s, E_m, t_c, t_d)$ is the set of nodes of V whose creation time is 0 (i.e. those nodes that were in existence when the recording of the process started). Similarly, the *final set*, $F(G)$, is the set of nodes of V for which t_d is ∞. The final set represents those nodes that are still in existence when the recording of the process ends. The maximum finite value of the t_c's and t_d's, $T_{max}(G)$, represents the length of the observation interval for G.

The *concatenation* of split-merge processes G_1 and G_2 is denoted by $G = G_1||G_2$. The sets V, E_m and E_s of G are formed by taking the union of the corresponding sets of G_1 and G_2. The concatenation is proper when the final set of G_1 is the starting set of G_2 and these are the only nodes that the processes have in

common, i.e. $F(G_1) = S(G_2) = V_1 \cap V_2$. Proper concatenations support the clean slicing and reassembly operations needed to preserve data integrity in the presence of panning and zooming.

The time functions t_c and t_d for G are defined as follows. For the nodes only in G_1, the t_c and t_d values are the same as in G_1. For nodes only in G_2, the t_c and t_d values are calculated by taking the corresponding values from G_2 and adding $T_{max}(G_1)$. For nodes in both G_1 and G_2, the t_c values are taken from G_1 and the t_d values are calculated by adding $T_{max}(G_1)$ to the corresponding values of t_d from the second process.

The *translation* of a split-merge process involves adding a constant to all of the finite creation and destruction times. The translation operation can be used in conjunction with concatenation to insert split-merge processes in an absolute time frame.

The *partition* of a split-merge process G, denoted by $G[t_1, t_2]$, creates a split-merge subprocess in a manner analogous to the substring operation on strings. Assume that $0 \leq t_1 \leq t_2$, for otherwise the partition is empty. A node v from G is in the partition if it is alive during the interval $[t_1, t_2]$, that is $[t_c(v), t_d(v)] \cap [t_1, t_2] \neq \emptyset$. In this case the creation time of v is 0 if $t_c(v) \leq t_1$ and $t_c(v) - t_1$ otherwise. The destruction time of v in the partition is $t_d(v) - t_1$ if $t_d(v) \leq t_2$ and ∞ otherwise. A split edge (v, w) of G is in the partition if $t_c(w) \in [t_1, t_2]$, and a merge edge (v, w) of G is in the partition if $t_d(v) \in [t_1, t_2]$.

2.3 Size, Scaling, and Granularity

A general graph with n nodes can have $O(n^2)$ edges. Split-merge processes on the other hand have $O(n)$ edges. This result can be seen as follows. Consider the split-merge process $G = (V, E_s, E_m, t_c, t_d)$. If G has n nodes, $|V| = n$. Each node has exactly one creation time (which is possibly 0) and one destruction time (which is possibly ∞). Each edge (v, w) in E_s is uniquely identified by $t_c(w)$, while each edge (v, w) in E_m is uniquely identified by $t_d(v)$. Therefore $|E_s| \leq n - S(G)$ is $O(n)$ and $|E_m| \leq n - F(G)$ is $O(n)$. The inequality results because nodes may be born spontaneously or die spontaneously, and the corresponding creation (or destruction) times are not associated with split (merge) edges.

The number of edges and vertices is not sufficient to characterize the complexity of a split-merge process from the viewpoint of visualization because of its dynamic nature. A process in which a large number of nodes exist, but only a few at a time is likely to be visually simpler than a process with fewer nodes that have long lifetimes. The *diameter*, $D(G)$, of a split-merge process, G, is the maximum number of nodes

that exist at the same time. Another relevant measure is the average diameter, $D_a(G)$, which is defined as the average number of nodes that exist simultaneously.

The diameter of a process can be adjusted, to some extent, by changing the granularity of the time scale. The number of unique times at which splits and merges take place is less than $2n - F(G) - S(G)$. These unique times can be sorted and labeled by the time at which they occur to form an alternative time-scale for G based on *event time* rather than *actual time*. Whereas actual time is considered to be a continuous variable, event time is discrete.

Another possible discretization of time involves dividing the time scale into intervals and treating events that occur within a single interval as occurring at the same time. The sizes of the intervals may be based on physical time scales or merely convenient dividing points and do not need to be of equal size. Selecting larger time interval bins shortens the time record but usually increases the diameter. The trade-off depends on the application. A meaningful granularity is sometimes imposed by the application as discussed in the next section. In any case, care must be taken not to allow splits and merges of the same node to fall in the same bin.

3 Time-Scales and Time-Lines

This section explores the interpretation of split-merge visualization based on time lines. Figure 2a shows a time-line representation for one possible ordering of events for the network communication depicted in Figure 1. The horizontal scale corresponds to event time based on numbering events chronologically. In this example, receiving nodes processed the initiation in the order b, c, d, e and f. The communications complete in the order c, d, f, e and b. Distinct copies of node a were required to display communication completions as distinct events. This replication introduces an asymmetry in the representation of initiation and completion. Also, in order to see which nodes exist at the time of a particular event, one must follow the arcs that cross the vertical line marking that event.

Figure 2b shows an alternative representation in which each node is physically replicated for all event times during which it exists. This representation generally improves visual appearance of the graph and makes the state at a particular time easier to determine. Hierarchical graph drawing algorithms often add ghost nodes at intermediate event times so that the layout more clearly conveys graph organization [22].

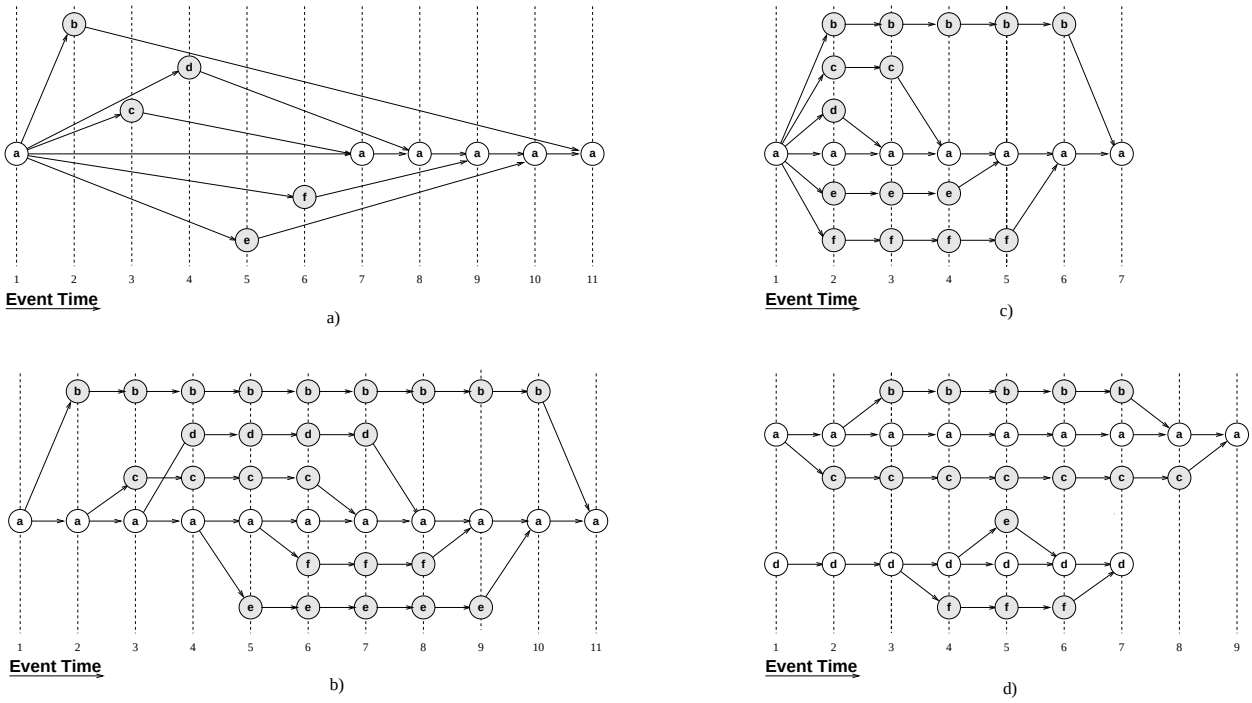


Figure 2: In event-time representations, time is numbered by distinct events. a) Node a is replicated to represent distinct communication completion times. b) A time line in which all nodes are replicated at intermediate event times. c) A visual model that uses binning to group closely-spaced events into one event time. Since the initial splits occur almost simultaneously, they are grouped in a single time. d) If events from two communications interleave, the perceived structure of identical communications will be different as illustrated by these time lines based on event time.

In the broadcast protocol, receivers usually process a sender's initiation within milliseconds of the actual start of the communication. In contrast, the time to complete the entire communication (roughly the time interval between the first split and last merge) is usually on the order of seconds. Although the small time differences in initiation at different nodes are represented by staggering them horizontally in Figure 2a, the much greater variability in completion times is not shown at all. In Figure 2c the initiation events are treated as occurring at a single event time. With this approach users specify the granularity of event time. Events that happen within a specified time interval are labeled with the same event time. By varying the event time bin size, the user can view events on different time scales. Binning makes sense when groups of split events or groups of merge events are clustered in time, but splits and merges are widely separated in time.

Figure 2d illustrates the importance of viewing time lines in conjunction with other types of structural visualization. In this figure, two communications (initiated by a and d) occur during the same time interval. The two communications have exactly the same structure, yet appear to be different because the milestones of the respective communications are interleaved.

The time-line representation is illustrated in Figure 3 by an application involving cell splitting and merging in a combustion experiment [7]. In this experiment, a flame front oscillates irregularly between having three, four or five cells. The clip shown at the top of the figure starts with three cells. The cell in the upper left corner splits into two cells by the third frame. In the fourth frame another cell is starting to dissipate. An irregular sequence of split-merge events continues for the entire experimental run. The videotape of the experiment provides the equivalent of direct animation of the split-merge process, but it is difficult to understand structure from direct viewing of the tape. The time-line visualization provides a better summary of overall behavior. In order to construct a time-line visualization from the video, an image processing system was built to track cells from frame to frame in digitized video tape [24, 25]. A split-merge process was extracted from the tracking information.

Figure 3 displays the time-line representation derived from 1000 frames of videotape of the experiment. The vertical grid lines show the distinct event times for the system. Each node, v, is represented by a horizontal line that begins at $t_c(v)$ and ends at the event time immediately preceding v's death. Split events are indicated by cyan arcs, and merge events are indicated by red arcs. Two color bars are across the top display other features. The first color bar represents the elapsed time between each event. The color bar uses

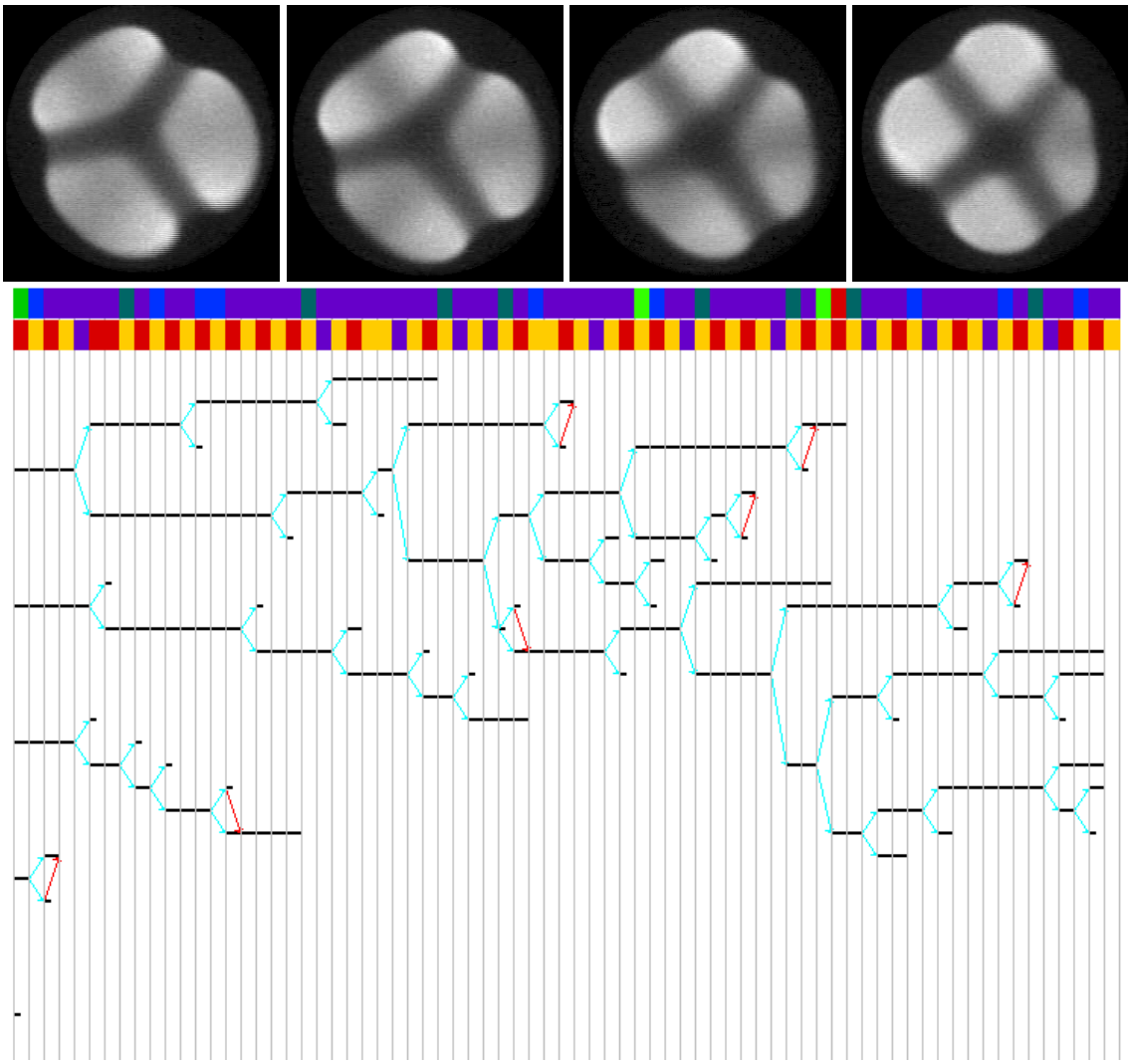


Figure 3: A time-line representation of the split-merge process represented by the combustion experiment in which a flat flame front flutes and forms cells that split and merge. The images across the top of the figure were extracted from consecutive frames of video data of the combustion experiment to produce the time line. The first color bar across the top gives a color-coded representation of the elapsed time between each event, with violet corresponding to the shortest time and red corresponding to the longest time. The second color bar represents the number of cells with red corresponding to 5 cells, yellow to 4 cells, and violet to 3 cells.

a linear scale with red representing the largest time and violet representing the shortest time. The second color bar represents the number of cells in existence. Red corresponds to 5 cells, yellow to 4 cells, and violet to 3 cells.

Figure 3 indicates that the system oscillates in a irregular manner between 3, 4 and 5 cells. 72 of the 1000 frames in the original videotape had split-merge events. Initially, the system has 5 cells. Most events correspond to a cell splitting into two children. The original cell does not exist after these splits. The few merges that occur in the video involve a cell splitting into two cells and then immediately merging. The color code for actual elapsed time on the upper color bar is violet indicating that the cells were short-lived before merging. This behavior can be understood in terms of the data acquisition process. The split-merge data for this experiment was derived by digitizing videotape of an experiment and applying image processing techniques to track cells. The distinction between a split followed by an immediate dissipation and a split followed by an immediate merge is somewhat arbitrary. The tracking program classifies the event based on the position of the surviving cell relative to the two split cells. A slightly different selection criterion would reclassify some of the merges as dissipation events.

Figure 3 shows that the cell displayed at the top of the graph is the source for most of the activity. The graph also shows that all of the split events that occurred when there were 4 cells were short-lived. There was one very long interval between events (corresponding to the red block at the top) in which 5 cells persisted. There only appear to be 4 cells during this interval, because lines representing dissipating cells stop at the event time before their death. This representation displays cells that die by a split or merge in the same way as cells that die from dissipation.

The color bars in Figure 3 provide a temporal reference that allows the user to make a correspondence between event time, real elapsed time and the temporal distribution of cells. Color bars can be coded to indicate general information such as the number of nodes at each time, the total number of splits and merges at each time, and the average in-degree or out-degree of nodes in existence at each time. In the latter case a node that splits into two counts 1, while a node that splits into three counts 2. Merges are handled similarly. Several color bars can be displayed simultaneously to present more visual structure for navigation. The color bar scale can also be set so that multiple levels are combined and represented as a single block for viewing the overall structure of large split-merge processes.

4 Viewing Structure on Different Scales

The trade-off between global views and visibility of details is a long-standing problem in visualizations of large data sets [14, 16]. Traditional distortion techniques such as fish-eye views can be used to focus on detail in particular areas [11, 12, 19, 20]. Partitioning is convenient for restricting a visualization to a specified interval of time, but visualizations lose their global view when such restrictions are made. Another approach is to restrict views based on substructure. Obvious candidate substructures are the connected components. Connected components are independent of each other, but they are problematic because information about whether two objects are in the same connected component may not be available until long after the objects have been created. Many of the more interesting split-merge applications have a combination of very large and very small components that do not lend themselves to viewing on multiple scales.

This section describes an iterative technique for producing coarser-grain structural information based on a generalization of split-merge processes called *merge processes*. Merge processes can be *abstracted* by iteration to produce successively simpler, higher-level representations of structure.

A merge process $M = (V, E_s, E_m, t_c, t_d, t)$ is similar to a split-merge process except that the second inequality in condition 3 is eliminated. If $(v, w) \in E_m$, then we only require that $t_c(w) \leq t_d(v)$. The merge process includes an additional nonnegative function t on $E_s \cup E_m$ that represents the time of the arc. Every split-merge process, $G = (V, E_s, E_m, t_c, t_d)$, is also a merge process, M, with the natural identification for V, E_s, E_m, t_c and t_d. If $(v, w) \in E_s$, then $t(v, w) = t_c(w)$. Similarly if $(v, w) \in E_m$, then $t(v, w) = t_d(v)$. A *split process* can also be defined in an analogous manner.

Formally, we define a merge process $M = (V, E_s, E_m, t_c, t_d, t)$ as a sextuple, where V is a nonempty finite set, E_s and E_m are subsets of $V \times V$, and t_c and t_d are nonnegative functions on V. t is a nonnegative function on $E_s \cup E_m$. M satisfies the following conditions:

1. If $v \in V$ then $t_c(v) < t_d(v)$.

2. If $(v, w) \in E_s$ then $t_c(v) < t_c(w) \leq t_d(v)$.

3. If $(v, w) \in E_m$ then $t_c(w) \leq t_d(v)$.

4. $E_m \cap E_s = \emptyset$.

5. If $(v, w) \in E_s$, $t(v, w) = t_c(w)$.

6. If $(v, w) \in E_m$, $t(v, w) = t_d(v)$.

We now define an abstracting function $\mathcal{A} : M \to M'$ that transforms merge process M into another merge process $M' = (V', E'_s, E'_m, t'_c, t'_d, t')$ that is at least as simple as M. M' is a visual abstraction of M. The abstracting function $\mathcal{A}$ can be applied iteratively until a fixed point is reached. The nodes in V' are called *merge seeds*. V' consists of the nodes in V which do not arise solely because of a single split. The *merge seeds* are those nodes that are not the destination of exactly one split arc. This definition includes nodes that are not the destination of any arc, nodes that are the destination of two or more arcs (of any kind) and nodes that are the destination of exactly one merge arc. The functions t'_c and t'_d are the same as the corresponding functions in M, but these functions are only defined for nodes in V'.

The *seed set* of merge seed $v' \in V'$, $S(v')$, consists of v' and the non-seed nodes that are reachable from v through a path that does not contain any other merge-seeds. The definition of merge seeds implies that this path must be unique and consists solely of split arcs. If $v' \in V'$, then $t'_c(v')$ is the minimum of $t_c(v)$ for all nodes v in the seed set of v'. Similarly, $t'_d(v')$ is the maximum of $t_d(v)$ for all nodes v in the seed set of v'. Suppose (v, w) is an arc in E_s, v' is the seed of the seed set containing v, w' is the seed of the seed set containing w, and $v' \neq w'$. Then (v', w') is an arc in M'. This arc will always satisfy $t'_c(w') \leq t'_d(v')$. If it also satisfies $t'_c(v') < t'_c(w')$, then this arc is in E'_s and otherwise it is in E'_m. If $(v', w') \in E'_s$, then $t'(v', w')$ is the minimum of $t(v, w)$ when v is in the seed set of v', w is in the seed set of w' and (v, w) is an arc of M. Similarly, if $(v', w') \in E'_m$ then $t'(v', w')$ is the maximum of $t(v, w)$ where v is in the seed set of v', w is in the seed set of w' and (v, w) is an arc of M.

A biological analogy to the seeds can be made by thinking of a seed as containing genetic material. If a vertex splits, the genetic material is cloned so that the split vertices are identical. A vertex that is the result of a merge has chromosomes containing a new combination of genes from the vertices that merged. It also possible to define *split processes* and *split seeds* that in a manner parallel to the definitions for merging just defined.

Figure 4 illustrates abstracting for the combustion experiment of Figure 3. Figure 4a shows a time-line representation of the original data set, while Figure 4b shows the time line after one level of abstraction. The simple bundle merges in the original data become splits on the first iteration of $\mathcal{A}$. The split structure is abstracted by the seed nodes. The arcs in the abstracted merge process represent connections between sets of objects allowing coarser views of complex systems. On the next iteration of $\mathcal{A}$ (not shown), each connected component is collapsed onto a single node and the time-line representation gives 5 horizontal lines displaying the lifetimes of the seed sets.

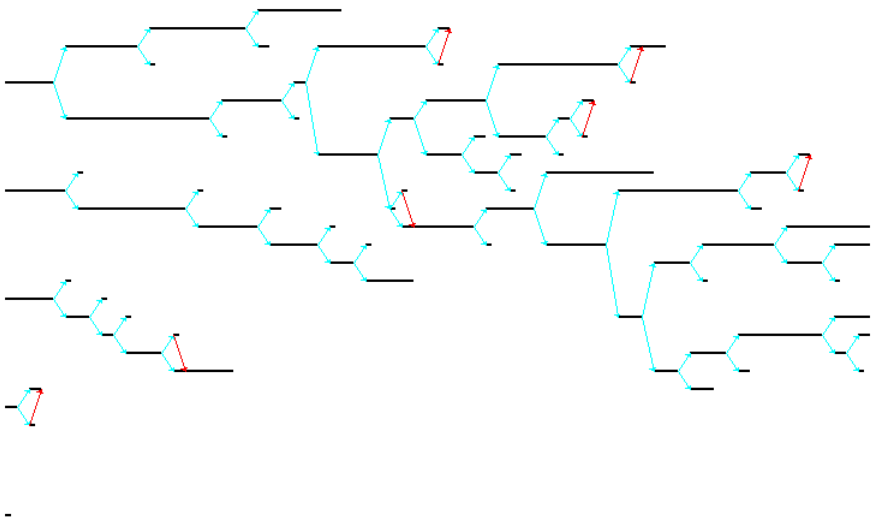
A more complex example from another combustion experiment is shown in Figure 5. Initially, there are 19 objects. The top four time lines represent objects that don't split or merge. The remaining cells undergo complex sequences of splits and merges. Several sections are boxed in Figure 5. The corresponding sections after one level of abstraction are compared with the original sections in Figure 6.

At the beginning of Box a) two cells exist. The lower cell merges into the upper cell about 1/4 of the way through. The upper cell, which lives for the entire interval, later splits into another cell which in turn splits again. On one iteration of $\mathcal{A}$, the merge is preserved, but all additional splitting is removed.

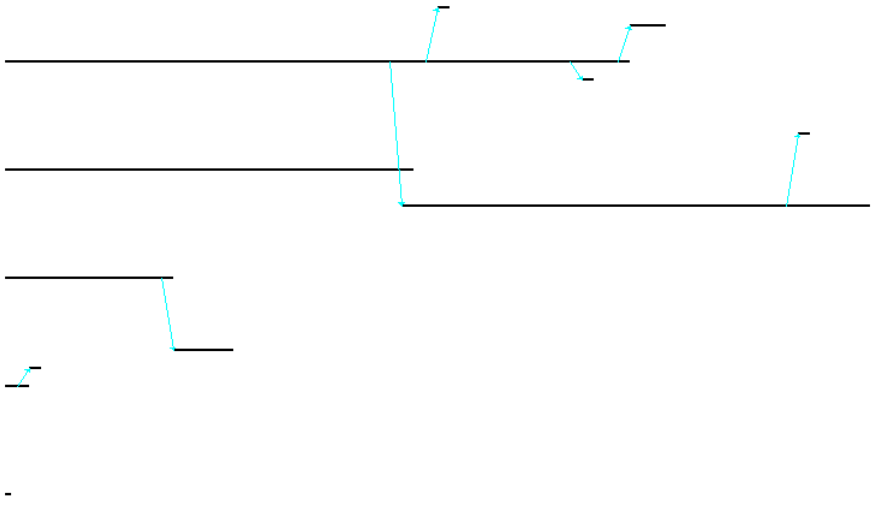
Box b) also initially has two cells. The upper cell splits off a second cell which in turn undergoes several splits. One of the cells that splits off later merges with the lower original cell. Under the action of $\mathcal{A}$, all of the split structure is removed. The merge is represented by an edge from the top cell to the bottom cell. Notice that the merge edge is drawn at the time of the birth of the child that merges into the bottom cell.

Box c) starts with four cells. The middle cells merge into the top cell. These merges are preserved under the action of $\mathcal{A}$. A child of the bottom cell also merges with the top cell. This merge is represented by an arc from the bottom cell to the top cell drawn at the time of the birth of the merged child. Later the bottom cell splits off a child and merges into that child. This child is not in the seed set of the bottom cell, because it is the destination of a merge. Because the original cell dies in the merge operation, the merge arc becomes a split arc under the iteration of $\mathcal{A}$.

Compare Box c) with Box d). Box d) contains three original cells. The upper cell splits into a child. Later the child merges back with the original upper cell. Because the merged child was in the seed set of the original cell, it is collapsed into the original cell under abstraction. Later the original upper cell splits into another child. This child spawns three additional cells, two of which merge back. Because the child is the



a)



b)

Figure 4: The time-line representation and abstraction of the combustion experiment shown in Figure 3. a) The time-line representation of the video data. b) After one iteration of $\mathcal{A}$, all of the splitting structure is collapsed and the remaining branches indicate merges.

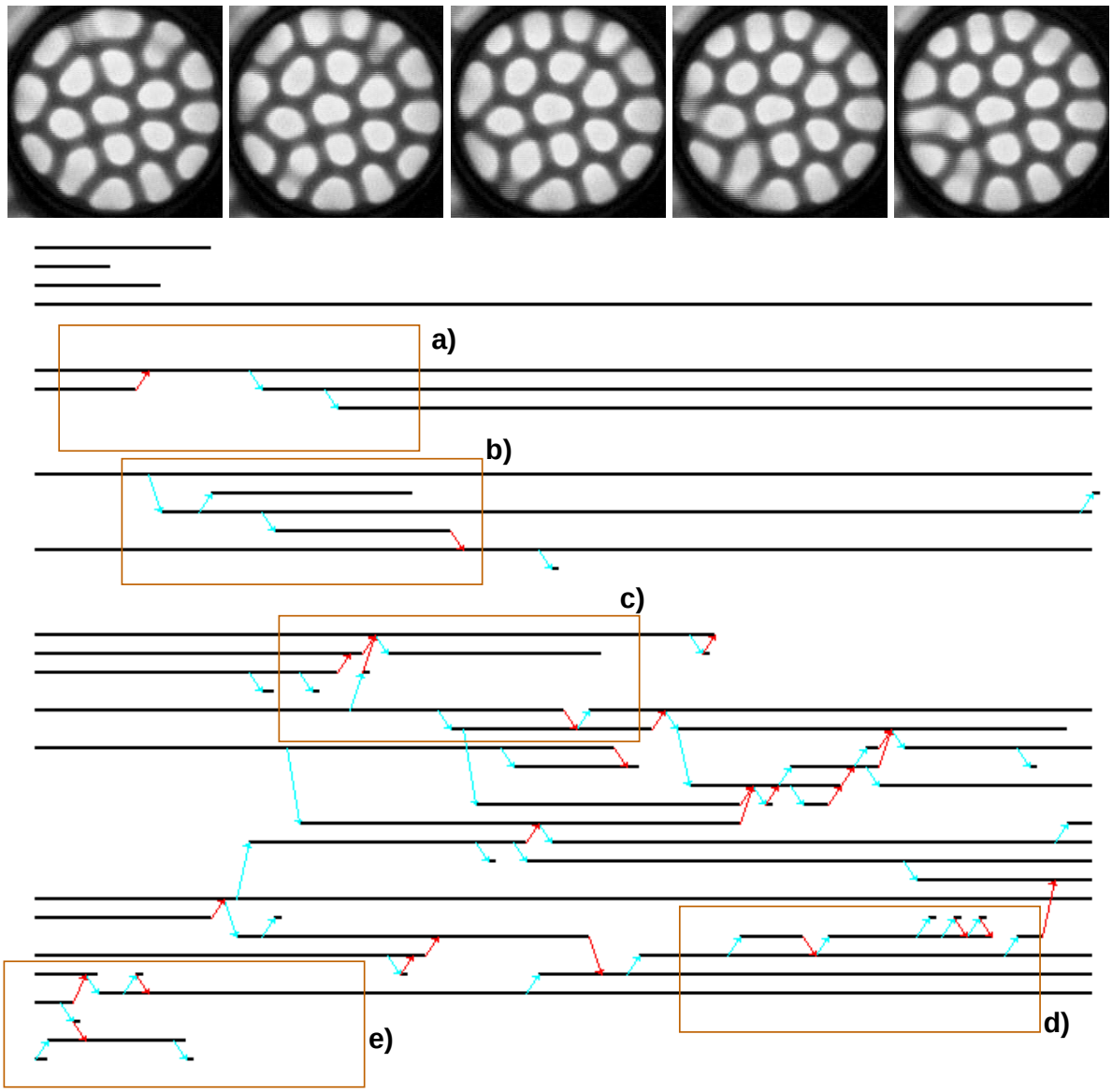


Figure 5: The time-line representation for a combustion experiment exhibiting unstable rings of cells. The images across the top are consecutive frames of videotape of the experiment. Several sections are outlined in the time line for comparison with the corresponding boxes in Figure 6.

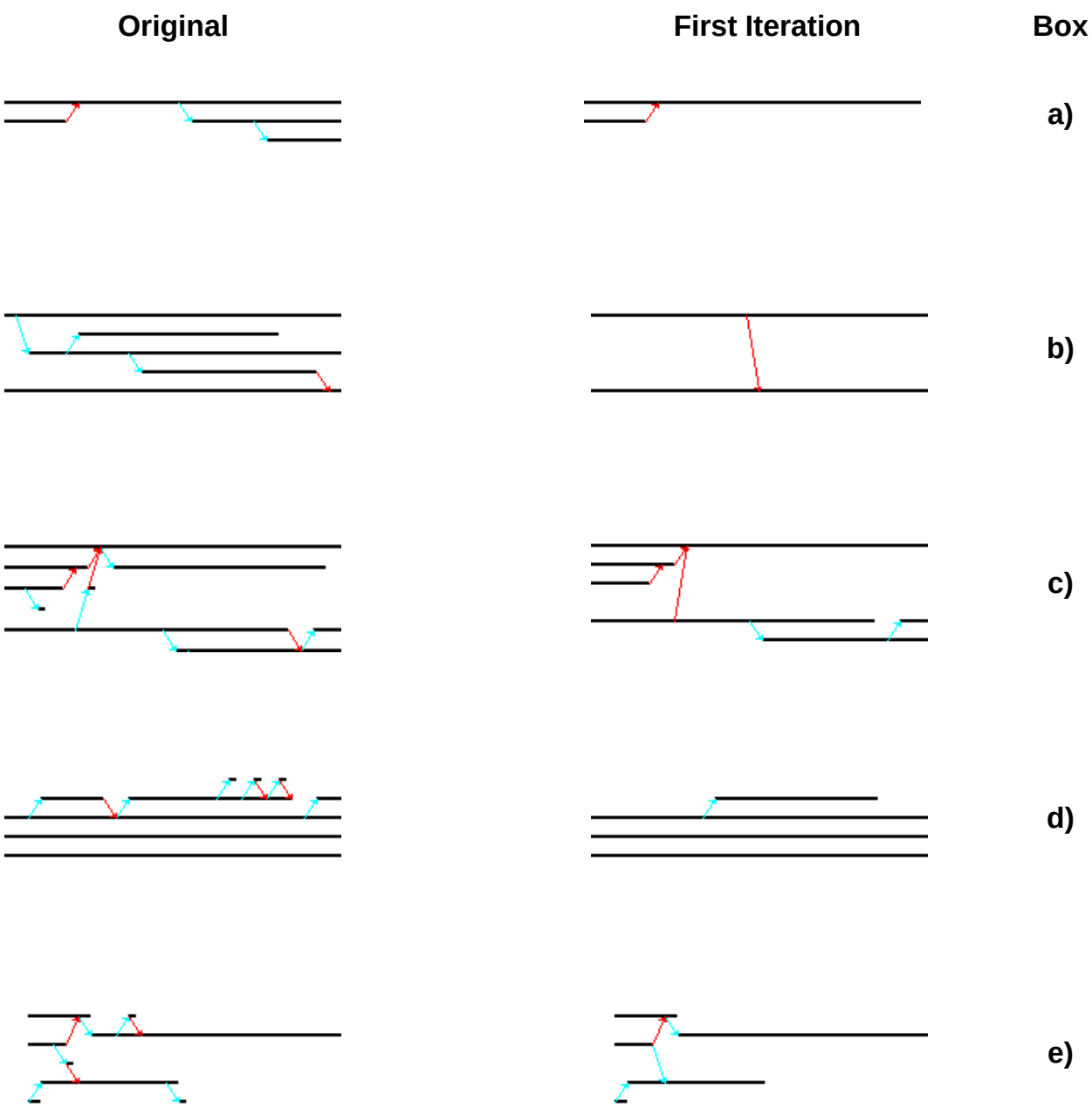


Figure 6: Comparison of the original time lines of the boxed sections in Figure 5 with the time lines under one iteration of $\mathcal{A}$.

destination of two merges, it is not in the seed set of its parent. It therefore is preserved under abstraction. Box e) also illustrates the distinction between seed nodes and non seed nodes. Figure 7 shows the time-line representation after two iterations of $\mathcal{A}$. The basic structure is preserved, but much of the detail has been abstracted.

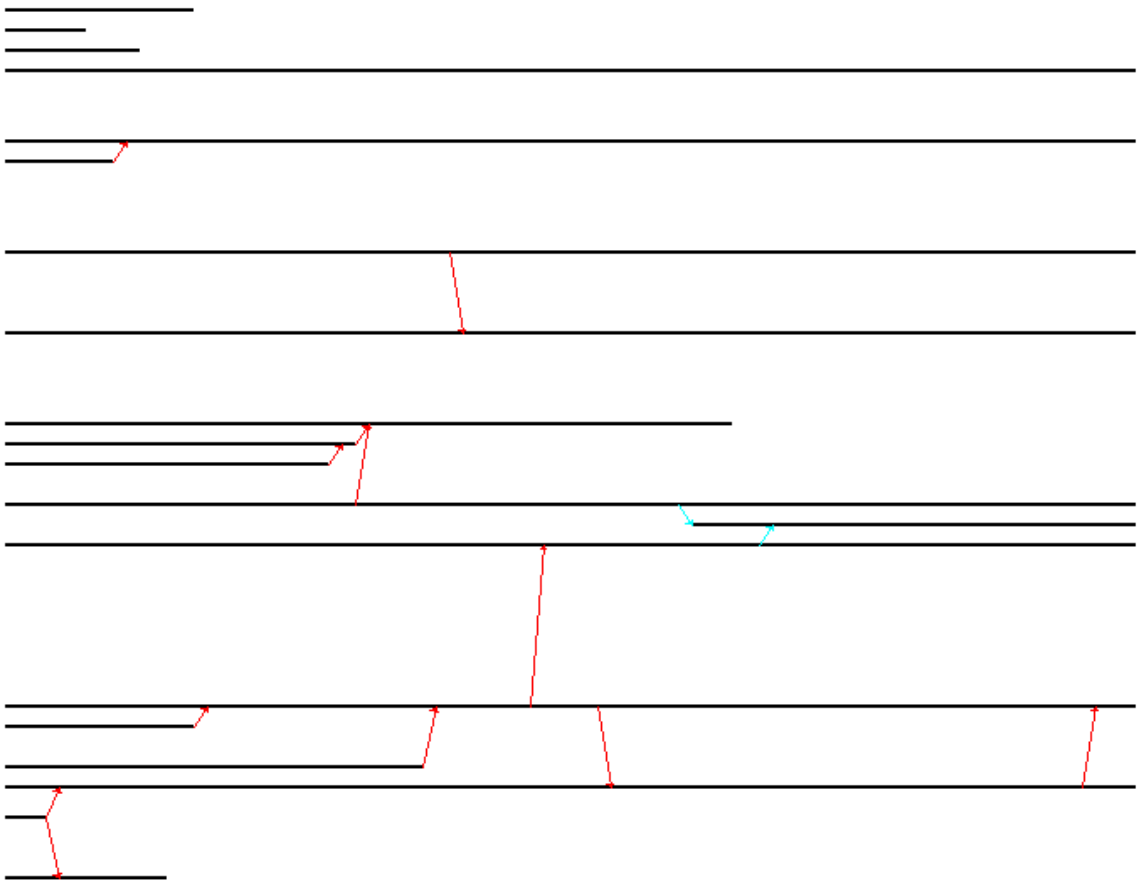


Figure 7: The time-line representation of a combustion data set after two iterations of $\mathcal{A}$. Compare with the time line for the original data shown in Figure 5.

5 Layout, Navigation, and Focus

Split-merge processes are dynamic, and the definition presented in this paper specifically accommodates on-the-fly generation. As splits and merges occur, nodes and arcs are added to the process, while still preserving

its split-merge structure. Similarly time lines can be maintained and updated as new information becomes available. This section focuses on layout and navigation problems that arise because of the dynamic nature of split-merge processes.

5.1 Layout

Time lines are naturally visualized using a strip-chart metaphor with time (either actual or event) progressing horizontally. In the time line shown in Figure 3, each node is represented by a horizontal line that begins at the time of the node's birth and ends at its death. Splits are shown branching off to lines at different vertical positions, while merges show lines coming together. The time-line representation is a special type of *layered digraph* or *hierarchy*, where nodes are constrained to lie on a set of equally spaced vertical lines called *layers*. The excellent review article by Battista et al. [3] provides an extensive discussion of work on automatic layout of such graphs.

The general problem of finding a layout that minimizes the number of edge crossings for layered digraphs is NP-hard. Our problem has the additional constraint that edges connecting the same node at successive times should be horizontal. While it is possible to come up with heuristics that provide reasonably good layout in these circumstances, that is not the focus of this paper. Our tools make a minimal attempt to optimize based on the information in the available record and then permit the user to adjust the position of the lines as desired.

The iteration function $\mathcal{A}$ is useful in providing layout heuristics. Given a merge process M, one can first apply a layout optimization function to an iteration of M under $\mathcal{A}$ to establish a good structural location for the *seeds* of the process. One can then display the seed sets, while constraining the seeds to be in the positions established under $\mathcal{A}$.

5.2 Navigation

For time lines that do not fit on a screen, one can devise simple navigation tools based on panning through the history similar to a strip chart recording device. The user can drag a slider along the bar to step ahead in the visualization (drag-through) or can play through the visualization at a fixed rate (play-through). The rate can be related to real time or to time between events. Other types of navigation are also possible. By

drawing a box using the mouse, the user can restrict the view to a particular part of the process. Seed sets can be highlighted and removed or displayed by themselves.

5.3 Enhancement of Focus

The ability to restrict a visualization to a small focus area is important for large scale processes. Summary information about the splitting and merging history within a focus area can be displayed to facilitate the comparison of individual nodes. We discuss two types of node history visualizations — dynamic boxes and tree rings. The *dynamic box* representation [24, 25] shows the order of splits and merges that have occurred to produce an object. The *tree ring* representation shows the relative timing of splits and merges of each object. In a sense, the dynamic box representation displays the past, while the tree rings show the future.

In the dynamic box representation, an object that has undergone no splits or merges is represented by solid color square. When an object splits in two, the square representing the original object is replaced by a square consisting of two horizontal rectangles. The lower rectangle in each square contains the pattern displayed in the original square shrunk to half its height. The upper rectangle in each square contains a new color. Dynamic boxes should not be confused with tree-maps [8]. A dynamic box encapsulates history from a leaf-node looking back, while tree-maps usually depict successors of the tree root.

Dynamic boxes visualize the order of split-merge events that a specified node went through to reach the current time. Figure 8a shows an object that has undergone a single split. Figure 8b shows an object that has undergone three past splits. The dark horizontal strip across the top half of the image indicates that the last operation was a split. One can tell what the node looked like prior to the split by removing this strip and rescaling vertically by a factor of 2. Figure 8c shows the result of merging the objects of Figure 8a and Figure 8b. The fact that the last operation was a merge is evident from the top half of the resulting square, which does not consist of a single color.

Dynamic boxes can be used in two ways. As a static visualization, the box representation gives a snapshot of the system at a particular time. Each large square corresponds to an object in existence at that time. The structure within each large square represents a compressed view of the split-merge history of the corresponding object. It shows the structure and order of the sequence of splits and merges that occurred to obtain that square. Dynamic boxes can also be used in animations of split-merge behavior. As splits

a) b) c)

Figure 8: a) An object that has undergone a single split. b) An object that has undergone three splits. c) The result of merging the objects shown in a) and b).

and merges occur, the squares rescale and physically split or merge so that at each time during the running animation, the large squares give the state of the system.

Dynamic boxes give a structural representation of split-merge history, but they do not preserve timing information. They also have limited scalability and are best used to compare individual nodes in a small focus area. Tree-rings, which are based on a biological tree metaphor, visualize both timing and structure and have better scalability properties. Tree cross-sections generally have a ring for each season of growth, and the characteristics of the individual rings give information about the environment during that growth season. In the tree ring metaphor for splits and merges, radial distance indicates time with the birth at the center. At a given radial distance, pieces of arc represent descendent nodes in existence at that time. A node that neither splits nor merges is represented by a circle whose radius indicates the lifetime of the node.

Consider the splitting structure represented by the traditional tree graph shown in Figure 9a. Node a splits into nodes b and c. Node c in turn splits into nodes d, e and f. A simple graphical representation of splitting does not convey information about the timing of split events, nor does it distinguish between the case when node a splits into nodes b and c and disappears in the split and alternative scenarios (e.g. node a first splits off b and later splits off c and all three nodes coexist).

The tree ring visualizations on the other hand, convey both descendent information and timing scenarios. Figure 9b shows a scenario in which node a no longer exists after it splits off nodes b and c. Node a is represented by the inner circle. The radius of the inner red circle represents the lifetime of node a relative

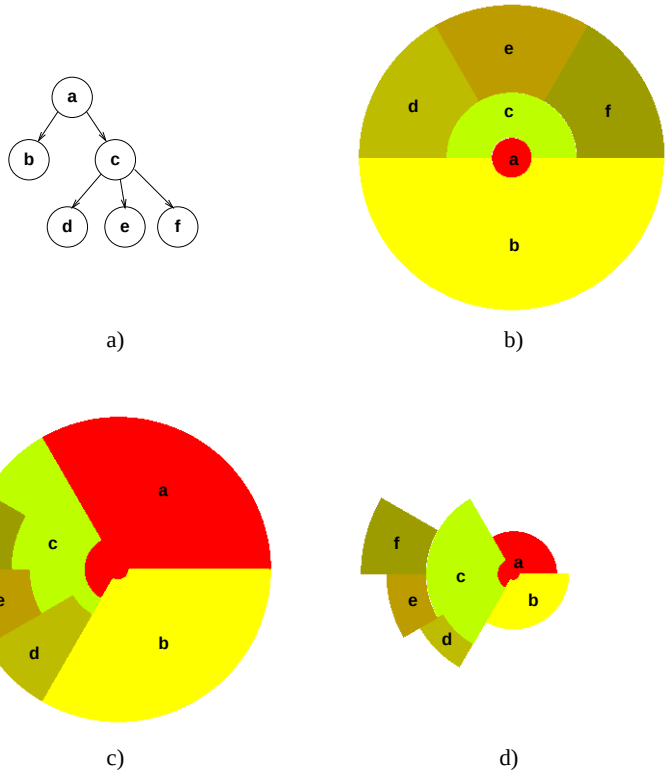


Figure 9: The tree ring representation shows timing as well as parentage. a) A traditional graph representation of a sequence of splits. b) A tree ring representation of the same sequence of splits under the timing scenario where node a splits into b and c and disappears. c) A tree ring representation of the same sequence of splits under an alternative timing scenario where the nodes continue to exist after splitting. d) A third scenario in which only node f survives until the end of the record.

to its descendents. Moving outward, the sectors representing nodes b and c each subtend 180 degrees, indicating that node a has two direct descendents. The sector representing node c extends to the radius representing the time at which it disappears by splitting into nodes d, e and f respectively.

Figure 9c shows an alternative scenario in which node a first splits off node b and later splits off node c. Later c splits off d, then e and then f. All six nodes exist until the end of the record. Figure 9d shows still another possible timing. In this third scenario node a first splits off b and then node c. Node a then dissipates followed by the dissipation of node b. Later node c splits off node d. Then node c splits into nodes e and f, disappearing in the split. Node d then dissipates followed by the dissipation of node e leaving a single surviving node.

The tree ring representation shows all descendents of a specified node. These descendents were either created by splitting from the designated node or one of its children. A node may also become a descendent of the specified node if one of the specified node's children merges into it. The depiction of merge operations in the tree ring representation has two forms. When the specified node undergoes a merge, a dotted ring appears at the radius corresponding to the time of the merge. If one of the children underwent a merge, a dotted arc appears across the node's sector at the radius corresponding to the time of the merge. Figure 10a presents the tree ring representation of the process shown in Figure 2a from the viewpoint of node a. Figure 10b shows a similarly timed example in which the child nodes dissipate rather than merge.

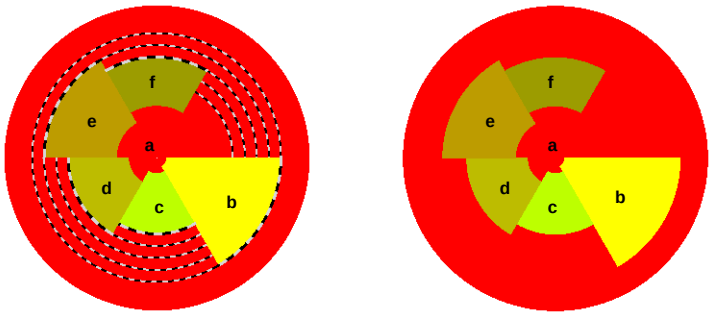


Figure 10: Merges are represented by striped arcs. a) Tree ring representation of the process shown in Figure 1. b) Same sequence of events except that nodes b through f dissipate rather than merge with a.

Radial distance represents time in the tree ring representation, and an arc is assigned to every descendent

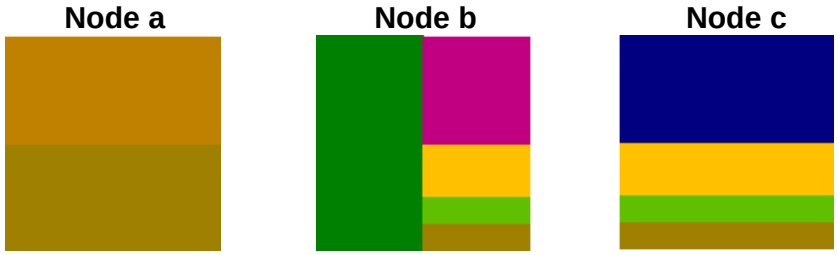
that is alive at the corresponding time. Figures 9 and 10 use a proportional scheme for assigning radial distance and arc length. The proportional scheme determines the time interval over which the node exists and constructs the complete descendent tree prior to display. Events corresponding to splitting, merging and dissipation are then linearly mapped to radii corresponding to their positions in the time interval. If a node disappears by splitting, the arc occupied by that node is subdivided among the children who split (e.g. node c in Figure 9b). If a node splits off a series of children but remains in existence, the arc is divided among it and its future children (e.g. node c in Figure 9c).

The proportional radius-arc mapping produces balanced tree rings, but the approach cannot be used for on-the-fly generation because the entire descendent tree must be traversed prior to display. An alternative mapping assigns half of the remaining arc for each split in which the node coexists with its children. While not maintaining balanced rings, this approach is easily adapted to on-the-fly generation. An estimate of the total time can be made prior to the start, and the rings can be redrawn if the time is exceeded. A logarithmic mapping of time can be more effective than linear time for large descendent trees, because it gives relatively smaller radial weight to events that happen at larger times. Since these times appear farther out on the tree, they are given relatively more area. The logarithmic mapping balances the assigned display area for events that happened near the beginning with those that appear later. For systems that are bursty — having long periods of quiescence followed by clusters of split and merge events, a time scale based on the order of the events rather than actual time may better emphasize the structure.

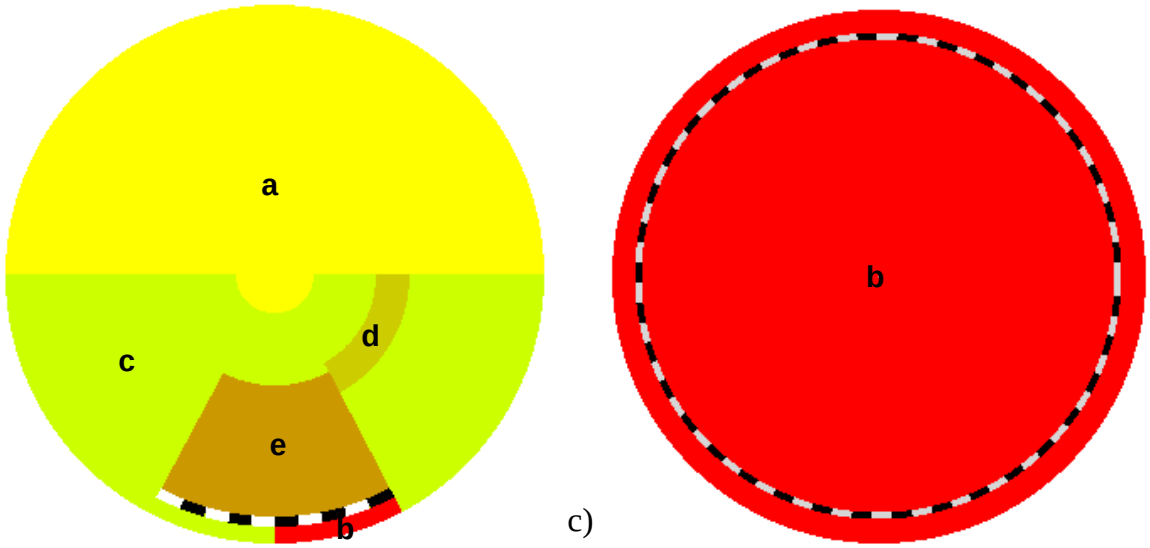
Figure 11 illustrates how different portrayals of split-merge events are related. The left time line in Figure 11a shows a sequence of events labeled as Box b) of Figure 5. Two nodes (a and b) are present in the initial record. Node a splits off node c. Later node c splits off node d and then e. Node d dissipates or dies. Finally node e merges into node b. Nodes a and b are seeds, and c, d and e are in a's seed set. The merge-seed abstraction of this sequence is shown by the right time line in Figure 11a. Notice that node c does not appear in the abstraction even though node c is alive at the end of the record.

Figure 11b shows the dynamic box representation of the sequence. At the end of the sequence, only nodes a, b and c exist. Each box shows the history of the corresponding node looking backwards on the time line. In contrast, the tree ring representation shows the sequence looking forward. Figure 11c shows the tree ring representation for the seeds a and b. Notice that from the viewpoint of node b, there are no

a)



b)



c)

Figure 11: A comparison of different representations of split-merge events for the sequence of Box b) in Figure 5. a) Time lines for the original sequence (left) and the abstracted sequence (right). b) The dynamic box representation of the leaves. c) The tree ring visualization for the seeds.

other nodes in its descendent tree. The time at which node e merges into b is marked by a striped arc in b's tree ring.

The dynamic box and tree rings can be used as auxiliary displays to enhance information about a particular focus area in the process. The user might specify a vertical strip in the time-line visualization as a focus area and extract a split-merge process. Each node that exists at the beginning of the interval could be associated with a tree-ring visualization that looks forward at its split-merge future. Similarly, each node that exists at the end of the interval could be associated with a dynamic box showing its split-merge past. Compare the dynamic box representation of node b in Figure 11b with the tree ring visualization of node b in Figure 11c. The box represents the arcs emanating from b looking backward through the record, while the tree ring represents the arcs emanating from b looking forward.

6 Discussion

Split-merge processes have a dynamic structure that is not captured by static graphs, and appropriate visualizations are somewhat application-dependent. This paper emphasized the development of a formal definition of split-merge processes and a corresponding visualization framework. The developments were explored using two real applications — one from a network protocol study and the other from an analysis of videotape of a combustion experiment. The framework provided the abstract data types needed to adequately represent a complex collection of split-merge structures, and these data structures were used directly in the visualization implementations developed in this paper. The definition allows on-the fly generation of split-merge information and permits concatenation and partition operations that support panning and zooming in visualization.

A method of abstracting structure by decomposing the process into seed sets was also developed. This abstraction allowed complex split-merge processes to be represented as a whole in a simpler, but structurally faithful manner. This abstraction can be used in visualization systems to provide an overall representation of the process. The viewer can choose to zoom in on interesting regions and see successively more detail.

Time lines provide an intuitive graphical representation of split-merge processes, but there are a number of subtle interpretation issues that arise based on layout and representation of time. The applications examined in this paper confirmed the well-known adage that viewing different visualizations in combination

is more effective than viewing isolated visualizations. Dynamic boxes and tree rings present history from the viewpoint of a single node. Dynamic boxes summarize previous split-merge history of a node and are effective for displaying histories on the order of tens of split-merge events. Tree rings summarize future history and are effective for displaying histories on the order of a hundred events.

Scalability in such problems is always an issue. The temporal constraints on split-merge processes force the number of edges to be of the same order as the number of vertices, so operations such as panning, zooming and abstraction can be performed very efficiently. The scalability from the perspective of visualization is how large a process can be laid out on the screen. In general, these techniques are expected to be effective for graphs on the order of 1000 nodes, but not 10000 nodes. The question of how large a process can be examined, depends to some extent on how many nodes exist simultaneously and how well the process can be abstracted. When the number of nodes is too large for convenient screen display, abstraction can be used to provide summary information.

Acknowledgments

We would like to thank Michael Gorman and Mohamed el-Hamdi for providing experimental videotape of the combustion experiment and Jo Ann Withers for extracting split-merge information from these tapes. Jo Ann also first proposed the dynamic box visualization as part of her Masters thesis. We would also like to thank Xiaoping Chen for her role in developing the software implementation of the broadcast protocols that were studied. This work was partially supported by grants NSF (CCR-9409082, CDA-9633299, and ACI-9721348) and ONR (N00014-97-0029). Some of the visualization tools used in this work were developed with support from grants NSF (DUE-9750953 and DUE-9752165).

References

[1] K. Birman, A. Schiper and P. Stephenson (1991) Lightweight causal and atomic group multicast. ACM Transactions on Computer Systems 9(3), 272–314.

[2] P. G. Danzig (1994) Flow control for limited buffer multicast. IEEE Trans. Software Engineering 20(1), 1–12.

[3] G. DiBattista, P. Eades, R. Tamassia and I. G. Tollis (1994) Algorithms for drawing graphs: an annotated bibliography. Computational Geometry 4, 235–282.

[4] M. Doar and I. Leslie (1993) How bad is naive multicast routing? IEEE INFOCOM '93, 82–89.

[5] S. E. Elmaghraby (1977) Activity Networks: Project Planning and Control by Network Models. John Wiley.

[6] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker (1988) Solving Problems on Concurrent Processors, Vol I: General Techniques and Regular Problems. Prentice Hall.

[7] M. Gorman, M. el-Hamdi and K. A. Robbins (1994) Four types of chaotic dynamics in cellular flames. Combustion Science and Technology 98, 79–93.

[8] B. Johnson and B. Shneiderman (1991) Tree-maps: A space-filling approach to the visualization of hierarchical information structures. Proc. of IEEE Visualization '91, 284–291.

[9] V. Kumar, A. Grama, A. Gupta and G. Karypis (1994) Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing.

[10] Kusnadi, J. D. Carothers, F. Chow and C. Beebe (1995) Single phase algorithm for hierarchical graph visualization. 1995 IEEE Intl Conf. on Systems, Man and Cybernetics, Intelligent Systems for the 21st Century, 2025–2028.

[11] J. Lamping and R. Rao (1996) The hyperbolic browser: a focus+context technique for visualizing large hierarchies. Journal of Visual Languages and Computing 7(1), 33–55.

[12] Y. K. Leung and M. D. Apperley (1994) A review and taxonomy of distortion-oriented presentation techniques. ACM Trans. on Computer-Human Interaction 1, 126–160.

[13] P. Linos, V. Rajlich and B. Korel (1991) Layout heuristics for graphical representations of programs. 1991 Intl. Conf. on Systems, Man and Cybernetics, 1127–1132.

[14] J. D. Mackinlay, G. G. Robertson, and S. K. Card (1991) The perspective wall: detail and context smoothly integrated. Proc. ACM SIGCHI Conference on Human Factors in Computing Systems, 173–179.

[15] S. Moen (1990) Dynamic drawing of trees. IEEE Software 7, 21–28.

[16] G. Parker, G. Franck and C. Ware (1998) Visualization of Large Nested Graphs in 3D: Navigation and Interaction. Journal of Visual Languages and Computing 9(3), 299–317.

[17] G. G. Robertson, J. D. Mackinlay and S. K. Card (1991) Cone trees: Animated 3d visualizations of hierarchical information. Proc. ACM SIGCHI Conference on Human Factors in Computing Systems, 189–194.

[18] K. A. Robbins and X. Chen (1997) Lazy broadcast protocols for distributed computing. UTSA Technical Report 97-1.

[19] M. Sarkar and M. H. Brown (1994) Graphical fisheye views. Communications of the ACM 37, 73–84.

[20] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubbs and M. Roseman (1996) Navigating hierarchically clustered networks through fisheye and full-zoom methods. ACM Transactions on Computer-Human Interaction 3(2), 162–188.

[21] P. D. Stots and Z. N. Cai (1988) Hierarchical graph models of concurrent CIM systems. 1988 IEEE Workshop on Languages for Automation: Symbiotic and Intelligent Robots, 100–105.

[22] K. Sugiyama, S. Tagawa and M. Toda (1981) Methods for visual understanding of hierarchical system structures. IEEE Trans. Systems, Man and Cybernetics, SMC-11(2), 109–125.

[23] K. Van Romaey, I. Bolsens and H. De Man (1992) Just in time scheduling. Computer Design: VLSI in Computers and Processors (ICCD '92), 295–300.

[24] J. A. Withers (1996) Tracking Splits and Merges in Large-scale Evolving Structures. Master's Thesis, University of Texas at San Antonio, May 1996.

[25] J. A. Withers and K. A. Robbins (1996) Tracking cell splits and merges. Proceedings IEEE Southwest Symposium on Image Analysis and Interpretation, 117–122.