Enhancing State-Space Tree Diagrams for Collaborative Problem Solving

Steven L. Tanimoto

University of Washington, Seattle WA 98195, USA
tanimoto@cs.washington.edu
http://www.cs.washington.edu/homes/tanimoto/

Abstract. State-space search methods in problem solving have often been illustrated using tree diagrams. We explore a set of issues related to coordination in collaborative problem solving and design, and we present a variety of interactive features for state-space search trees intended to facilitate such activity. Issues include how to show provenance of decisions, how to combine work and views produced separately, and how to represent work performed by computer agents. Some of the features have been implemented in a kit "TStar" and a design tool "PRIME Designer."

1 Introduction

1.1 Motivation

Problem solving and design processes tend to confront more and more complex challenges each year. A solution to a single problem often requires expertise from several domains. For example, Boston's "Big Dig" required expertise from civil engineers, geologists, urban planners, and politicians, among many others.

A group at the University of Washington is studying the use of the classical theory of problem solving as a methodology in support of collaborative design. One part of the approach involves the use of interactive computer displays to anchor the shared experience of a team. This paper describes research in progress on a set of issues and possible features to address them. The issues and features pertain to the use of interactive tree diagrams that show portions of a design space or solution space, and that support a team in managing their work.

1.2 State-Space Search Trees

Problem solving has been cast in the formal framework of "state space search" by early researchers in artificial intelligence, such as Newell and Simon (see Simon, 1969). A solver starts with an initial state, representing an absence of commitments to the particulars of a solution, and step by step, tries adding particular elements or modifying the current state, in an attempt to reach a "goal state". The goal state, or perhaps the sequence of steps taken to reach it, constitute a solution to the problem (Nilsson, 1971). The states visited in such a

G. Stapleton, J. Howse, and J. Lee (Eds.): Diagrams 2008, LNAI 5223, pp. 156–164, 2008.

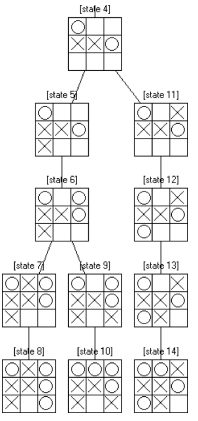

Fig. 1. A familiar kind of state-space search tree. This diagram corresponds to a partial search in the middle of a Tic-Tac-Toe game.

search can usually be plotted as nodes of a graph. Each move from one state to another, in turn, corresponds to an edge of the graph. If we decide that each state is described by the sequence of operators applied, starting at the initial state, then the graph has the structure of a tree, and it can be drawn as a tree. Such diagrams have been used to illustrate textbooks of artificial intelligence, often in conjunction with classical puzzles such as the Eight puzzle (Nilsson, 1971) or the game of Tic-Tac-Toe as shown in Fig. 1. The use of state-space search trees in interfaces was explored in an earlier paper (Tanimoto and Levialdi, 2006), and their use in the design of image-processing procedures was discussed in the context of face-image analysis (Cinque et al, 2007).

1.3 Issues

This paper is concerned with a set of issues that arise when one chooses to use state-space search tree diagrams as interactive representations of the progress of a design or problem-solving team. Some of these issues are the following.

Provenance. Each node represents a partial design or solution. Who contributed what to this design? From what other designs was it derived? Supports for querying and viewing provenance are needed.

Negotiating Shared Views. A designer's personal view of a design tree reflects her priorities and preferences. In a shared view, priorities and preferences may collide, leading to the need for compromises. Algorithms can offer certain compromises automatically, but shared views that are most satisfactory to a group may require give and take. Facilities to manage such negotiation are needed.

Visualization of Opportunity. The members of a team may understand their own components of a design or solution, but it's often difficult to understand

the potential interactions of the pieces. In some cases, separately designed pieces can be combined, and the diagram can indicate the compatibility of a piece for a particular role. Some opportunities result from the applicability of operators to newly computed states. Others result from the availability of resources in relation to defined tasks. Helping users see and evaluate opportunities is an important goal for collaborative design and problem-solving tools.

Work by Agents. A collaborative team traditionally consists of people. However, computer agents may be present and available to help out. Agents generally work very differently from people, and special affordances are required to define and display tasks for agents and the results of those tasks.

1.4 Previous Work

While much research has been done separately on problem solving, tree diagram layout, operations on trees, and interacting with trees, we mention in this section some work that bridges these topics.

Flexible Tree Layout. In order to display a large tree so that various parts of the tree can be distinguished, or so that a minimum or limited area is used, many methods have been proposed. A survey of tree display methods was presented by Hanrahan (2001). Popular methods include "hyperbolic trees" (Lamping et al, 1995), subdivision diagrams, and cluster diagrams. Usually, the goal is to display a large tree in such a way that individual nodes get their fair share of screen real estate.

Folder Tree Affordances. A family of interactive techniques for trees has been developed in the context of operating-system file hierarchies (Kobsa, 2004). The best-known example of a file system browser is the Microsoft Windows Explorer (not Internet Explorer). Interior nodes of the tree shown in Explorer correspond to folders in the file system, and the leaves correspond to individual files. The user can click on folder icons (situated at the nodes) to open and close the folders. When a folder that contains files is closed, it has a hotspot labeled with a plus (+) sign. Clicking on the plus sign opens the folder to reveal its contained files and subfolders. A more general set of affordances for hiding and revealing folders is described in a US patent (see Chu and Haynes, 2007). Here it is possible to have two nodes in a relationship where A is an ancestor of B, where A is closed, but B is visible and open. In such a view, all of A's descendants, except B and its descendants (and any other nodes explicitly opened) are hidden.

Interest-Sensitive Layout. There have been several developments that have combined flexible layout with interactivity. A notable example is "Degree of Interest Trees" (Card and Nation, 2002). In such a tree, the user can indicate interest or disinterest in particular nodes of the tree by clicking on them, and the layout of the tree immediately adapts to the new distribution of interest, using a continuous animation to move from the current layout to the new layout.

2 Application to Collaborative Design

2.1 Rationale

We use the classical theory of problem solving for two reasons: (1) provide a common language for the problem solving process to design teams whose members represent different disciplines, and (2) help humans interact, via the computer interface, not only with each other, but with computational agents that perform design or problem-solving services. To put our use of this theory into context, we now explain the general category of atomic actions that we call "design acts."

2.2 Design Acts

We assume that the entire process of design can be broken down into small steps that we call *design acts*. There are several types of design acts, including communication acts, design steps, evaluation acts, and administrative acts. Communication acts include writing and reading messages associated with the project. Design steps are primarily applications of operators to existing nodes to produce new nodes in a search tree. in order to apply an operator, may also be considered a design step. An evaluation act is either a judgment or an application of an evaluation measure to a node or set of nodes. A judgment is a human-created quantitative or qualitative estimation of the value of a node on some scale with regard to some particular characteristic. An evaluation measure is a mathematical function that can be applied to a node to return a value (usually numeric). Such a value may correspond to the degree to which a partial design meets a particular criterion. For example, one evaluation measure for designs of houses is the number of rooms in the house. Another is the square footage of area in the designed house so far. A judgment might correspond to an aesthetic evaluation – how pleasing is this floor layout to the eye of designer Frank? Administrative acts include actions such as the definition of tasks and subgoals for the design team, commitments of effort to tasks, and adjustments to views of the progress made so far.

2.3 Roles of the Diagram

By using tree diagrams, it is possible to provide computer assistance, at some level, for each type of design act mentioned above. Many communication acts relate to parts of a tree. Messages can relate to parts of a tree in two ways: (1) by naming a labeled node or by describe the path to it from the root, and (2) by being embedded in the node as an annotation. Messages can also refer to nodes via descriptions or expressions in a node-specification language. Such a language is a kind of query language with features for identifying nodes not only via their properties and annotations, but via their "kinship" relations.

Design steps are supported via interactive tree-building functionality. The selection of nodes and application of operators is done with clicks and menus. (An example of a design tree built with a tool called PRIME Designer is shown

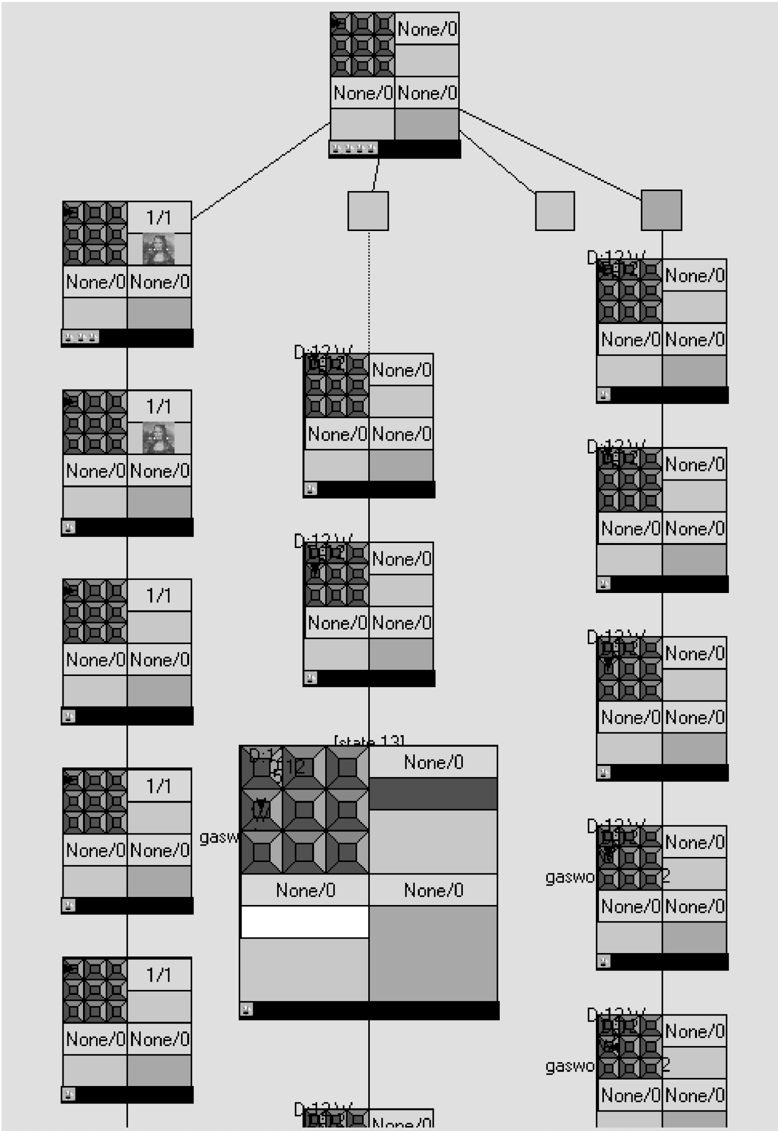

Fig. 2. Design tree showing multiple paths developed by separate members of a team. Also shown are "minified" nodes, an ellipsis under the first minified node (represented with the dotted line), and an enlarged node. The four quadrants in each node correspond to different roles on the design team. This view is an "all-roles" view, and most nodes are empty in three out of the four roles; the "None/0" label means that no components have been added in the role, and none is selected.

in Fig. 2. The tool supports a four-person team – musical composer, architect, image-puzzle designer, and game-logic programmer – in creating a multimedia game. It has been used in focus-group trials.)

In addition to helping with specific design acts, the diagrams provide an easily browsable record of the history of the design process. They also provide contextual information for particular tasks. Thus a state-space search tree provides an organizing framework to a team for pursuit of solutions.

3 Layout, Visibility, and View Control

State-space search trees have the potential to grow very large. In order to make large trees manageable on an interactive display, the following techniques are commonly used: scrolling, overall scaling, and opening/closing (revealing/hiding) of subtrees. We have developed variations on each of these techniques that can provide additional user control in the case of search trees.

3.1 Scaling

Overall scaling, equivalent to zooming, is an essential viewing technique for large trees. A challenge for designers of visualizations is providing controls so that users can perform *differential scaling*, so that different parts of the tree can have different scale factors. A form of differential scaling can be found in dynamic tree layout methods such as hyperbolic trees (Lamping et al, 1995) and "degree-of-interest" trees (Card and Nation, 2002). These methods are convenient for giving prominence to regions of a tree, a region being a subtree or the nodes in a graph neighborhood of a given node.

Another approach to differential scaling allows the user to associate, with any node or nodes s/he deems special, a particular scale factor for that node. That factor is then used as a relative factor, and zooming of the overall view maintains the differential. Our TStar software supports this technique.

3.2 Hidden Nodes

While scrolling and scaling are often considered to be geometric operations, they are also means of information hiding, since scrolling is a way of showing some and hiding other information in a diagram, and scaling makes some details visible or invisible and shows more or less of the area of the entire diagram.

Now we describe two additional forms of information hiding, one of which we call "hidden nodes" and the other "ellipsis." They are closely related. They differ in three respects: how affected portions of the diagram are rendered, what kinds of sets of nodes can be hidden, and how the user interacts with them. We make these distinctions in order to provide a rich set of view controls to users, and to suggest additional possibilities for dealing with the complex but important issue of working with state-space tree diagrams.

We define a hidden node to be an invisible, but fully computed node object, that is neither rendered on the screen nor has any effect on the rendering of the rest of the tree. From the standpoint of screen appearance, a hidden node may just as well be a node that was never created or that has been deleted. If it has

any children, they, too, must be hidden. In our TStar system, the user can hide a subtree in order to create a simpler view, or unhide a subtree to (re)include it in the diagram. Any overhead that would be associated with deleting or recreating corresponding states is avoided in the hiding and unhiding operations.

In order for the user to know that a hidden node exists in a tree, a small indication is available, on demand, in the form of a highlight on nodes having hidden children. Hiding the root is not permitted in TStar.

3.3 Ellipsis

A node *in ellipsis* is a tree node that is not displayed but is represented by an ellipsis indication. One or more nodes, in an ancestor chain, can be represented by a single ellipsis indication. An *ellipsis* is a collection of all the nodes in ellipsis that are represented by the same ellipsis indication.

An ellipsis therefore represents a set of nodes in a tree whose existence is important enough to indicate, but whose details and whose number are sufficiently unimportant that they are hidden. For example, the nodes in an ellipsis may have an important descendant, displayed in detail, but not have any details of their own worth showing. The technique of ellipsis gives us a means to show nodes that have been explored but which we show in a very abbreviated form. Ellipses not only hide node details and reduce clutter (see, e.g., Taylor et al, 2006) but they save space by reducing the amount of space allocated to the nodes in ellipsis.

While we have just defined an ellipsis as a static set, in practice the user may work with dynamic ellipses. A basic dynamic ellipsis is a changing set of nodes in ellipsis, and a user-controlled dynamic ellipsis is a basic dynamic ellipsis with a boolean variable "currently-in-ellipsis" that can be set true or false by the user to adjust the visibility of the nodes without losing the grouping.

3.4 View Merging

When two members of team merge their trees via basic merging, very few conflicts can arise. The trees share only the root, and root can neither be put into ellipsis nor hidden, at least in our example system. Conflicts are another story when the trees are merged according to node equivalence based on operator sequences. Although a node N_a in tree A is considered path-equivalent to a node N_b in tree B if they are both derived from their (assumed equivalent) roots via the same operator sequence, they have have lots of attributes that are by no means equivalent. They may share different authorship, different annotations, different memberships in ellipses, they may be hidden or not, and they may have different scale factors associated with them.

Attributes of nodes such as authorship, creating timestamps, and descriptive annotations can usually be "unioned" – as entries on lists, the lists can be concatenated to preserve everything. View properties such as node scale factors can also be unioned (kept along for dynamically changing the view), but an actual

scale factor for the current view must also be determined. Three different methods are suggested, to be selected by the user at the time view merging is invoked, for different purposes. These methods are, briefly, "min," "max," and "diff." The min method is useful when each view represents work done to reduce scale factors on nodes judged to be unimportant, but each tree represents an incomplete job. The combined view will represent the totality of this work by the two team members. The max method is useful in the complementary situation, where each member has spent some effort to identify particularly important parts of the tree and has put in higher-than-normal scale factors for those nodes. The diff method is a means to highlight those parts of a combined tree in which the ratings of two team members (as expressed by node-specific scale factors) differ notably. Such a method takes the absolute value of the difference of the scale factors and maps it monotonically into a standard range of scale factors (0.1 to 2.0).

3.5 Representing the Work of Agents

Agents can explore large numbers of states, but most states tend to be relatively uninteresting. This calls for display methods that succinctly summarize searches of large subtrees. Agents also blur the boundary between explored states and unexplored states. If an agent has explored the state, has it really been explored? It depends on *how* the agent explores the state as well as the extent to which the user understands what the agent has "seen." One degree of exploration has been reached when a set of states is realized – the states have been computed. But that may be of absolutely no consequence to a user if there is no "end product" of the realization – no additional nodes displayed on the screen, no reports of solutions found or promising nodes reached.

This suggests a need for new ways of rendering subtrees to show different degrees or forms of exploration and results of exploration. If evaluation functions are computed on nodes, color coding according to value is a natural approach. Subtrees currently being analyzed may be shown with flashing or other animation to indicate the locus of activity.

4 Collaboration Operations on Trees

Several topological operations on trees are important in collaborative design as we have structured it: tree merging, path extraction, path merging, path simplification, and path reorganization to enable merging. These operations permit the separate work of individuals and subteams to be combined. Our PRIME Designer system currently the supports the first three of these five operations. Tree merging takes two trees and produces a new one whose root represents both input roots, and having copies of all the other nodes. Path extraction is the separation of a path in a tree as a separate tree. Path merging is the concatenation of two paths to form one long one.

The path merging operation is the most important one in combining the work of subteams, and it is the most problematical in terms of coordination. But it

offers interesting challenges for visualization of opportunities for combinations. Two paths A and B cannot be merged unless they are compatible: every operator application used to create a move in B must still be legal when the last state of A is used as the first state of B. The display of compatibility works as follows: after a path A has been selected, all nodes not on path A that begin paths that could be merged with A are highlighted, and the highlighting color gets brighter as the length the potential path is longer. In our future work, we intend to experiment with such compatibility displays, in combination with automatic evaluation of the merit of the potential merged paths.

Acknowledgments

Thanks to Brian Johnson, Richard Karpen, Stefano Levialdi, Tyler Robison, and Linda Shapiro for comments on parts of the software described here. Partial support under NSF Grant 0613550 is gratefully acknowledged.

References

1. Card, S., Nation, D.: Degree-of-interest trees: A component of an attention-reactive user interface. In: Proc. AVI, Trento, Italy. ACM Press, New York (2002)
2. Chu, H., Haynes, T. R.: Methods, Systems and Computer Program Products for Controlling Tree Diagram Graphical User Interfaces and/or For Partially Collapsing Tree Diagrams. US Patent 20070198930
3. Cinque, L., Sellers Canizares, S., Tanimoto, S.L.: Application of a transparent interface methodology to image processing. J. Vis. Lang. Comput. 18(5), 504–512 (2007)
4. Hanrahan, P.: To Draw A Tree. In: Presented at the IEEE Symposium on Information Visualization (2001), `http://graphics.stanford.edu/~hanrahan/talks/todrawatree/`
5. Kobsa, A.: User experiments with tree visualization systems. In: Proceedings of the IEEE Symposium on Information Visualization 2004, pp. 9–16 (2004)
6. Lamping, J., Rao, R., Pirolli, P.: A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In: Proc. ACM Conf. Human Factors in Computing Systems, pp. 401–408 (1995)
7. Tanimoto, S.L., Levialdi, S.: A transparent interface to state-space search programs. In: Kraemer, E., Burnett, M.M., Diehl, S. (eds.) Proc. of the ACM 2006 Symposium on Software Visualization (SOFTVIS 2006), Brighton, UK, September 4-5, 2006, pp. 151–152 (2006)
8. Nilsson, N.: Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York (1971)
9. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1969)
10. Taylor, J., Fish, A., Howse, J., John, C.: Exploring the Notion of Clutter in Euler Diagrams. In: Barker-Plummer, D., Cox, R., Swoboda, N. (eds.) Diagrams 2006. LNCS (LNAI), vol. 4045, pp. 267–282. Springer, Heidelberg (2006)