Triangle-Table Trees

Nils J. Nilsson

August, 1990

[The following document is part of the technical material included in proposals to the Army Research Office, the Air Force Office of Scientific Research, and the National Aeronautics and Space Administration submitted in August, 1990.]

A. Tree Plans

Triangle-table trees are a generalization of the triangle tables of Fikes, Hart, and Nilsson (Fikes 1972). We describe them here as a particular way of representing plan structures that we call *tree plans*. Tree plans, in turn, are defined in terms of the state-space graph characterizing a planning problem.

By way of an example, we show in Fig. 1 the state-space graph for the familiar problem of moving blocks. The nodes of the graph are labeled by pictures representing the possible block configurations. We have labeled the arcs by the actions that change one configuration into another. To simplify the graph we have represented a pair of arcs by a bi-directional edge; each edge is labeled by the two actions corresponding to its two arcs. The action `Move(x,y)` means move the block named `x` to the top of the block (or table) named `y`. The symbol `T` stands for table.

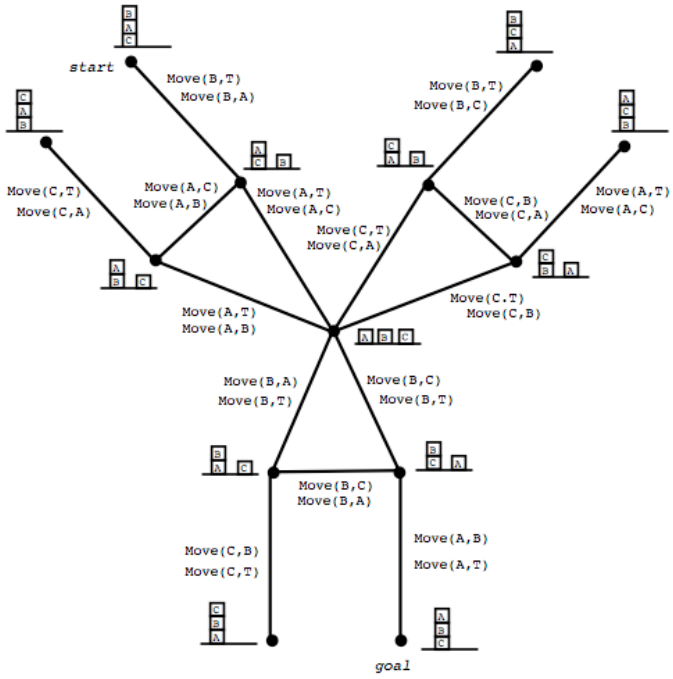

Figure 1. State-Space Graph for Blocks Problem

Suppose we desire to re-arrange the blocks in the state marked *start* to the configuration shown in the state marked *goal*. We can compute the plan of actions to take by searching the graph of Fig. 1 for a path from the start node to the goal node. One path from start to goal corresponds to the sequence of actions {Move(B,T), Move(A,T), Move(B,C), Move(A,B)}. This path is indicated in Fig. 2 by the heavy lines.

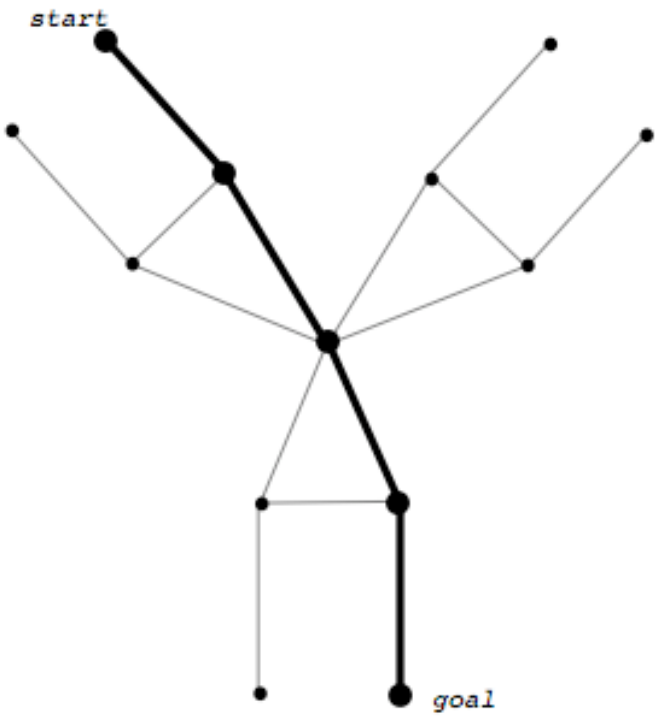

Figure 2. A Path from the Start Node to the Goal Node

This particular method of solving this block-stacking problem is just one of many that we might have described. For example, we might have described each state by a set of predicate calculus expressions and then represented the actions by computations that transformed one set of expressions into another—as in STRIPS (Fikes 1971). We shall not be concerned here with the specific methods for representing and solving planning problems in terms of graphs and graph-searching. Suffice it to say that many such methods can be expressed in terms of a graph or state-space model. [As an alternative to searching a space of states, some planning systems (Sacerdoti 1977, Tate 1977, Wilkins1988) search a space of partially developed plans; the arcs in the graphs used by such systems then correspond to various plan-altering and plan-completing steps that can be taken by the planner.]

The path from the start node to the goal node, expressed in terms of the sequence of actions that the path represents, is a *plan* or program which can be executed to solve the problem. If the world were such that there were no other source of change except the agent itself, and if the effects of the agent's actions were accurately described, then we could be sure that the execution of the plan would achieve the desired effect. In such a case, there would be no need for the agent to verify the effects of each executed action by sensing the world; it could blindly execute the plan and assume that the effects are always as predicted. Of course, for robots or other agents operating in realistic worlds, these assumptions do not hold.

When an agent executes an action in a dynamic world, it cannot assume that the action will always have its nominal effect. For example, in our block-stacking problem some other process might topple the blocks, or the robot might accidentally drop a block. In the worst case, after executing an action in a plan, the world might be in *any* of all the possible states, and the agent will have to determine through its sensors (as best it can) which of these it is in before it can decide on the appropriate action to execute next. [When the world is not so grossly unpredictable, the agent might be able to narrow the set of states the world might be in. In the control theory work on discrete event systems (Ramadge 1989), automata theory is used to express the constraints on the set of possible world states.]

To be maximally teleo-reactive, an agent would either have to be prepared to plan anew at run time after it executes each action, or it would have to pre-plan a path to the goal from several (possibly all) states prior to run time. Paths from several states to the goal can be stored as a tree rooted at the goal node. When the tree contains the paths from all possible states to the goal, it is called a *spanning tree* of the state-space graph. A spanning tree for the graph of our blocks-world problem is shown in Fig. 3.

The tree that contains paths from several nodes to a goal node can be regarded as a plan for the agent just as can the list of actions along a single path to the goal. To execute a *tree plan*, the agent first determines which state in the tree the world is in (by performing appropriate tests) and then executes the action labeling the arc exiting that state. Regarded as programs, tree plans contain implicit run-time conditionals in the tests that must be performed to determine which state the agent is in. We are assuming here that generating a new plan at run time will typically exceed the time bound allowed for action, and thus pre-computing some kind of tree plan (perhaps a partial one) suggests itself as an interesting way of meeting the real-time requirement.

If the tree is a spanning tree of the state-space graph, then the tree plan is equivalent to what Schoppers calls a *universal plan* (Schoppers 1987). Given that an agent's actions typically achieve their expected effects, the execution of a spanning-tree plan will ultimately get the agent to its goal even in environments that are otherwise unpredictable. Often it is impractical to compute and store a spanning-tree plan, but it may be worthwhile to compute and store smaller tree plans and to use them as programs for controlling an agent. It should be pointed out that if the search process that computes a path from a current state to the goal proceeds backward from the goal, then the search process itself inevitably computes a tree of paths; thus, there is no extra expense in computing the tree plan that corresponds to the search tree. Of course the most useful tree plan for an agent would be one that contains paths to the goal from environmental states likely to be visited given the dynamics of the environment.

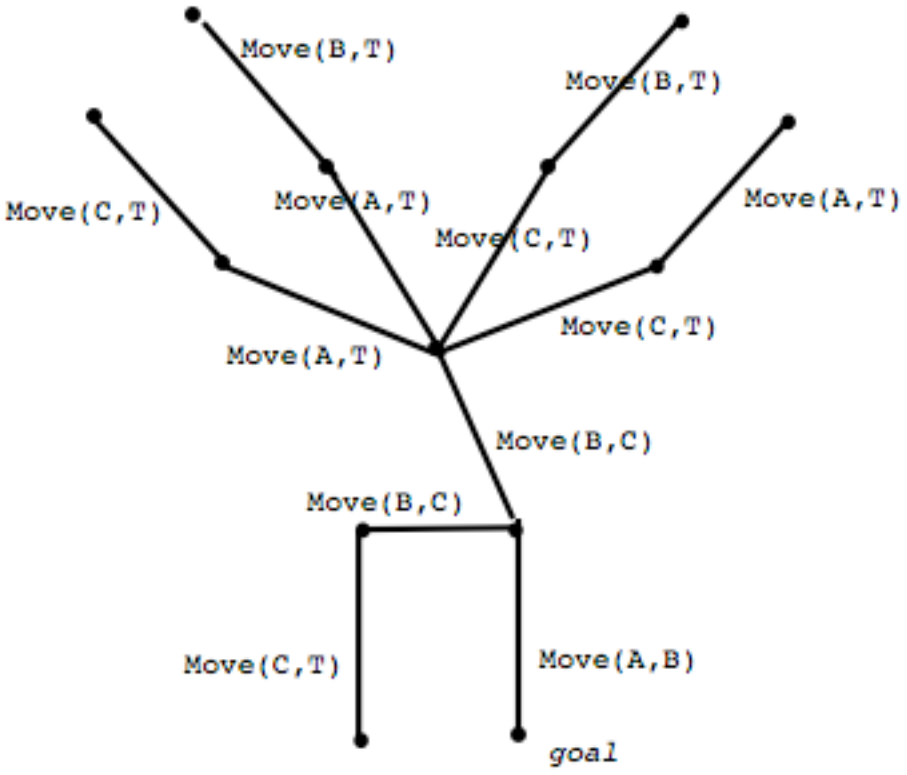

Figure 3. A Spanning Tree

B. Triangle-Tables and Triangle-Table Trees

An interesting special case of a tree plan has been captured in a formalism called *triangle tables*. This formalism was used in connection with the STRIPS planning system and the SRI robot *Shakey* (Fikes 1972, Nilsson 1984). We give a very brief description of triangle tables here before presenting the generalization of them that we want to study. A triangle table, of rank *N*, is an *N* x *N* triangular array of cells; each cell may contain a collection of ground literals. The rows of the array are numbered from the top, starting with row *1*; the columns of the array are numbered from the left starting with column *0*. Each column except the *0*-th is headed by the name of an action.

A triangle table is intended to be a representation of the preconditions and effects of a sequence of actions beginning in a world state that satisfies a certain condition. The conjunction of all the formulas in the row to the left of an action is the precondition for executing that action. More precisely, if this conjunction follows from the formula describing a world state, then the action can be executed in that world state. The formulas in the column immediately beneath an action are those formulas that are on the add list of that action and are also either preconditions of actions heading higher-numbered columns or effects in the last row of the table. The formulas in the column below an action are distributed among the column cells in such a way that those that are preconditions of other actions are in a cell to the left of that action. It is possible that a formula may be repeated in a column if it is part of the precondition of more than one action.

Literals in the last row of the table are those that are part of the goal formula. These are the final effects of the table. Preconditions of actions that are not realized by any actions in the table must

hold before the table is executed; these preconditions are represented by formulas in the *0*-th column of the table. These are the initial preconditions of the table. To be listed in the table, any literal needed by a particular action and provided by a previous action must survive the intervening actions. A literal in the last row of the table survives all actions subsequent to the one that achieved it.

The conjunction of the literals in the rectangular subarray consisting of the bottom $N - (n - 1)$ rows of the leftmost n columns is called the n-th *kernel* of the table. Each kernel can be thought of as the precondition that must hold for a certain sequence of actions to be executable and for the effects of the table to be achieved. Let a_i denote the action heading column i; then the i-th kernel is a precondition for the action sequence $a_i, \ldots, a_N$ to be executable and to achieve the effects that appear in the last row of the table. If the N-th kernel is true in a world state, then without executing any actions, the table's effects also follow from that state. The first kernel, which is the precondition of the entire table, must be true in a world state in order for the entire sequence of actions to be executable in that state and to achieve the table's effects.

Because the i-th kernel is precisely the same formula as that labeling the i-th node in the solution path to the goal, a triangle table is just another way of representing the plan that consists of just the single path from start to goal. It contains all the information needed to decide which action along the solution path to execute given that the agent can determine which of the states along that path it is in.

We can extend a triangle-table, representing a single path plan, to a triangle-table tree representing an entire tree plan. Consider, for example, the small tree plan shown in Fig. 4. A triangle-table tree version of this tree plan is the three-dimensional structure shown in Fig. 5.

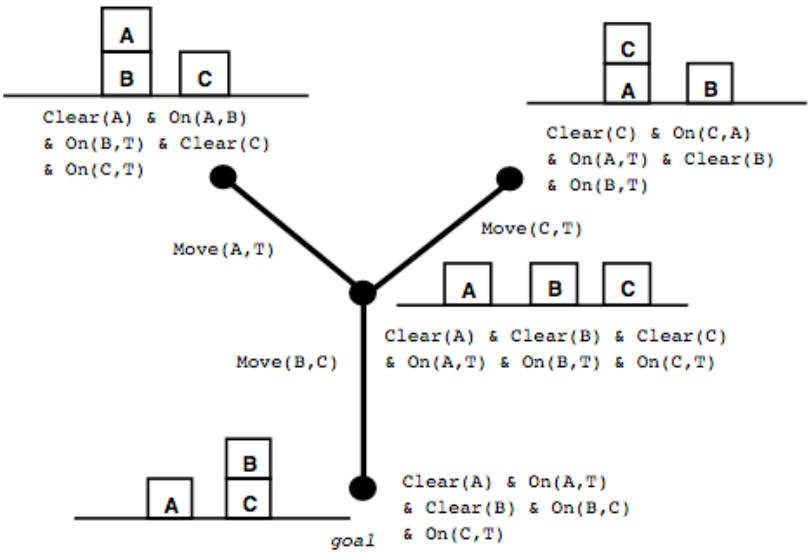

Figure 4. A Tree Plan

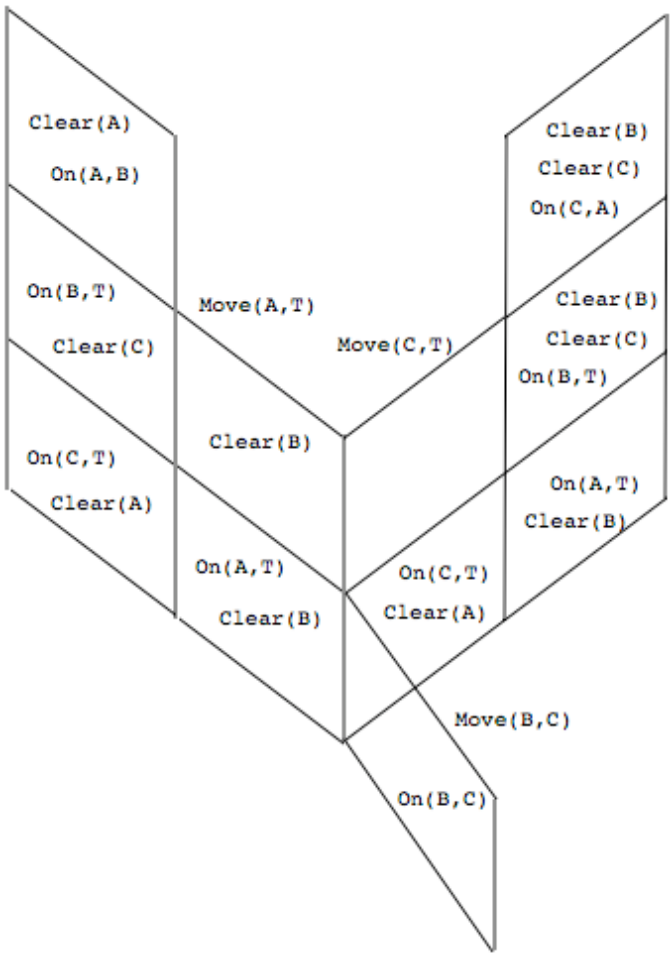

Figure 5. A Triangle-Table Tree

One can imagine a triangle-table tree structure of a similar sort representing the spanning-tree plan of Fig 3.

C. A Scanning Algorithm for Triangle-Table Trees

When the conditions labeling the nodes of a tree plan are represented in triangle-table-tree form, if there is a node satisfying the current state, then one of the triangle-table paths in the tree will have an active kernel. We propose here an efficient algorithm for finding such a kernel without having to check any cells twice. We first define some terminology. If our search procedure has already concluded that a node in the tree is not the current state, we shall call such a node *failed*. We shall say that a kernel (in one of the tables in the tree) is *failed* if its corresponding node is failed. Our procedure will search the kernels of the tables backward from the goal and will mark those failed that are not satisfied by the current state. Each table will have a *boundary* between two of its columns which separates the failed kernels from those that have not yet been tested. Our procedure is as follows:

0. Set the boundaries of all tables just to the right of their right-most columns. (Initially, there are no failed kernels.)

1. Select (arbitrarily) one of the tables from that set of tables whose boundaries are closest to their right-most columns.

2. Perform the standard triangle-table scan algorithm (Fikes, 1972) on this triangle table. Either the scan succeeds and finds a satisfied kernel (we are assuming there is at most one in the tree), or it fails to find any satisfied kernels.

If it succeeds, we execute the corresponding action. If it fails:

a) mark all nodes on this path failed, and also
mark failed any kernels in the other triangle tables that
correspond to these failed nodes.

b) reset the boundaries on the other tables so that they
separate failed kernels from those not yet failed.

c) go to 1.

Although this algorithm seems to be the appropriate generalization of the standard triangle-table scan algorithm, it does not use information that may be available about which node in the tree plan is most likely to be the current state. After executing an action in a tree plan, we would expect (all other things being equal) to be in the state that this action is supposed to put us in. The triangle-table tree scanning algorithm does not bias the search toward checking this state first. In the proposed research we will investigate other scanning algorithms that take account of likely state transitions using problem-specific information.

Tree plans and triangle-table trees appear to be a promising formalism for bounded-time action computation [in agent architectures]. An important advantage of this formalism is the ease with which it can be coupled to a planning subsystem. The planner can make incremental changes to the action computation subsystem by adding subtrees to the plan-tree structure.

D. References

Fikes 1971
Fikes, R. and Nilsson, N., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, **2**(3-4): pp. 189-208, 1971.

Fikes 1972
Fikes, R., Hart, P., and Nilsson, N., "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, **3**, 1972.

Nilsson 1984
Nilsson, N. (ed.), *Shakey the Robot*, SRI Technical Note 323, April 1984, Menlo Park, CA 94025, 1984.

Ramadge 1989
Ramadge, P. J. G., and Wonham, W. M., "The Control of Discrete Event Systems," *Proceedings of the*

IEEE, vol. 77, no. 1, pp. 81-98, January 1989.

Sacerdoti 1977
Sacerdoti, E., *A Structure for Plans and Behavior*, American Elsevier, 1977.

Schoppers 1987
Schoppers, M. J., "Universal Plans for Reactive Robots in Unpredictable Domains," *Proceedings of IJCAI-87*, 1987.

Tate 1977
Tate, A., "Generating Project Networks," *Proceedings IJCAI-77*, pp. 888-893, Cambridge, MA, 1977.

Wilkins 1988
Wilkins, D., *Practical Planning*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.