Hierarchical Plan Merging with Application to Process Planning

J. Britanik* and M. Marefat*
Department of Electrical and Computer Engineering,
University of Arizona, Tucson, Arizona 85721

Abstract

We have developed a domain-independent systematic methodology for plan merging at the various levels of plan abstraction. This method manifests itself in the hierarchical plan graph where each level contains a complete, partially merged plan. The principle advantage of this approach is that, once external interactions between nodes on a given level have been established, the continued merging of the plan fragments in one node can take place independently of plan fragments in other nodes on that level. This provides a decomposition or divide-and-conquer approach to plan merging. Another advantage to this decomposition approach is that replanning effort is minimized in the presence of the selection of alternative actions at some level of the hierarchical plan graph. Only those plan fragments which are in the same branch as the alternative selection need be considered for replannmg. Also, an algorithm ts proposed which takes a bilateral approach to breaking cyclic dependencies between nodes in the hierarchical plan graph. We demonstrate the utility of this hierarchical approach to plan merging through examples in the process planning domain.

1   Introduction

A common approach to planning is to make the linear assumption by decomposing the problem into subproblems and planning for each subproblem independently. The resulting subplans are combined into the final plan via plan merging techniques [Hayes, 1989; Karinthi et a/.,1992]. Plan merging consists of unifying separately generated plans into one global plan while obeying the constraints due to interactions within and between the individual plans. Merging is typically done in a complex manner such that all actions of all the subplans are considered simultaneously

* email: britanik@ecc.arizona.edu
* email: marefat@ece.arizona.edu

in their lowest-level representation [Foulser *ei* a/.,1992; Karinthi *et ai,* 1992].

Our approach involves a hierarchical plan graph which we use to develop a domain-independent systematic methodology for plan merging at different levels of plan abstraction. The hierarchical plan graph is broken into levels, where each level contains a partially merged plan called a subplan sequencing graph. This approach decomposes the plan merging problem in that, once external interactions between nodes on a given level have been established, the continued merging of the plan fragments in one node can take place independently of plan fragments in other nodes on that level. In the presence of the selection of alternative actions at some level of the hierarchical plan graph, this decomposition approach minimizes replanning effort. Only those plan fragments which are in the same branch as the alternative selection need be considered for replanning. We also develop a method for breaking cyclic dependencies between nodes in the subplan sequencing graph.

We demonstrate the utility of this hierarchical approach to plan merging through examples in the process planning domain. The planning system consists of two major components: 1) a case-based planner which generates feature subplans [Britanik and Marefat, 1995], and 2) a plan merging component which hierarchically merges the feature subplans into a global plan. We will not discuss the case-based planner here. It is sufficient to note that the case-based planner generates process plans for each feature in the part, along with a list of interactions between the individual feature plans and plan fragments. The hierarchical plan merging component is the subject of the remainder of this paper.

1.1   Definitions

Before discussing hierarchical plan merging (HPM), we need to discuss what a plan is and define our notion of plan merging.

A hierarchical plan graph (HPG) is a directed graph of plans such that each level represents a plan at some level of abstraction. Figure 1 shows a generic hierarchical plan graph. Each node at a given level of the HPG represents a subplan of the plan at that level. The root node represents the initial set of individual plans. The leaf nodes represent the fully merged plan. A directed arc is drawn from node A' on level *i* to node *Y* on level
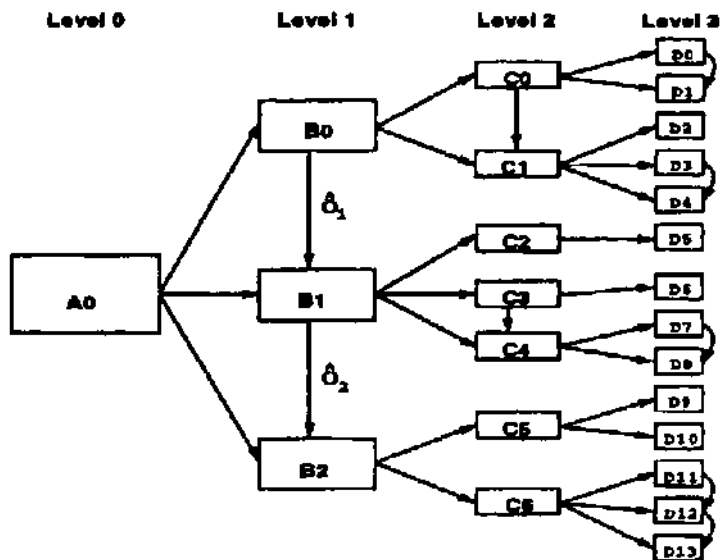
Figure 1: Generic hierarchical plan graph.

$i + 1$ if node Y contains a component from node X. A directed arc is drawn from node M to node N on the same level if there is a constraint between a component of node M and a component of node N that can be resolved by sequencing node M before node N. Nodes at subsequent levels of the HPG must obey the orderings introduced at the previous levels. The key property of the hierarchical plan graph representation is that the merging of the components of a node takes place in a manner independent of the other nodes on the same level.

At any level of abstraction, we view a plan, P, as a set of actions, A, along with a set of orderings, O, which sequence the actions in A. Thus $P = \{A, O\}$ and $A = \{a_1, a_2, ..., a_n\}$ where a_i are actions (possibly abstract actions), and $O = \{o_1, o_2, ..., o_m\}$ such that $o_i = (a_j \prec a_k)$ which represents the fact that action a_j must precede action a_k in the plan.

In our approach, the goal of plan merging is to combine the separate subplans into a global plan while minimizing the cost of the global plan. We reduce cost during the merging operation by eliminating redundant actions. Let M denote the set of actions in all plans that can be merged into one action a', and let $P_i = \{A_i, O_i\} = \{a_{i1}, a_{i2}, ..., a_{in}; o_{i1}, o_{i2}, ..., o_{im}\}$, then $P' = \{A', O'\}$ such that $A' = \left[\bigcup_{i=1}^{n} A_i\right] - \{M\} + a'$, and $O' \subseteq \left[\bigcup_{j=1}^{m} O_i\right] \bigcup \hat{O}$

where $\hat{O}$ are called *external orderings* since they represent the orderings between actions in different subplans. Likewise, the O_i are called *internal orderings* since they represent the orderings between actions in the same subplan at a given level.

1.2 Guide to the paper

To emphasize that our methodology is domain-independent, we first present the general theory in section 2. Then we show how this theory can be applied through a detailed process planning example in section 4. Section 3 briefly discusses how our approach can yield savings in replanning cost. Section 5 reviews directly related work, and section 6 concludes the paper.

2 High-Level Merging Algorithm

At each merging level of the hierarchical plan graph, the same high-level merging algorithm is used to generate the next level in the graph. The algorithm involves two primary steps: 1) building a minimal subplan sequencing graph, and 2) removing any cycles from the subplan sequencing graph.

2.1 Generating a minimal subplan sequencing graph

A subplan sequencing graph is a directed graph, $G = (N, E)$, where each node, n, $\in N$ is a subplan which contains the result of merging two or more actions from the preceding node on the previous level. A directed edge e, $\in E$ is drawn from node n_j to node n_k if there is an external ordering o, such that some component in T_j must be completed before some component in n^*. An example of a subplan sequencing graph is shown in level1 of figure 1. J30, *B1,* and *B2* are subplans which are sequenced by the orderings O_i = *(ag_0 -< OB1)* and O_2 = *(a_{B1} -< 32)* where a^i is an action in node *Bi.*

The generation of the nodes in the subplan sequencing graph is dependent on the domain in two ways. First, domain-specific information is used to determine the actions to be merged at the graph level of interest. Second, the criteria to determine whether two actions are mergeable are also domain specific. Assuming we have such information, we can use the following 3-step algorithm for generating a near-minimal number of nodes in the subplan sequencing graph: 1) generate a set of maximal nodes that cover all possible mergings of actions from the previous level in the HPG, 2) produce a near minimal covering of the actions from the previous level, 3) generate edges which correspond to external orderings between nodes.

To generate the set of maximal nodes, first a new node is created for each action type or mergeability type to be considered at this level. Next, components or plan fragments from the previous level are placed in each node, provided they are mergeable with that node's action type.

We now have a set of nodes, each of which contains plan fragments that can be merged together. A greedy approach is used to find the near-minimal covering of plan fragments in the maximal nodes[1]. First the largest node (ie: the node which contains the largest number of plan fragments) is chosen. Each plan fragment contained in the largest node is removed from all other nodes. Then the next largest node is chosen, and so on until there are no non-empty nodes remaining unselected (until a covering is achieved by all of the selected nodes). Figure 2 shows the algorithm to generate a minimal covering from the set of maximal nodes using this greedy approach.

We can improve the above greedy algorithm by checking for the case where all of the plan fragments in a node are mergeable with other nodes in the minimal covering. We take advantage of the following lemma.

[1] In [Britanik,1994], we compare this approach with two optimal approaches. This greedy approach was found to be efficient and near-optimal.

Greedy-FindMinimalCover

$S \longleftarrow$ *set of maximal nodes*
$MIN \longleftarrow \emptyset$
While $S \neq \emptyset$ *do* :
    $\forall_{f, f \in LARGEST}[$
       $\forall_{s, s \in S}[$ **if** $f \in s$ **then** $s \longleftarrow s - f$ **endif**
          if $s = \emptyset$ **then** $S = S - s$ **endif** $]]$
    $LARGEST \longleftarrow \alpha : \alpha \in S$ *and* $\forall_{s, s \in S}[|\alpha| \geq$
    $|s|]$
    $S \longleftarrow S - LARGEST$
    $MIN \longleftarrow MIN + LARGEST$
endWhile

Figure 2: Greedy algorithm to find a minimal covering which is stored as the set MIN.

Lemma 1 *If there exists a case where all of the plan fragments in a node, N, are mergeable with other nodes, $\{ON\}$, in a minimal covering generated by the algorithm of figure 2, then $|N| > |n|$, $\forall_n [n \in \{ON\}]$.*
Proof: *The proof is contained in [Britanik, 1994].*

The result of the lemma is that we only need to check those nodes that have cardinality less than the cardinality of the node of interest. This can be done for all nodes, n_i, in $O(n^2)$ time.

In the subplan sequencing graph, a directed edge is drawn from node N to node M if node N contains a fragment that must be sequenced before a fragment contained in node M. If an edge already exists between the two nodes, then the ordering is added to the ordering list for that edge (the ordering list is the edge label). If one maintains an explicit list of orderings of plan fragments, then the edges could be generated in $O(K)$ time where K is the total number of orderings. After generating edges in the subplan sequencing graph, it is possible that cycles may exist in the graph.

2.2 Removing cycles from a subplan sequencing graph

Cycles prevent us from determining an ordering between plan fragments; therefore, they must be removed. But before cycles can be removed, they must first be detected and enumerated. The cycle enumeration algorithm is a straight-forward depth-first search algorithm which explores each edge of the subplan sequencing graph and reports cycles when it encounters a node more than once along a path. This algorithm runs in $O(E)$ time, where E is the number of edges (external orderings) in the graph, since every edge in the graph is explored exactly once. Once we have found all of the cycles in the subplan sequencing graph, we need to remove them by deleting or relocating specific edges in the graph. This is done by moving plan fragments from one node to another node. The choice of which edge to break in a cycle depends on three factors: 1) the number of cycles to which the edge belongs, 2) the number of plan fragment moves required to remove the edge, and 3) the number of references to the plan fragment that would be moved. A reference to a plan fragment is simply the appearance of that fragment in the label of another edge in the graph. Consider the example subplan sequencing graph of figure 3. We
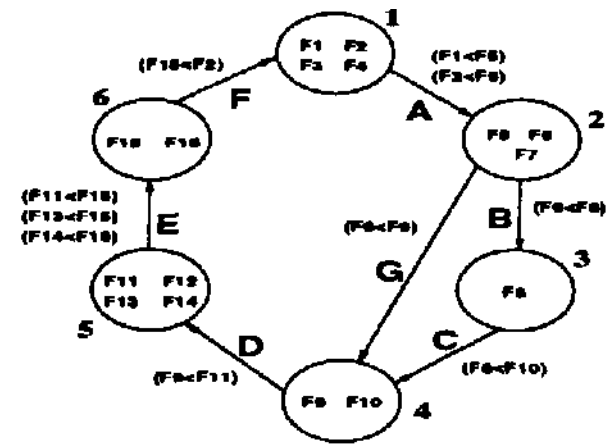

Figure 3: Example subplan sequencing graph with two cycles: (1 2 4 5 6 1) and (1 2 3 4 5 6 1).

define a *break set* as a set of plan fragments that when moved will break an edge. Edge E in the graph is labeled with three fragment pairs. These correspond to the three orderings between fragments in nodes 5 and 6. Let us call the first member of an ordering the source fragment or S-fragment, and let us call the second member of an ordering the destination fragment or D-fragment. The following is a list of all of the possible break sets for the edge **E**: (F11 F13 F14), (F11 F13 F16), (F14 F15), and (F15 F16).

We weight each break set of each edge in the cycle to be broken by the following evaluation function:

$$F(breakset) = (\#cycles\_contained\_in)$$
$$- (\#fragments\_to\_move)$$
$$- \Sigma(\#refs\_per\_fragment\_to\_be\_moved)$$

The edge break set which has the largest value of this function is the one most favorable to break. Breaking an edge requires that fragments be moved from the nodes involved to other nodes, but it may be the case that a fragment cannot be merged with any other node. Another problem that would prevent one from breaking an edge is if moving a fragment to another node induces a new cycle in the graph.

In summary, the following must be true for each fragment to be moved to an alternate node in the subplan sequencing graph to break an edge: 1) the fragment (and any fragments which have an identical-merge-action interaction with this fragment[2]) must be mergeable into the new node; that is, it is contained in that node's corresponding maximal node, 2) moving the fragment (and any fragments which have an identical-merge-action interaction with this fragment) to the new node does not induce a new cycle in the subplan sequencing graph, and 3) the fragment must not have been placed in the new node in a previous attempt to break the edge (avoids infinite loops). If it is the case that a cycle can not be broken by moving fragments to alternate nodes in the subplan sequencing graph, then there are two other approaches to removing the cycle: 1) use an alternate minimal covering, and 2) add additional nodes to the original graph.

[2] Two fragments have an identical-merge-action interaction if they must be merged in the same node at some level of the hierarchical plan graph.
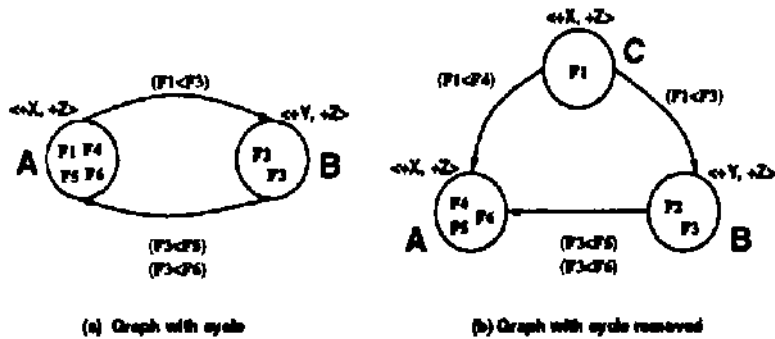
Figure 4: An example subplan sequencing graph before (a) and after (b) cycle removal.

A cycle in the subplan sequencing graph can be broken by adding one node to the graph if the following are true: 1) there exists a break set, *B*, which contains all S-fragments or all D-fragments, and 2) all fragments in *B* can be moved to the new node. To break a cycle by adding a single node to the subplan sequencing graph, it is first necessary to find a break set that meets the above enumerated criteria. This can be done by scanning the break sets in order of cost (see section 2.2). Let *E* represent the edge being removed and *B* represent the break set being used. A new node, N', is created with the same rnergeability type as the source node of *E* if *B* contains all S-fragments, or the destination node of *E* if *B* contains all D-fragments. All of the fragments in *B* are then moved from their original node to N'. Finally, the edges are updated by removing all edges from the graph and rebuilding them incorporating N'. Figure 4a shows a subplan sequencing graph which contains a cycle. A list of the break sets is as follows, ranked left to right: {FI}, {F3}, {FB, F6}, {F2, F3}. And the following are the orderings among the plan fragments: (FI α F3), (FI α F4), (FI α F5), (F4 α F5), (F3 α FB), (F2 α F3), (F2 α F6), and (F3 α F6). Each of the fragments in the nodes of the graph can be contained only in their current node. For example, fragment FI can not be moved to node B to break the cycle. Since none of the fragments can be moved to another node currently in the graph, a new node must be added. {FI} is an eligible break set; hence a new node, C, is created which has the same rnergeability type as node A. *F* is placed in node C and the edges of the graph are regenerated to yield figure 4b, and the cycle has been broken.

If breaking the cycle by adding one node to the graph is not possible, then it may be possible to break the cycle by adding two nodes. Note that if it is possible to break a given cycle, then that cycle can be broken by the addition of at most two nodes to the graph[3]. We can simply duplicate the two nodes at the ends of an edge in the cycle and move the fragments inducing the edge. This is a similar process to that of adding one node to the graph as discussed above.

If the cycle can not be broken by creating one or two new nodes in the subplan sequencing graph or by moving fragments to alternate existing nodes in the graph, then an alternate minimal covering may need to be generated and/or replanning may have to be done.

---

[3]Proof of this is presented in [Britanik,1994].

3 Replanning with Alternate Choices

From time to time it is necessary to replan some portion of the overall plan to compensate for changing criteria or to satisfy interactive user preferences. We consider replanning in the hierarchical plan graph (HPG) on a level-by-level basis. The following will demonstrate how the decomposition approach of hierarchical plan merging reduces the amount of rework necessary due to replanning of a specific plan component.

We approach replanning as follows. Replanning at level i of the HPG implies that one or more of the nodes at level i contain plan fragments that were reworked. All of the nodes that are successors of those that were replanned must also be replanned (or remerged). We consider separately the following two cases: 1) only one node on a branch of level *i* contains replanned fragments, and 2) more than one node on a branch of level i contain replanned fragments. A branch of a level is the group of nodes which have a common parent at the previous level of the HPG. Consider figure 1. At level 2, {C0$_T$C1}, {C2,C3,C4}, and {C5,C6} are the three branches.

When one node on a branch of a level is replanned, say *R*, then two actions will result: 1) it may be necessary to reorder the nodes within that branch, and 2) it is necessary to replan or remerge all successors of *R*. However, it is not necessary to replan other nodes in the same branch or in other branches on the same level. It is only necessary to replan the successors of the node that was replanned. This is a direct result of our decomposition approach in hierarchical plan merging. Consider node CI in figure 1. Replanning in node CI will require the consideration of only nodes C2, D2, D3, and D4. A planner which does not use this approach may have to replan at least all of level 2 and all of level 3. If we coarsely approximate the amount of replan work in each node as one unit, then our approach would yield a 17/21 or 80% savings in replan work, since only four nodes out of 21 would need to be reworked.

It is sometimes the case that replanning affects more than one node of a branch in the HPG. This would occur in situations where plan fragments were moved from one node to another in the course of replanning. In this case, it would be necessary to replan the entire branch of the level that was affected. This is due to the fact that the minimal covering set of nodes may have changed due to the relocation of plan fragments. Consider the second branch of level 2 in figure 1 which consists of nodes C2, C3, and C4. All the nodes of this branch as well as their successors would have to be replanned; however, the effect of replanning this branch is still decoupled from the other branches in level 2. Using the same cost assumptions as above, this decoupling yields a savings of 14/21 or 66% over a planner that would reconsider all of the nodes on level 2 and level 3.

If nodes in more than one branch of a level are affected by replanning, then each branch of that level can be considered as one of the two cases enumerated above, and replanning can propagate on a branch-by-branch basis.
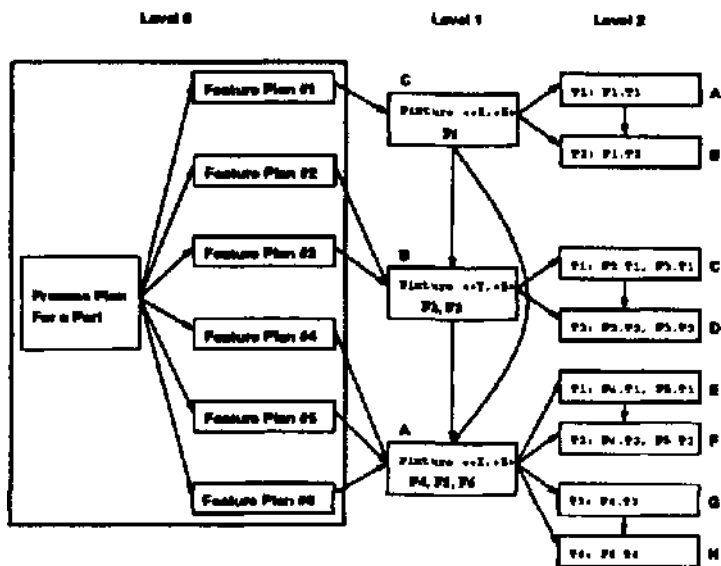
Figure 5: Hierarchical process plan graph incorporating the tooling subplan sequencing graphs into the tooling level (level 2).

4 Hierarchical Process Plan Merging

In this section we demonstrate an application of hierarchical plan merging in the domain of process planning. As previously discussed, we assume the availability of a process planner which will generate subplans for the individual features of a part. The output of such a planner is a set of feature subplans and a set of orderings between the subplans. Examples of this type of planner can be found in [Marefat and Kashyap,1992; Britanik and Marefat,1995],

4.1 Approach

Our approach to hierarchical process plan merging is a three level approach as shown in figure 5. Level 0 is the output of the planner, a sequenced list of feature subplans. Level 1 is the grouping of feature subplans into common fixtures. Each mergeability type (node) at this level is a unique fixture specification. Each feature subplan which is a member of a given node in level 1 can be executed in the fixturing setup specified by that particular node. Directed edges between nodes implies that the fixtures represented by the source nodes of the edges must be sequenced before the fixtures represented by the destination nodes of the edges. Level 1 establishes a minimal set of fixturings and their appropriate sequence for executing the global plan. Each fixturing naturally decomposes into a set of toolings for that fixture. Level 2 is the grouping of feature subplan fragments into common tooling groups. Each mergeability type at level 2 is a unique tooling for the branch (fixturing parent from level 1) the node is in. Each subplan action that is a member of a node in level 2 can be executed using the tooling specified by that particular node. Directed edges between nodes in level 2 (toolings) implies that certain toolings must be executed before other toolings in the current fixturing setup. Level 2 establishes a minimal set of toolings necessary to execute the plan fragments in the particular parent fixturing.

Figure 6b shows an interaction graph for the object in figure 6a.
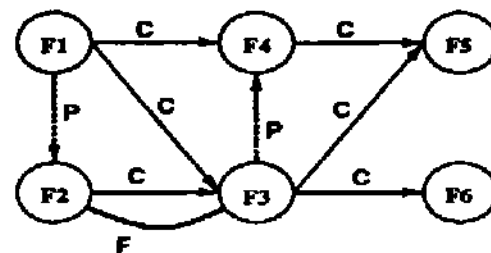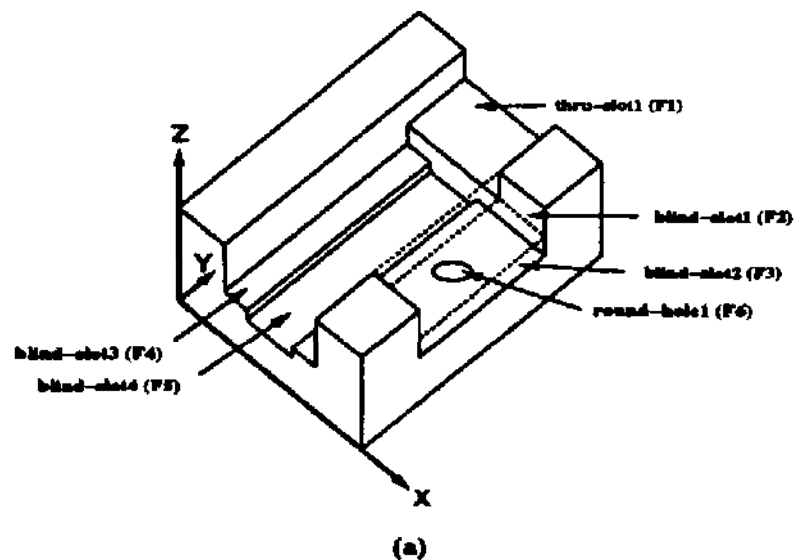

(a)

Figure 6: (a) An example object with several interacting features and (b) its associated interaction graph. In the graph, solid edges represent strong constraints, while dashed edges represent weak constraints. An edge label of C represents a containment interaction and an edge label of P represents a perpendicularity interaction. An edge label of F represents a fixturing identical-merge-action interaction.

For our planner, we will focus on generating fixture positions for a parallel vise-clamp type fixture. However, the hierarchical plan merging mechanism is not limited to the simplified fixture model we will present. We use this simple model to facilitate clear exposition of the process planning domain application. Along with the clamp-type fixture, we will assume a vertical-type machine; that is, the tool chuck approaches the part from above the fixture. Given this physical description, we can model a fixture setting as a two-tuple $< CA,0 >$, where CA is the clamping axis and O is the orientation of the part about the clamping axis. The CA is specified as the direction parallel to the principle normal (in the part's local coordinates) of the faces to which the clamp is applied. O is the direction of the principle normal of the part surface that is facing up towards the tool chuck.

4.2 Process Plan Merging

We will now discuss our application of hierarchical plan merging to process planning. First we describe the subplan sequencing graph at the fixturing level, then at the tooling level.

Fixturing Level

Once we know from which fixtures a feature can be machined, we can construct the fixture subplan sequencing

```
F<+X,+Z>:{F1,F4,F5,F6}    F<-X,+Z>:{F1,F4,F5,F6}
F<+Y,+Z>:{F2,F3,F6}       F<-Y,+Z>:{F2,F3,F6}
F<+X,-Z>:{F6}             F<-X,-Z>:{F6}
F<+Y,-Z>:{F6}             F<-Y,-Z>:{F6}
```

Figure 7: Common fixture sets for the part of figure 6a.

graph. First we build a list of features applicable to each fixture. We call these lists *common fixture sets* (CFS). The common fixture sets for the part in figure 6a are shown in figure 7. Notice that a feature may appear in more than one CFS, demonstrating alternative fixtures from which the feature may be machined. Also note that common fixture sets correspond directly to the set of maximal nodes as discussed in section 2.1.

The next step in building the fixture subplan sequencing graph is to generate a minimal covering of nodes; that is, a minimum set of fixturings. This corresponds directly to our desire to minimize the number of fixture changes while machining the part. The minimal covering for the example part consists of the two nodes: F<+X,+Z> = {F1,F4,F5,F6} and F<+Y,+Z> = {F2.F3}.

Since our minimal covering algorithm really only generates a near-minimal covering, we next check to see if we can can remove any unnecessary nodes as described in section 2.1. By observing the minimal covering and figure 7, we see that there are no redundant nodes which can be subsumed by nodes of lesser cardinality.

With this minimal set of fixtures, we now determine the sequence by which the fixtures are used in machining the part. This sequence is constrained by the strong interactions between features in different fixtures. Using the interaction graph in figure 6b, we have an explicit list of orderings (strong interactions), and can generate the edges for the fixture subplan sequencing graph as discussed in section 2.1. We simply examine each strong interaction (represented by solid arrows in figure 6b), and include it in the fixture subplan sequencing graph as an edge if its source and destination features are in different fixture nodes (ie: it represents an external ordering). The fixture subplan sequencing graph after edge generation is shown in figure 4a. Note that this graph has a cycle; hence, cycle removal is necessary. The list of break sets is as follows, ranked left to right. {FI}, {F3}, {F5, F6}, and {F2, F3} The first approach of moving one or more plan fragments (feature subplans) to other nodes in the graph would be unsuccessful since only F6 can be moved to another node in the graph (this can be seen by observing the common fixture sets in figure 7), but there is no break set which contains only F6. Hence, the second approach of node addition to the graph will be attempted. By observing the ranked list of break sets, we see that all of the break sets contain nodes that are either all S-fragments or all D-fragments. If it is possible to break the cycle, it can be done by adding only one node to the fixture subplan sequencing graph. Break set {Fi} has the highest ranking and is an eligible break set; hence node A of figure 4a is duplicated with the fragment FI as its contents. Using the interaction graph of figure 6b, the edges of the fixture subplan sequencing graph are regenerated incorporating the new node. The

resulting cycle-free fixture subplan sequencing graph is shown in figure 4b.

Now that the fixturing subplan sequencing graph, that is, the fixturing level of the hierarchical process plan graph, is complete, it is necessary to build the tooling level by generating tooling subplan sequencing graphs for each fixture node in the fixturing level.

Tooling Level

To proceed with plan merging at the tooling level, we need to refine our definition of a plan fragment for this level. For the purposes of our example, we consider each feature subplan to consist of an ordered set of tool applications, one for each process used to machine the feature. We represent the application of tool TI in feature subplan FI as FI.TI. If FI consists of applying tool TI followed by applying tool T2, then FI contains two plan fragments, namely F1.T1 α F1.T2. It is these plan fragments that will be merged into common tooling sets in the tooling subplan sequencing graph. Since the prismatic features in figure 6a (features FI through FB) are relatively similar, we assume that they all use the same rough cut tool, TI, and finishing cut tool, T2. The round hole (feature F6) uses tool T3 *as* its rough cut tool and T4 as its finishing cut tool. Note that there is a strong constraint which sequences tool TI before tool T2 in each of the first five feature subplans, and T3 is sequenced before T4 in F6. This ordering is due to the common sense fact that the rough cut must precede the finishing cut. With this information, we are now ready to generate the tooling subplan sequencing graph for node A in figure 5. The generation of the tooling subplan sequencing graphs for the other nodes of level 1 in the hierarchical process plan graph is a similar process and will not be shown explicitly.

First we need to generate the set of maximal nodes. One node is generated for each specific tool used in the feature subplans. The following is the set of maximal nodes for the tooling subplan sequencing graph: (TI:F4.T1,F5.T1), (T2:F4.T2,F5.T2), (T3:F6.T3), and (T4:F6.T4). Since all fragments in all nodes are unique, the set of maximal nodes is also the minimal covering in this case. Also, there are no redundant nodes which can be removed from the minimal covering. To generate the edges in the tooling subplan sequencing graph, we utilize the orderings in the interaction graph in figure 6b as well the orderings implied by the specification that rough cuts precede finishing cuts. The result of adding the appropriate edges is the cycle-free tooling subplan sequencing graph shown in level2 of figure 5, which is the final hierarchical process plan graph.

Now that the hierarchical process plan graph is complete, we can generate an outline of the final plan (it is an outline in the sense that we do not include specific motions and dimensional data etc.). The plan is shown below with the execution sequence from top to bottom and then left to right:

```
setup fixture <+X,+Z>      setup fixture <+X,+Z>
   setup tool TI              setup tool TI
     { do actions }             { do actions }
```

```
setup tool T2              setup tool T3
  { do actions }             { do actions }
setup fixture <+Y,+Z>      setup tool T2
  setup tool      T1         { do actions }
    { do actions }         setup tool T4
  setup tool T2              { do actions }
    { do actions }
```

5 Related Work

Our work is somewhat complementary to that of Karinthi et al. [Karinthi et a/., 1992]. Their interesting work focuses on generating an optimal global plan from all possible alternatives of feature subplans in process planning via state-space search. Our focus has been on merging one specific set of feature subplans such that replanning effort is minimized when alternatives are selected. However, in a sense we also consider many alternative plans during the merging process at the fixturing level.

The work of Foulser et al. [Foulser et ai, 1992] presents a formal treatment of the complexity of domain-independent plan merging using STRIPS-style operators. An optimal algorithm for plan merging utilizing dynamic programming methods has been developed. Since practical implementation of the optimal algorithm is infeasible for larger inputs, several greedy-based approximate (near optimal) algorithms are also developed along with their worst-case and average-case complexities for large inputs. It is also empirically shown that the approximate algorithms performed well for larger inputs. Yang et al. [Yang et al,1992] generalizes some of these algorithms to handle a wider range of interaction types.

Hayes [Ilayes,1989] has developed the Machinist system which considers operator overlap in process planning. The approach uses "cues" in the problem specification to search for mergeable operators. Like our near-minimal covering algorithm, Machinist uses a greedy-type algorithm to group operators into a near-minimal number of set-ups (fixturings).

HUTCAPP [Mantyla and Opas,1988] is a generative process planner for prismatic parts. A lattice algebra is used to find a minimal covering of work directions to machine all features in the part. This work inspired our lattice algebra approach to generating an optimal minimum covering from the set of maximal nodes. This algorithm and another optimal algorithm are analyzed in [Britanik,1994].

6 Conclusions

We have developed a systematic methodology for uniform plan merging at the various levels of plan abstraction. This method manifests itself in the hierarchical plan graph where each level contains a complete, partially merged plan. The principle advantage of this approach is that, once external interactions between nodes on a given level have been established, the continued merging of the plan fragments in one node can take place independently of plan fragments in other nodes on that level. This provides a decomposition or divide-and-conquer approach to plan merging. Another advantage

to this decomposition approach is that replanning effort is minimized in the presence of the selection of alternative actions at some level of the hierarchical plan graph. Only those plan fragments which are in the same branch as the alternative selection need be considered for replanning.

Rather than considering mergings which result in cyclic dependencies to be infeasible, an algorithm is proposed which takes a bilateral approach to breaking cyclic dependencies between nodes in the subplan sequencing graph. The first approach is to maintain the same number of nodes and remove the edge of the cycle by relocating plan fragments to other nodes. If this approach fails or is inappropriate according to the priorities of the planner, then a second approach which considers adding at most two nodes to the subplan sequencing graph may be used. If these two approaches have failed, then the merging which resulted in the cyclic dependency will be considered infeasible and replanning will be necessary.

We have demonstrated the utility of this hierarchical approach to plan merging through examples in the process planning domain. It was shown that the merging of fixtures (fixture level) and the merging of tools (tooling level) can be done in a systematic and uniform fashion.

References

[Britanik and Marefat, 1995] J. Britanik and M. Marefat. Case-based manufacturing process planning with knowledge sharing support. *Submitted for journal publication,* 1995.

[Britanik, 1994] John Britanik. Hierarchical plan merging. Technical report, Intelligent Systems Laboratory, Department of Electrical and Computer Engineering, University of Arizona, 1994.

[Foulser et al, 1992] David E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence,* 57:143-181, 1992.

[Hayes, 1989] Caroline Clarke Hayes. A model of planning for plan efficiency: Taking advantage of operator overlap. In *International Joint Conference on Artificial Intelligence,* pages 949-953, 1989.

[Karinthi et ai, 1992] Raghu Karinthi, Dana Nau, and Qiang Yang. Handling feature interactions in process-planning. *Applied Artificial Intelligence,* 6:389-415, 1992.

[Mantyla and Opas, 1988] Martti Mantyla and Jussi Opas. HUTCAPP - a machining operations planner. In *Robotics and Manufacturing,* pages 901-910. ASME, November 1988.

[Marefat and Kashyap, 1992] M. Marefat and R.L. Kashyap. Automatic construction of process plans from solid model representations. *IEEE Transactions on Systems, Man, and Cybernetics,* 22(5):1097-1115, September/October 1992.

[Yang et ai, 1992] Qiang Yang, Dana S. Nau, and James Hendler. Merging seperately generated plans with restricted interactions. *Computational Intelligence,* 8(4):648~676, 1992.