Graph-Theoretic Generation of Assembly Plans
Part II: Problem Decomposition and Optimization Algorithms

Kedar S. Naphade
Bell Labs, Lucent Technologies
Princeton, New Jersey

S. David Wu and Robert H. Storer
Lehigh University
Bethlehem, Pennsylvania

Abstract

Efficient generation of assembly plans has significant implications on manufacturing costs, however, the underlying mathematical problem is both theoretically and computationally intractable. In the previous paper (Part I), we developed a graph-theoretic framework for the general problem, and we described a constraint satisfaction construct that generates correct assembly precedence graphs given establishment conditions. In this paper, we develop additional components for the framework: a *problem decomposition scheme* which decomposes a general set of establishment conditions into generalized 2-SAT problems. After mapping these satisfiability problems into specialized decision graphs, we develop a *branch-and-bound algorithm* which generates optimal precedence graphs and optimal assembly plans from the decision graph. Our method generates assembly precedence graph(s) which satisfy two basic criteria:

(i) All sequences defined by the precedence graph are feasible according to design specifications,

(ii) Given any graph-computable performance measure (sparsity, number of node-disjoint paths etc), the precedence graph is optimal.

The latter links the graph-theoretic framework to practical applications where performance criteria motivated by line balancing and scheduling can be directly associated to assembly plan generation. Computational results show that the proposed algorithm can solve moderate-size problems within seconds. Finally, we investigate *alternative decomposition schemes* and explore their theoretical and computational implications, including redundancy, dominance, and global optimality.

1. Motivations

Assembly Planning problems have received a lot of attention over the past fifteen years. The problems encompass modeling and representation of assembly constraints, (e.g. Bourjalt, 1984), generation of feasible assembly plans (Homem de Mello and Sanderson, 1990) and selection of assembly plans for final assembly (Bonneville et. al., 1995).. Any mechanical assembly process can be decomposed into a set of tasks, where each task involves joining two or more components or subassemblies together. Given a set of establishment conditions (precedence constraints), the *sequence generation problem* involves generating one or more sequences in which all the tasks in this set can be performed in order to feasibly assemble the product. Similarly, the *precedence graph generation* problem involves generating precedence graphs, such that all assembly sequences defined by them are feasible.

The sequence of tasks in which assembly is performed can have a significant impact on cost and efficiency through both quantitative and qualitative measures. Quantitative measures may be derived from resource allocation, line balancing, scheduling, or more direct measures such as number of re-orientations and number of tool changes required. Qualitative measures, or perhaps measures difficult to quantify may include ease and stability of assembly, fixturing requirements and complexity of operations.

In this paper, we focus our attention on the optimal generation of precedence graphs. We generate and select optimal precedence graphs from a correct and complete set. The word *complete* implies that *all* possible assembly sequences

can be obtained from this set of precedence graphs. The word *correct* implies that all these sequences are *feasible*, i.e. they satisfy all the stated establishment conditions or assembly constraints. A basic thesis of this paper is that a precedence graph holds more merit as an assembly "plan" than a fully specified assembly sequence. This is true because a precedence graph permits flexible execution of activities while delaying "less critical" decisions to the last minute (Wu et. al.,1999). This view is carefully examined and justified in the previous (Part I) paper (Naphade et al., 1999), where we propose a theoretical framework for the correct generation of precedence graphs given establishment conditions.

2. Problem Definition

The problem of primary interest in this paper, is the Precedence- Graph Optimization Problem or POP. Embedded within POP is the constraint satisfaction problem or CSP. CSP determines a "correct" precedence graph - a precedence graph that satisfies the specified set of assembly constraints. A correct precedence graph in term defines a set of correct assembly sequences. POP refers to the problem of determining a precedence graph that is not only correct but also optimal given a performance criterion of interest.

The method developed in this paper requires that the performance measure be "graph-computable", i.e. it can be uniquely derived from the structure of the graph. Examples of such performance measures include sparsity, number of node disjoint paths, average nodal in-degree etc. Many resource allocation and scheduling criteria however are not only graph based but also sequence based. For instance, in order to optimize the expected performance of an assembly plan in the context of assembly line balancing or resource constrained project scheduling, sequence based criteria are necessary.

We now define CSP and POP more formally. Assume that the product to be assembled contains *m* parts. A typical assembly task connects (or establishes a liaison between) two or more of these *m* parts. Let there be *n* such liaisons or assembly tasks involved in fully assembling the product. Clearly, some sequences for performing these tasks may be infeasible. Similarly, if each assembly task represents a node on a precedence graph, all precedence graphs constructed on *n* nodes may not necessarily feasible. Precedence constraints are imposed by spatial or functional limitations such as accessibility, stability and resource requirements during the assembly. Thus, CSP and POP can be stated as follows:

CSP: *Generate a precedence graph with assembly tasks as nodes and task precedence as arcs that satisfies all establishment conditions.*

POP: *Given a minimizing performance criterion Z(H) on precedence graph H, generate a precedence graph H^* that satisfies all establishment conditions and $Z(H^*) \leq Z(H)$ for all feasible precedence graphs H.*

3. Literature Review and Need for Optimization Methodologies

Bourjalt(1984) started research in assembly sequence generation. Through a question and answer method, he obtained establishment conditions for assembly tasks. DeFazio and Whitney(1987) improved Bourjalt's method by reducing the number of questions needed. They developed the "diamond graph" method to generate all assembly sequences through a directed state-transition graph in which the nodes represent partial assembly states. Later Baldwin et. al.(1991) developed an integrated computer aid to generate and evaluate sequences using the method provided by DeFazio and Whitney (1987). Homem de Mello and Sanderson(1990) introduced the use of AND/OR graphs to solve the sequence generation problem. We assume that the establishment conditions are input to our method. Baldwin et. al. (1991), Bourjalt (1984), DeFazio and Whitney (1987) etc. provide methods for generating these conditions. Homem de Mello and Sanderson(1991a) demonstrate the equivalence between different representations of assembly constraints and also provide mappings of the different representations onto one another.

As noted in the Part I paper, CSP is by itself is NP-Complete. Adding an optimization problem on top of the CSP makes the overall problem even more challenging. Perhaps for this reason, research on automated selection or optimization of assembly plans has been limited. Lee and Raz (1989) solve an integrated sequencing and robot selection

problem, but assume the availability of a precedence graph. Homem de Mello and Sanderson (1991c) present an integrated assembly sequence generation and optimization algorithm by maximizing the number of parallel tasks. This algorithm is based on the AND/OR graph representation. A similar procedure is developed by Lapierre and ElMaraghy (1994). Bonneville et. al. (1995) provide a genetic algorithm to generate and evaluate assembly plans. They start off with a set of feasible assembly plans provided by an expert and generate more plans by combining plans from the initial set. Since they assume the prior availability of several feasible plans, the NP-completeness of CSP is not an issue for their methodology. Minzu and Henrioud (1993) adopt a two stage approach to solve the POP. They first systematically generate all valid assembly plans for the product and then determine for each one, all possible assembly systems (sequences or line balancing configurations). This is a total enumeration approach which becomes impractical as the problem size increases.

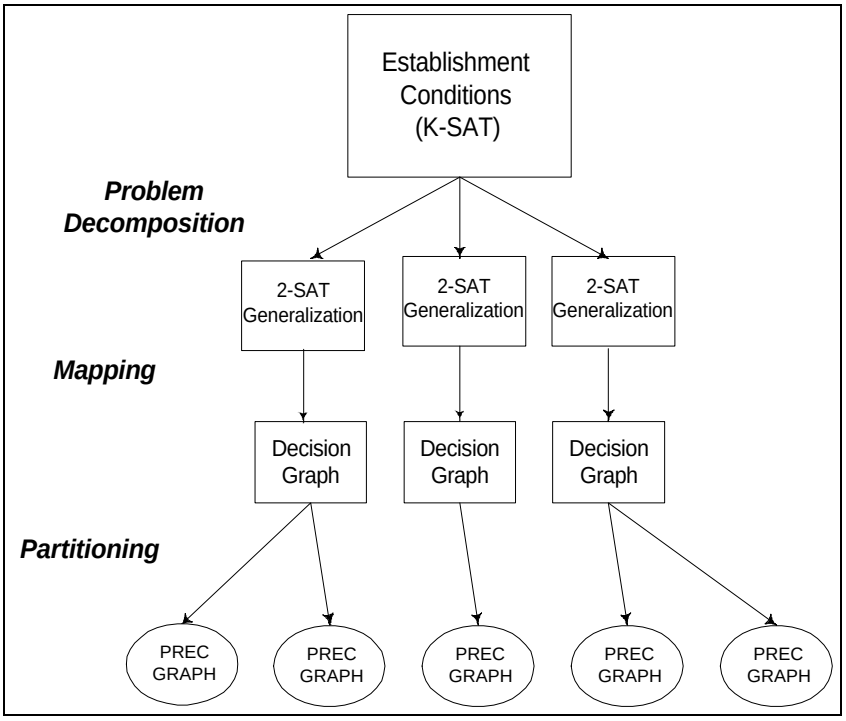

Figure 1
The Hierarchical Graph-Theoretic Framework

4. The Graph-Theoretic Framework- A Summary of Previous Results

We first summarize main components of the graph-theoretic framework described in the Part I paper. Basic to the framework is an extended predicate calculus operator (→) (reads "must precede"). Basic properties of this operator are as follows:

1. **Transitivity :**
((A→B) *and* (B→C)) implies A→C.

2. **Distributive Properties :**

(i)	A→(B and C)	is equivalent to	(A →B) *and* (A→C).
(ii)	A→(B *or* C)	is equivalent to	(A →B) *or* (A→C)
(iii)	(A *or* B) → C	is equivalent to	(A→C) *or* (B→C)
(iv)	(A *and* B)→C	is equivalent to	(A→C) *and* (B→C)

3. **Negation :** We shall use the tilde **(~)** sign for negation in this paper. If ~ (A→B) is true, then either (B →A) is true or there is no constraint between tasks A and B. (They may be performed simultaneously).

Overall scheme of the graph-theoretic framework for assembly planning is summarized in Figure 1. The framework consists of three basic steps: *Problem Decomposition*, *Mapping*, and *Partitioning*. In the *Problem Decomposition* step, we convert a given set of establishment (assembly design) conditions into a set of generalized two-satisfiability (2-SAT) problems. To solve each of these subproblems, we *map* each 2-SAT onto a "decision graph" such that a proper partitioning of the graph would correspond to a "feasible and consistent" assembly precedence graph. The *problem decomposition* and the *partitioning* steps of this framework are the main foci of this paper. In the following, we describe the main essence of these two steps.

Problem Decomposition

Given a set of establishment conditions ***(P)*** we decompose them into ***N*** subproblems $\boldsymbol{Q_i}$ **.** Each $\boldsymbol{Q_i}$ is an instance of a generalized 2-SAT problem, where all clauses in their normal form are either single literals or a disjunction of two literals. A sub-problem $\boldsymbol{Q_i}$ is a generalization of the 2-SAT problem in that a literal (representing an arc between nodes a, b on a precedence graph) can assume three, rather than two, values: a→b, b→a and a≡ b where a≡ b implies that there is no arc between nodes a and b on the precedence graph. The simple scheme described in the following can be used to solve a generalized 2-SAT problem where a literal could take an arbitrary number of values, and different literals may take different numbers of values.

Some establishment conditions in ***(P)*** are conjunctive precedence constraints (e.g. 3→4), some are disjunctions of length 2 (e.g. 2 or 3 → 4) and the others are disjunctions of length 3 or more (e.g. 1 or 2 or 3 → 4). Problem ***P*** is decomposed as follows. Disjunctive constraints of length three or more are split arbitrarily into subconstraints of two literals each. For example:

(1 or 2 or 3 → 4) is split as (1 or 2) → 4
3 → 4

(4 or 5 or 6 or 7 → 8) is split as (4 or 5) → 8
(6 or 7) → 8

Thus for every long (length > 3) establishment condition, we form a group of subconstraints. A subproblem $\boldsymbol{Q_i}$ is defined as follows:

a) $\boldsymbol{Q_i}$ inherits all constraints of length 1 and 2 from ***P***.
b) From each group of subconstraints associated with long establishment conditions in ***P***, one and only one constraint is included in $\boldsymbol{Q_i}$.

The following example clarifies the above decomposition rules:

P: 1 → 2
(1 or 4) → 5
(3 or 4 or 5)→ 6 Group of subconstraints: (3 or 4)→ 6; 5 → 6
(3 or 5 or 6 or 7) → 9 Group of subconstraints: (3 or 5)→ 9; (6 or 7)→ 9

$\boldsymbol{Q_1}$	1 → 2 ;	(1 or 4) → 5;	(3 or 4)→ 6;	(3 or 5)→ 9
$\boldsymbol{Q_2}$	1 → 2 ;	(1 or 4) → 5;	5→ 6;	(3 or 5)→ 9
$\boldsymbol{Q_3}$	1 → 2 ;	(1 or 4) → 5;	(3 or 4)→ 6;	(6 or 7)→ 9
$\boldsymbol{Q_4}$	1 → 2 ;	(1 or 4) → 5;	5→ 6;	(6 or 7)→ 9

In the Part I paper, we proved that this simple decomposition scheme satisfies the following properties:

*(a) A solution that is feasible for **P** is feasible for at least one Q_i.*
*(b) A solution that is feasible for any Q_i is feasible for **P**.*

In other words, the set of all solutions for all the 2-SAT problems Q_i is identical to the set of solutions of the original satisfiability problem ***P***. Hence a correct and complete generation of solutions for each of the subproblems will constitute a correct and complete generation of solutions for the original problem. While the above simple scheme is correct, it does not address issues of computational efficiency. Alternative decomposition schemes will be the subject of discussion in Section 7 which considers computational efficiency and potential redundancy.

As detailed in the Part I paper, the after decomposition the generalized 2-SAT can be mapped into a unique decision graph. The next step is to partition this decision graph.

Partitioning the Decision Graph

Suppose a node I in the decision graph represents a precedence constraint a→b, denote its two complements b→a and a ≡b by nodes I_c and I_O respectively. We further define a complement set C_I which contains the complements of I. For instance the complement set for node I contains the nodes I_o and I_c, the complement set for node I_o contains the nodes I and I_c. The following propositions summarize important properties of the partitioning problem.

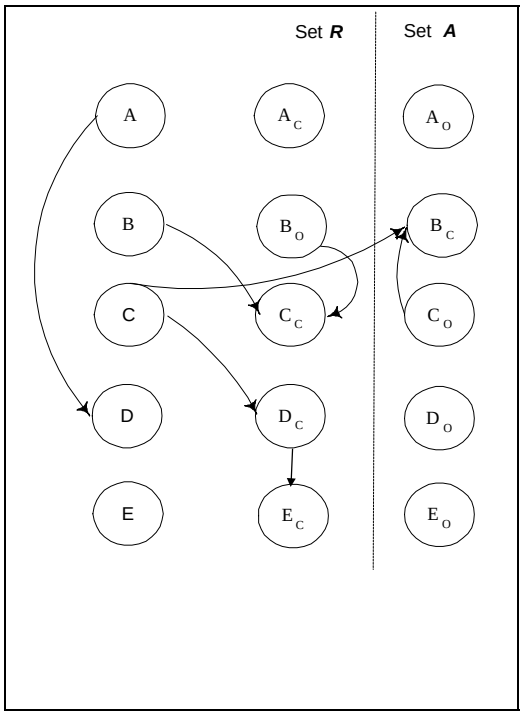

Figure 2
Feasibly Partitioned Decision Graph

Proposition 1

A feasible and consistent partition of G^* into sets ***R*** and ***A*** must satisfy the following properties:
*Property 1 : From the set of nodes {I, I_o, I_c } one and only one node must be in set **A**. The other two nodes must be in set **R***
*Property 2 : ∃ arc IJ∈ G^* with I∈**A** and J∈ **R***

Property 1 ensures that for every decision, exactly one alternative is accepted. Property 2 ensures that decision dependent constraints are not violated.(If I implies J, and I is accepted, then J must be accepted). The graph partitioning problem can now be defined as follows:

*Partition graph G^* into two sets **R** and **A**, such that one and only one alternative (node) for each decision belongs to set **A** and any directed arcs that cross the partition, are directed from set **R** to set **A**.*

Figure 2 shows an example of a decision graph partitioned according to the above requirements.

For every node I on graph G^*, let R_I be the set of all nodes J such that there is a directed path from node I to node J. Let us define the set R'_I as the union of the complement sets of all nodes J in set R_I. Similarly, let T_I be the set of all nodes K such that there is a directed path from node K to node I.

Proposition 2
If I ∈ ***A*** then $R_I \subset A$ and hence $R'_I \subset R$.

Proposition 3
If I∈ ***R***, then $T_I \subset R$.

Proposition 2 states that the acceptance of alternative I completely resolves the decisions associated with nodes in $\boldsymbol{R_I}$. Proposition 3 statest htat the rejection of alternative I forces the rejection of alternatives in $\boldsymbol{T_I}$. Note that this does not completely resolve the decisions since there may be two other alternatives from which one needs to be accepted.

The above conclusions suggests a simple partitioning algorithm as follows:

Proc_Part(I):

1. Place node I in set $\boldsymbol{A}$.
2. Place nodes belonging to $\boldsymbol{R_I}$ in set $\boldsymbol{A}$ and nodes belonging to $\boldsymbol{R'_I}$ in set $\boldsymbol{R}$.
3. For all nodes K placed in set $\boldsymbol{R}$ at this point, place nodes belonging to $\boldsymbol{T_K}$ in set $\boldsymbol{R}$
4. If Inconsistency found, STOP.
   Else
   If there are any decisions with just one node not assigned to any set and all other nodes assigned to set $\boldsymbol{R}$, then place those nodes in set $\boldsymbol{A}$ in order to satisfy Property 1. As a result of this some nodes may be newly introduced into set $\boldsymbol{A}$. For each such node I, GOTO Step 1.

   Else STOP.

Proposition 4

Step 1 of Proc_Part(I) places a node I in set $\boldsymbol{A}$ and propagates its effect on the decision graph**.** This stipulates three possible outcomes:

Case 1: An inconsistency. Hence placing node I in set $\boldsymbol{A}$ is not a feasible decision.

Case 2: The graph is completely partitioned without inconsistency. This results in a feasible partition

Case 3: The graph is partially partitioned without inconsistency. In this case the remaining unpartitioned portion of the graph is independent of the partitioned subgraph. As a result, the partitioning procedure can now be applied recursively to the remaining subgraph.

Proc_Part terminates in $O(m)$ time where m is the number of arcs on the decision graph.

This concludes the summary of the graph-theoretic framework consisting of the problem decomposition, mapping and partitioning components . We now use the partitioning procedure **Proc_Part(I)** to find an optimal precedence graph using a branch-and-bound algorithm.

5. The Branch-and-bound Method

In this section we discuss the generation of all feasible precedence graphs and selection of an optimal assembly plan from ***a given decision graph***. Since the branch-and-bound procedure operates on a single decision graph, for cases where problem decomposition is necessary, the procedure has to be separately applied to each subproblem or decision graph. The specific decomposition scheme chosen could substantially affect the efficiency of the overall solution. We discuss issues related to decomposition schemes in section 7.

5.1 Branch and Bound Tree Representation.

We first describe the tree representation for the branch and bound algorithm. Complete generation of this tree is equivalent to generating all feasible precedence graphs, or equivalently, evaluating all partitions of a decision graph. The tree representation is based on the fact that every unique feasible combination of decision alternatives leads to a unique feasible precedence graph.

Node: A node in the branch and bound tree represents a partial or complete set of resolved decisions. Specifically, we associate with each node, a partially or completely partitioned decision graph.

Root Node: The root node of the branch and bound tree represents a state in which no decisions are taken. Associated

with the root node is the completely unpartitioned decision graph.

Leaf Nodes: An unfathomed leaf node represents complete feasible solutions: decision graphs that are fully and feasibly partitioned.

Fathomed Nodes: Nodes in the tree may be fathomed because of *sparsity cutting* or *feasibility cutting* described below.

Intermediate Nodes: Intermediate nodes in the tree represent partially solved problems: decision graphs that are partially partitioned due to the partitioning procedure having terminated with case 2 of proposition 4.

Transition: A Transition from a parent node to a child node represents the process of adding an independent decision (say alternative I) to the set of alternatives associated with the parent node. The partitioning procedure with alternative I as the argument is invoked for the decision graph associated with the parent node. This further partitions the decision graph to yield the decision graph associated with the child node. If one child is formed by adding alternative I to the accepted set of the parent node, the other two children are formed by adding the nodes I_o and I_c respectively.

Accepting alternative I and invoking the partitioning procedure leads to three possibilities according to Proposition 4:
1. An inconsistency: In this case, the node is pruned through "feasibility cutting" described below.
2. A completely partitioned decision graph: In this case, the child node is a feasible leaf.
3. A feasibly but partially partitioned decision graph: In this case, there are some more decisions to be resolved and the node is a potential parent, or an open node. In such a case, we must select the next decision to branch upon or the next node to partition. This creates a branching choice.

Branching: An unresolved decision is selected for branching. A child node is created for every decision alternative. If the decision is completely unresolved, three child nodes will be created. If the decision has been partially resolved due to elimination of one decision alternative, two child nodes will be created.

To select a decision to branch upon, we heuristically select an alternative (node on the decision graph) that is the "most influential", i..e. affects a large number of other decisions. In so doing, a large number of decisions could be resolved through the partitioning procedure, leaving us with a smaller unpartitioned subgraph. Hence a smaller branch and bound tree may result from using this criterion. The specific criterion used for this purpose is :

$$selected\_node = argmax(I) \{ (\text{intra-subgraph outdegree of node } I + \sum (\text{intra-subgraph indegree of complements of node I}) \}$$

Recall that the unpartitioned subgraphs are independent of the partitioned portion of the graph. Hence a selected node influences only those decisions that are present in its own subgraph. The first term represents a lower bound on the number of decision alternatives that will have to be accepted if node I is accepted. For every complement of node I, the indegree represents a lower bound on the number of decision alternatives that are eliminated if node I is accepted. Thus the above criterion is a heuristic myopic indicator of the influence of selecting node I as an accepted alternative.

Feasibility Cutting: A large part of the branch and bound tree can be cut when the co-existence of any pair of alternatives is infeasible. Suppose that an intermediate node in the tree represents a partially partitioned decision graph with alternative A as a part of its accepted set, the next decision selected for branching is decision C and C forces A_c. Since C forces A_c, C and A_c cannot co-exist. Thus any child node represents accepting C (and A) is infeasible and the branch can be pruned. We call this *"Feasibility Cutting"*.

Sparsity Cutting: Branches in the tree can also be pruned based on the following concept: Suppose the nodes I, I_c and I_o on the decision graph represent the decision alternatives $A \rightarrow B$, $B \rightarrow A$, $A \equiv B$ respectively for nodes (A, B) on the

precedence graph. Let us assume that the alternative I_o is feasible. Also assume that the nodes I, I_c and I_o are independent of the remaining decision graph, i.e., the alternatives I_0, I and I_c are not forced by or do not force any other alternative. In this case, accepting alternative I_o and rejecting I and I_c does not violate the completeness requirement. The reason for this is that selecting I or I_c adds constraints to the precedence graph whereas selecting I_o does not. The graph obtained by selecting I_o is a less constrained graph and hence contains all solutions that would accrue from the graphs generated using I or I_c. (A less constrained space always contains all the solutions in more constrained space.) This concept is used to cut entire sub-trees at a time, as follows:
If the set of nodes I, I_c and I_o have a zero intra-subgraph degree, the decision I does not influence any other remaining decisions. In such a case we select the alternative (I_o) that does not add an arc to the precedence graph, to form the child node. The siblings (precedence graphs obtained by selection of I or I_c) of this child node are fathomed. We call this *"sparsity cutting"*. A more detailed discussion of sparsity and its merits occurs in Section 5.2.

Preprocessing: The concept used for sparsity cutting can also be used in a more generalized form for the root node or the completely unpartitioned decision graph. Let nodes I_o, I, I_c represent the alternatives of a decision and let I_o be the no-arc alternative. If node I (or I_c) has exactly the same predecessors and successors as node I_o, then node I(I_c) can be eliminated from the decision graph. The reason for this is that node I(I_c) and node I_o have an identical influence on the remainder of the decision graph. Hence any solution with node I(I_c) in the accepted set will be feasible if node I(I_c) is swapped across the partition with node I_o. Since this is true, consideration of node I(I_c) is not necessary using the same logic as in sparsity cutting. This can be applied only for optimization of regular performance measures as will become clear in the next subsection.

Tree Generation Method: The tree is generated using a standard depth first search with backtracking method, incorporating the above branching and pruning strategies.

Initial Feasible Solution: An initial feasible solution may be obtained using the partitioning procedure of Naphade et. al.(1997) or using a depth first search to locate a feasible leaf.

Termination: This tree generation terminates when all branches have been completely explored and no further branching is possible.

Bound based Cutting: The above paragraphs describe the procedure for obtaining all feasible partitions of a decision graph. For solving the POP, additional bound-based cutting schemes based on the relevant objective function can be used, as we will describe in section 5.3.

5.2 Graph Subsets and Regular Performance Measures

The branch and bound method of section 5.1 can be used to solve the POP with a "regular" performance measure by introducing appropriate bounds. In other words, if Z is a regular performance measure, we can find a graph H such that $Z(H) \leq Z(H)$ for all feasible or correct graphs H. As defined in Section 2, a graph H is feasible if graph H and hence all the sequences defined by it satisfy all establishment conditions

Let F be a graph that contains all the arcs contained in a certain feasible graph H, and some more additional arcs (constraints). Graph F is feasible because graph H is feasible. If Q_H is the set of sequences defined by graph H and Q_F is the set of sequences defined by graph F, then $Q_F \subset Q_H$. In other words, all sequences defined by graph F can be generated from graph H, but not vice versa. We then say that graph F is a subset of graph H. If S_H is the set of subsets of graph H, then $F \in S_H$.

For a minimization problem, we define a regular performance measure as follows : A performance measure Z is regular if for any feasible precedence graph H, $F \in S_H$ implies that $Z(H) \leq Z(F)$. In other words, if a graph H is feasible, and if we obtain a new graph F by adding arcs to graph H, the performance measure either remains the same or deteriorates. Section 8 contains some examples of regular performance measures that can be used to generate "good" precedence graphs.

In the tree representation described in section 5.1, as a new child is formed from a node, the precedence graph associated with the child node is either the same as the precedence graph associated with the parent, or is a subset of it. If the performance measure of interest is a regular performance measure, it will either remain the same or deteriorate. Under no circumstances will it improve as one travels down a branch. This is a key property for the branch and bound tree. Note however, that regularity is not at all a restrictive requirement, since for most scenarios it is difficult to imagine a performance measure that would *improve* by adding arcs (constraints) to the precedence graph. In the next section we describe in detail the solution of the POP with minimization of the number of arcs on the graph (maximizing sparsity) as the objective.

5.3 Solving the POP with Sparsity as the Objective

In this section, we demonstrate the use of the branch and bound algorithm to obtain the sparsest possible precedence graph that satisfies all establishment conditions. The objective Z to be minimized in this case is the number of arcs on the graph. Minimizing the number of arcs has benefits from the resource allocation or scheduling perspective. A sparser graph implies that fewer activities are precedence constrained. Hence more activities are likely to be performed simultaneously, leading to better resource utilization. Also, since sparser graphs are less constrained, in general they provide greater flexibility in scheduling tasks on resources. It is obvious that the number of arcs on a graph is a regular performance measure. In order to implement the algorithm to minimize the number of arcs, we use the following bounds:

Global Upper Bound: An initial feasible solution is obtained by repeatedly applying the partitioning procedure describe in Naphade et. al. (1997). The number of arcs on that precedence graph is a global upper bound (GUB). Whenever a new feasible leaf node is reached, GUB may be updated (tightened) if the number of arcs on the new leaf node is less than the previous GUB.

Local Lower Bound: Every intermediate node represents a partially partitioned decision graph which maps on to a (probably infeasible) precedence graph. The number of arcs on this precedence graph is a lower bound of the number of arcs that will be obtained at any leaf node of which this node is an ancestor. This is obvious as the precedence graph for every child node is obtained by adding zero or more extra arcs to the precedence graph of its parent. Thus at every intermediate node, we have a local lower bound (LLB_i). If the local lower bound at an intermediate node is greater or equal to the global upper bound ($LLB_i \geq GUB$), the node is fathomed and the tree is pruned at that point.

Global Lower Bound: The establishment conditions for the problem contain several conditions without any *"or"* clauses (e.g. $A \rightarrow B$). These conditions represent precedence constraints that must be satisfied by all precedence graphs. Every such condition imposes one conjunctive precedence arc on the precedence graph. Thus the number of conjunctive establishment conditions provides a global lower bound (GLB1) on the number of arcs in a feasible precedence graph. This lower bound can be further tightened as follows. In general, a decision graph may consist of several mutually disconnected subgraphs. Every such subgraph represents a system of linked establishment conditions. To satisfy one such system, at least one arc must be imposed on the precedence graph. Since the decision subgraphs are disconnected, they are independent of one another (Proposition 4) and must each be separately satisfied. Hence if the total number of mutually disconnected subgraphs is D, then $GLB2=GLB1+D \geq GLB1$ is a tighter global lower bound.

Termination: The branch and bound algorithm terminates in either of two cases. In the first case, if all possible branches are either pruned or completely expanded, the latest updated GUB is the optimal solution. In the second case, if at a new leaf node, the tightened GUB becomes equal to GLB2 the GUB is an optimal solution.

5.4 Sparsity as a separable cost function

When sparsity is the objective, a dramatic improvement to the algorithm can be achieved by taking advantage of the fact that the number of arcs is a "separable" cost function. This is further explained as follows. While discussing the calculation of GLB2, we noted that within a given decision graph, the mutually disconnected subgraphs are

independent of one another. Each subgraph represents a set of decisions, or maps to a set of possible arcs that may be imposed on the precedence graph. The fact that the subgraphs are independent implies that these sets of precedence arcs are not only mutually disjoint but also independent, i.e. the existence of an arc from one set has no relation to the existence of an arc from another set.

If we separate all the mutually disconnected subgraphs and partition each of them optimally using the branch and bound algorithm described above, we will obtain the optimum contribution to the number of arcs from each subgraph. Due to independence, the total number of arcs will be the sum of these individual contributions plus the original conjunctive arcs. This separability property dramatically improves the performance as we will now solve much smaller problems.

Unfortunately not all performance measures are separable. For instance the longest path on the precedence graph (non-resource-constrained makespan) is a non separable cost function as it depends on the combination of arcs chosen for each subproblem and it is not possible to calculate independent contributions to makespan from each subproblem or each arc. The same is true for most other resource or scheduling based objective functions such as resource utilization, cycle times, and total tardiness.

6. Computational Insights for the POP

6.1 Object Oriented Design

The branch-and-bound algorithm was tested on several real assembly examples. In this section, we briefly describe the details of the implementation. All experiments were carried out on a personal computer with a 200 MHZ Intel pentium processor and 64 M RAM in a Windows 95 environment. The code was developed in C++ and compiled on Borland C++ v 5.01. Six classes or object types used in the code were Gstar, Node, List, Item, bb_tree , bb_node. The Gstar class represents the decision graph $\boldsymbol{G}^*$. The Node class represents nodes on the decision graph. Each Node object has associated with it a linked list (class List) of predecessors and a linked list of Successors. The class List is essentially a linked list of Items, where each Item contains an integer that is the number of the predecessor or successor node and a pointer to the next Item. The partitioning algorithm coded for Gstar essentially consists of a number of calls to functions that identify the predecessors or successors of a given node. These are coded using the breadth first search algorithm provided in Ahuja et. al. (1993). There are additional supporting functions for checking feasibility computing bounds and evaluating objective functions where appropriate. The other two classes bb_tree and bb_node represent the branch-and-bound tree and nodes in the tree. The algorithm is coded as a depth first search with backtracking using recursion, as suggested in Horowitz et. al. (1995).

6.2 Computational Results

Table 1 provides a summary of the computational results on generation of all precedence graphs for the example assemblies. Table 3 provides the results for optimization of sparsity for the same assemblies. The assemblies and the

Table 1
Computational Results for Generation of All Precedence Graphs

Problem	Parts	Tasks	N_{G*}	A_{G*}	N_{PP}	N_T	N_F	N_L	CPU Time
Ball Pen	6	6	6	4	2	5	1	2	0.0 s
F. Wheel	9	9	12	12	2	12	2	4	0.0 s
R. Wheel	10	9	18	24	0	22	0	8	0.0 s
Valve	13	13	6	4	2	5	1	2	0.0 s
AFI 1	11	18	60	56	19	1217	329	288	1.49 s
AFI 2	11	18	57	56	18	989	259	240	1.21 s
AFI 3	11	18	57	56	18	989	259	240	1.21 s
AFI 4	11	18	54	52	17	969	247	240	1.26 s

constraints for each assembly are described in detail in Naphade(1997). The notation used in the tables is as follows:

N_{G*} : Number of Nodes on decision graph
A_{G*} : Number of Arcs on decision graph
N_{PP} : Number of nodes preprocessed.
N_T : Total Number of nodes in the BCM tree
N_F : Number of nodes fathomed by sparsity cutting
N_P : Number of nodes pruned because of bound based cutting (LLB>GUB)
N_L : Number of Leaf Nodes in the tree (equal to the number of precedence graphs generated)

The table provides a comparison of the different problems in terms of problem size parameters (such as number of nodes on the decision graph) and the size of the branch-and-bound tree. In Table 1 note that we have generated over 19000 precedence graphs for the AFI. This seems like an unreasonably large number since it is known that the AFI has only 440 total sequences. (De Fazio and Whitney, 1987). This number can be somewhat understood by taking a look at the total number of precedence graphs possible for an assembly of n tasks. We are interested in N1 which is the total number of acyclic labeled weakly connected digraphs on n nodes. An exact enumeration of such graphs is an intractable problem and is not available in the current combinatorial enumeration literature, to the best of our knowledge. However a lower bound for this number is available. The number N2 of connected labeled graphs on n nodes is a lower bound of N1. This is the case, since N2 enumerates all the *non-directed* graphs. From every graph included in N2, an exponential number of acyclic directed graphs can be obtained by adding a direction to each arc. Hence N2 is a lower bound of N1. Table 2 lists the values of N2 for n nodes along with the value of n! The value of N2 has been taken from Harary and Palmer (1973). We observe that the lower bound N2 explodes enormously even for small numbers. From this perspective, the 1000 graphs obtained for AFI is not a very large number. Also not that in addition to the establishment conditions, there are other factors such as pre-processing and fathoming which have

Table 2
The number of sequences and graphs on n nodes

n	n!	N2
3	6	4
4	24	38
5	120	728
6	720	26704
7	5,040	1,866,256
8	40,320	251,548,592
9	362,880	66,296,291,072

significantly reduced the number of graphs generated. In other words, we eliminated as many dominated graphs (or graphs that are subsets of generated graphs) as possible. On the other hand, all these graphs are indeed counted towards estimating N1.

This observation uncovers a disadvantage of generating precedence graphs - there are many of them. However we see that the computation time required is quite manageable, at least the example problems studied here. This is because, even though every tree generates 800 nodes, each node on the tree requires only a linear computation.

In Table 1 we reported the results from generation of all precedence graphs. In the next set of computations we generate precedence graphs with the least sparsity. This enables us to prune a large part of the branch-and-bound tree and obtain solutions much more quickly. We note here that for this set of experiments, the lower bound GLB2 was not used (Refer

Table 3
Computational Results for Sparsity Optimization

Problem	N_T	N_F	N_P	N_L	Z^*	CPU Time
Ball Pen	5	1	0	2	1	0.05 s
F. Wheel	12	2	0	4	3	0.0 s
R. Wheel	22	0	0	8	6	0.0 s
Valve	5	1	0	2	1	0.05 s
AFI 1	839	198	96	134	9	1.1 s
AFI 2	689	153	80	116	9	0.88 s
AFI 3	689	153	80	116	9	0.88 s
AFI 4	793	175	64	160	9	1.05 s

section 5.3). Also, we did not solve the problems by separately solving subgraphs to take advantage of the separability of the cost function (Section 5.4). Rather we simply used the branch and bound method on the decision graph with depth first search. The first leaf provided the initial upper bound and the lower bound at every node was simply the number of added arcs on the precedence graph corresponding to that node (LLB). In spite of this more primitive implementation, we were still able to achieve computation times of less than 2 seconds for the largest problem solved.

6.3 The Disjunctive Paradox

We offer the following paradox related to this problem:

> Given a set of n tasks and e establishment conditions, suppose that there are s feasible sequences and p feasible precedence graphs. If an extra constraint (in the form of a disjunctive establishment condition) is added to this problem, s will decrease; however, it is also quite possible that p will increase !

For example suppose the original set of establishment conditions does not contain any constraints between tasks A,B and tasks A,C. All p precedence graphs for this problem will have no arc between A,B or between A,C. Some of the s sequences will contain A before B and/or C and some of the sequences will contain B and/or C before A. Now let us add the establishment condition B or C $\rightarrow$ A. As far as the sequence set is concerned, all sequences that contain A before both B and C, will become infeasible, reducing s. However, each graph previously containing no arc (A,B) (A,C) will now be replaced by two graphs, one with the arc A$\rightarrow$B, and one with the arc A$\rightarrow$C. This is assuming that the decisions (A,B) and (A,C) are independent and do not force any other decisions on the graph. Thus it is quite possible that a larger number of graphs is required to generate a smaller number of sequences.

7 Problem Decomposition, Redundancy and Global Optimality

7.1 Alternative Decomposition Schemes and Redundancy

In Sections 5 and 6, we discussed methods to generate all precedence graphs or to find the optimal precedence graph from *one* subproblem or decision graph. Recall (from Section 4) that when the original establishment conditions has long (length > 2) clauses, we must decompose the problem into multiple 2-SAT subproblems. The branch and bound method described in Section 5 then needs to be applied to *each* of these subproblems in order to generate a complete set of precedence graphs, or to find an optimal precedence graph. Thus, *how* one decomposes a long clause could have significant computational consequences. In Section 4, we briefly described the issue of problem decomposition and stated a simple-minded decomposition scheme. Now we are in the position to discuss this subject in much greater details.

Throughout the following discussion, we will refer to one disjunctive string of literals as an establishment condition. For example we call the constraint "(3 or 8 or 4 or 6) → 9" an establishment condition. There may also exist a different establishment condition for the same task 9, say "(3 or 7) → 9". A problem decomposition method may have two general steps as follows::

1. Split a long (length > 2) establishment condition i into several length 1 or length 2 sub-constraints.
2. Select one sub-constraint belonging to each original establishment condition *i* and add this set of constraints to the already existing length two constraints

For instance, consider the following set of establishment conditions:

(3 or 8 or 4 or 6) → 9 **(7.1)**
(3 or 8 or 5 or 6) → 9 **(7.2)**

Let us assume that these are the only establishment conditions in the problem. We explain two decomposition schemes (D1 & D2) as follows:

D1: Split (7.1) as : (3 or 8) → 9 ; (4 or 6) → 9
Split (7.2) as : (3 or 8) → 9 ; (5 or 6) → 9

D2: Split (7.1) as : (3 or 8) → 9 ; (4 or 6) → 9
Split (7.2) as : (6 or 5) → 9 ; (3 or 6) → 9

D1		D2	
Subproblem	Constraints Included	Subproblem	Constraints Included
D1S1	(3 or 8) → 9 , (3 or 8) → 9	D2S1	(3 or 8) → 9, (5 or 8) → 9
D1S2	(3 or 8) → 9 , (5 or 6) → 9	D2S2	(3 or 8) → 9, (3 or 6) → 9
D1S3	(4 or 6) → 9 , (3 or 8) → 9	D2S3	(4 or 6) → 9, (5 or 8) → 9
D1S4	(4 or 6) → 9 , (5 or 6) → 9	D2S4	(4 or 6) → 9, (3 or 6) → 9

Table 4
Alternative Decomposition Schemes

Table 4 lists the subproblems that one would obtain using these schemes. The first concept we introduce is that of subproblem elimination. Compare problems D1S1 and D1S2. Recall that both these subproblems will produce precedence graphs which are feasible. Observing D1S1, we may conclude that the constraint (3 or 8) → 9 by itself is sufficient to guarantee feasibility. Thus the graphs generated by D1S2 will be overly constrained. According to the definition of graph subsets in section 4.2, every graph generated by solving D1S2 will be a subset of some graph generated by solving D1S1. We are not interested in such subset graphs because they do not produce any new sequences, and they do no better than their superset graphs for regular performance measures. Hence we can eliminate D1S2 from consideration. We call this process subproblem elimination. Note that D1S3 will be eliminated for the same reason. Thus if we use decomposition scheme D1, we will need to solve only two subproblems, D1S1 and D4S4.

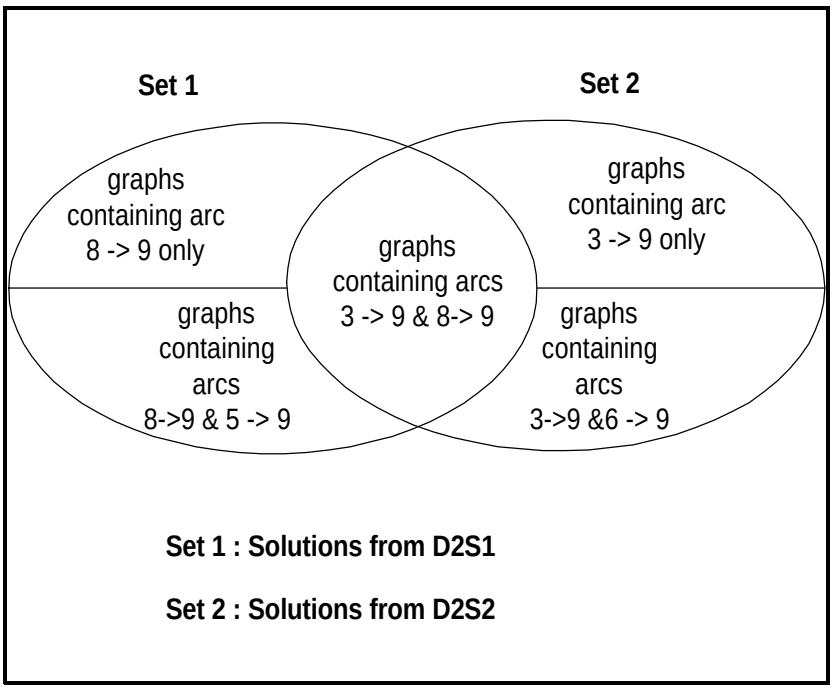

Figure 5
Redundancy

On the other hand we note that such subproblem elimination is not possible in D2. If we compare any pair of subproblems in D2, we find that one of them will produce at least one precedence graph which is not a subset of any precedence graph produced by the other. So D1 is a preferable decomposition scheme because we need solve only two subproblems as against four in D2.

A second undesirable property exhibited by D2 is the existence of redundancy between subproblems. We say that there is redundancy between two subproblems when they produce some common precedence graphs or one of them produces some precedence graphs that are subsets of those produced by the other. According to this definition, there is a redundancy in D1 between D1S2 and D1S1. However because this is a total redundancy as explained above, D1S2 can be completely eliminated. There is obviously no redundancy between D1S1 and D1S4. The same is not true for decomposition scheme D2. We demonstrate the redundancy in D2 in Figure 5. The graphs in the intersection of set S1 and Set S2 are generated by both the problems. If this redundancy was total as in D1, it could be eliminated. However D2S1 generates some precedence graphs that are not generated by D2S2 (set S1-S1∩S2) , and vice versa (S2 - S1∩S2). So to obtain a complete solution, one must solve both S1 and S2. This implies that some computational effort is wasted in generating the same precedence graphs twice.

Several questions regarding redundancy arise that we now pose - What is the cause of redundancy? Does the possibility of redundancy always exist? If not, under what conditions does redundancy never exist ? If the possibility of redundancy exists, is it possible to obtain a redundancy-free decomposition for any problem ? We address these questions in the next subsection.

7.2 Redundancy-free Decomposition Schemes

Clearly it is desirable to develop decomposition schemes that are free from redundancy . In this section we start our

exploration by first investigating the reasons for redundancy.

Proposition 5
Redundancy exists only if the following necessary conditions are true:

1. There is at least one long (length > 2) establishment condition.
2. There are at least two establishment conditions for the same task, at least one of which is long, and they share some common literals.

Proof:

The reason for the first condition is obvious. For the second condition, let us now define commonality. We say that there is commonality between two constraints if the same arc or set of arcs can satisfy both constraints, e.g. the constraints (A or B or C) → Z and (A or B or D) → Z have a commonality of two, as either of the two arcs : A → Z and B → Z satisfy both the constraints. Let us now turn our attention to condition 2 above which guarantees the non-existence of redundancy in the absence of commonality.

Let us first assume that there is only one establishment condition for a task Z and it is long. Let the condition be

(A or B or C or D) → Z. **(7.3)**

If we decompose the condition as follows :

D3: Split establishment condition as (A or B) → Z ; (C or D) → Z

D3S1 : (A or B) → Z
D3S2: (C or D) → Z

There is no redundancy above as D3S1 will have either A → Z or B → Z and D3S2 will have either C → Z or D → Z. Thus it is clear that at least two conditions for the same task will be needed for redundancy to exist.

Similarly it is easy to see that there must be commonality in the establishment conditions for redundancy to exist. Consider the conditions below:

(A or B or C) → Z **(7.4)**
(D or E or F) → Z **(7.5)**

and the following decomposition scheme :

D4 : Split 7.4 As (A OR B) → Z ; C → Z
Split 7.5 As (D OR E) → Z ; F → Z

D4S1 (A OR B) → Z, (D OR E) → Z
D4S2 (A OR B) → Z, F → Z
D4S3 C → Z ; (D OR E) → Z
D4S4 C → Z ; F → Z

Since there is no commonality between constraints, every precedence graph generated by each subproblem contains at least one arc that is not imposed by any other subproblem and vice-versa.

It is also true that no subproblem will generate a precedence graph that is a subset of a precedence graph generated by another problem. Thus there will be no redundancy. This is true regardless of the length and number of long constraints. We have thus proven that

For a given problem instance, there is no redundancy resulting from decomposition unless

(1) There are at least two establishment conditions for the same task

(2) At least one of them is long

(3) There is commonality of literals between the two conditions.

The above gives us a sufficient condition for a redundancy free decomposition. However it is not a necessary condition. In other words, because of the possibility of subproblem elimination, one may be able to obtain redundancy free decompositions even if the above three conditions exist. An example of this was demonstrated in Table 4.

Proposition 6

If there are one or more pairs of establishment conditions with commonality, and there is no commonality in between pairs, there is always a redundancy free decomposition.

Proof:

To prove this, we use the tool of intermediate decomposition

Suppose two establishment conditions are as follows :

$(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c \text{ or } B_1 \text{ or } B_2 \text{ or } .... \text{ or } B_m) \rightarrow Z$ **(7.6)**

$(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c \text{ or } C_1 \text{ or } C_2 \text{ or } .... \text{ or } C_n) \rightarrow Z$ **(7.7)**

The following intermediate decomposition DI suggests itself :

DI : Split 7.6 as $(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c) \rightarrow Z$
$(B_1 \text{ or } B_2 \text{ or } .... \text{ or } B_m) \rightarrow Z$

Split 7.7 as $(A_1 \text{ or } A_2 \text{ or } ..... \text{ o1r } A_c) \rightarrow Z$
$(C_1 \text{ or } C_2 \text{ or } .... \text{ or } C_n) \rightarrow Z$

DIS1 $(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c) \rightarrow Z$; $(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c) \rightarrow Z$

DIS2 $(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c) \rightarrow Z$; $(C_1 \text{ or } C_2 \text{ or } .... \text{ or } C_n) \rightarrow Z$

DIS3 $(B_1 \text{ or } B_2 \text{ or } .... \text{ or } B_u) \rightarrow Z$; $(A_1 \text{ or } A_2 \text{ or } ..... \text{ or } A_c) \rightarrow Z$

DIS4 $(B_1 \text{ or } B_2 \text{ or } .... \text{ or } B_u) \rightarrow Z$; $(C_1 \text{ or } C_2 \text{ or } .... \text{ or } C_n) \rightarrow Z$

By the process of subproblem elimination, we eliminate DIS2 and DIS3. It is easy to see that the remaining intermediate problems DIS1 and DIS2 are redundancy free according to Proposition 5. They will further be decomposed into subproblems which will also be redundancy free. It is also fairly obvious that the above proposition extends to the case where there exist several pairs of establishment conditions with commonality within the pair, but no commonality in between pairs.

The next logical question is what happens if there are three (or more) establishment conditions with mutual commonality and at least one among them is long. Consider the following example :

$A_1 \text{ or } A_2 \text{ or } A_3 \rightarrow Z$ **(7.8)**

$A_1 \text{ or } A_2 \rightarrow Z$ **(7.9)**

$A_2 \text{ or } A_3 \rightarrow Z$ **(7.10)**

In the above example, the condition (7.8) can be split in three ways :

1. $A_1 \text{ or } A_2 \rightarrow Z$ $A_3 \rightarrow Z$
2. $A_2 \text{ or } A_3 \rightarrow Z$ $A_1 \rightarrow Z$
3. $A_1 \text{ or } A_3 \rightarrow Z$ $A_2 \rightarrow Z$

This leads us to three decomposition schemes, each with two subproblems. These are listed as follows after constraint elimination:

Decomposition I (D1)

D1S1: A_1 or $A_2 \rightarrow Z$, A_2 or $A_3 \rightarrow Z$

D1S2: $A_3 \rightarrow Z$, A_1 or $A_2 \rightarrow Z$

Decomposition II (D2)

D2S1: A_2 or $A_3 \rightarrow Z$, A_1 or $A_2 \rightarrow Z$

D2S2: $A_1 \rightarrow Z$, A_2 or $A_3 \rightarrow Z$

Decomposition III (D3)

D2S1: A_1 or $A_3 \rightarrow Z$, A_2 or $A_3 \rightarrow Z$, A_1 or $A_2 \rightarrow Z$

D2S2: $A_2 \rightarrow Z$

It is quite clear that each of the above decompositions lead to redundancy along the same lines as explained in Figure 5. Thus for this problem there is no decomposition that is redundancy free. This is related to the fact that there is a different common set of literals between any two of the three conditions.

Proposition 7

For the general case of n establishment conditions with commonality ($n \geq 3$), a redundancy free decomposition scheme may not be obtainable.

The above example constitutes a proof by counter example for Proposition 7.

From the literature, (cf. Ben-Arieh and Kramer(1994), Nevins and Whitney (1989), Li and Hwang(1992a), De Fazio and Whitney (1987)) it seems that for most real world examples, all establishment conditions have disjunctions of two or less than two literals. Among examples that have disjunctions of three or more literals, not all have commonality. So the question of redundancy may arise only in rare cases.

7.3 Level I Decomposition

Throughout the above discussion, we decomposed problems with long establishment conditions into subproblems with establishment conditions of length two, with one length 1 establishment condition if the original long condition had an odd number of literals. Let us call such a decomposition, a Level II decomposition. For such a decomposition, we investigated the issues of redundancy and subproblem elimination.

Another option for decomposition is to split all establishment conditions into length one conditions and then form the subproblems. We call this a Level I decomposition. There are two advantages of doing this. First, the resulting subproblems are always redundancy free (as we shall shortly prove). Second, there is only one Level I decomposition for any subproblem and hence no choice needs to be made among decomposition schemes. The disadvantage is that we are left with a significantly higher number of subproblems to solve.

Consider the example in the previous section (establishment conditions 7.8 - 7.10). The Level I Decomposition DL1 is as follows:

DL1: Split 7.8 as $A_1 \rightarrow Z$; $A_2 \rightarrow Z$; $A_3 \rightarrow Z$

Split 7.9 as $A_1 \rightarrow Z$; $A_2 \rightarrow Z$; $A_4 \rightarrow Z$; $A_5 \rightarrow Z$

Split 7.10 as $A_2 \rightarrow Z$; $A_3 \rightarrow Z$; $A_5 \rightarrow Z$;

This leads to 36 subproblems. We list the first 6:

DL1S1: $A_1 \rightarrow Z$, $A_1 \rightarrow Z$, $A_2 \rightarrow Z$

DL1S2: $A_1 \rightarrow Z$, $A_1 \rightarrow Z$, $A_3 \rightarrow Z$

DL1S3: $A_1 \rightarrow Z$, $A_1 \rightarrow Z$, $A_5 \rightarrow Z$
DL1S4: $A_1 \rightarrow Z$, $A_2 \rightarrow Z$, $A_2 \rightarrow Z$
DL1S5: $A_1 \rightarrow Z$, $A_2 \rightarrow Z$, $A_3 \rightarrow Z$
DL1S6: $A_1 \rightarrow Z$, $A_2 \rightarrow Z$, $A_5 \rightarrow Z$

DL1S4 through DL1S6 can be eliminated because of subproblem elimination. In the remaining subproblems, there is no redundancy.

It is easy to see that this is true for the entire decomposition. The reason is that a Level I decomposition decomposes literals into conjunctive constraints. Hence the decomposition enumerates all possible solutions to the long constraints. If two solutions are identical, or one is overly constrained compared to another, subproblem elimination will eliminate one of them. Eventually, the process of subproblem elimination will completely eliminate all redundant solutions. Thus the following proposition can be stated.

Proposition 8 A Level I decomposition is always redundancy free

The disadvantage of a Level I decomposition, however, is a large number of subproblems.

We have seen that a Level II decomposition may not necessarily result in a redundancy free decomposition, but yields a smaller number of subproblems when compared to a Level I decomposition. There may be intermediate decomposition methods (decomposition of some establishment conditions in a level I fashion and others in a level II fashion) that yield redundancy free decompositions without producing as many subproblems as a level I decomposition does.

7.4 Decomposition and Global Optimality

A very important issue associated with decomposition and solving the POP is that of global optimality. Consider the following problem :

$$A_1 \text{ or } A_2 \text{ or } A_3 \rightarrow Z \qquad \textbf{(7.11)}$$
$$A_1 \text{ or } A_2 \text{ or } A_4 \rightarrow Z \qquad \textbf{(7.12)}$$

Two candidate decomposition schemes for the above problem are :

D5: Split 7.11 as $A_1 \text{ or } A_2 \rightarrow Z$; $A_3 \rightarrow Z$
Split 7.12 as $A_1 \text{ or } A_2 \rightarrow Z$; $A_4 \rightarrow Z$

After subproblem elimination, the remaining problems are :

D5S1: $A_1 \text{ or } A_2 \rightarrow Z$
D5S2: $A_3 \rightarrow Z$, $A_4 \rightarrow Z$

D6: Split 7.11 as $A_1 \rightarrow Z$; $A_2 \text{ or } A_3 \rightarrow Z$
Split 7.12 as $A_1 \text{ or } A_2 \rightarrow Z$; $A_4 \rightarrow Z$

After subproblem elimination, the remaining problems are :

D6S1: $A_1 \rightarrow Z$
D6S2: $A_2 \text{ or } A_3 \rightarrow Z$, $A_1 \text{ or } A_2 \rightarrow Z$
D6S3: $A_2 \text{ or } A_3 \rightarrow Z$, $A_4 \rightarrow Z$

Now let us compare the two decomposition schemes D5 and D7. We see that D6S2 may generate graphs containing both the arcs $A_3 \rightarrow Z$ and $A_1 \rightarrow Z$. None of the subproblems of D5 will generate such a precedence graph.

Thus, even if each decomposition scheme generates a complete set of precedence graphs from the point of view of sequence generation, a given decomposition scheme will not generate *all* possible precedence graphs. This implies that if we are trying to find the "optimal" precedence graph to solve a certain POP, we must, in the worst case, enumerate all possible decompositions of the problem to ensure that all precedence graphs have been included in the solution space. This leads to an exponential number of decompositions, each having an exponential number of subproblems that need to be solved. This will result in an unrealistically large computational effort and also the possibility of large amounts of redundancy *between decomposition schemes* as well as between subproblems of bad decomposition schemes.

More research needs to done on comparison of level I decompositions and different decomposition schemes from the perspective of global optimality, redundancy and computational effort. It may be possible to find conditions that prove dominance of certain decomposition schemes over others for certain objective functions of the POP. Such conditions will allow selection of a decomposition scheme which will generate a dominating set of precedence graphs thus retaining the capability of obtaining the globally optimal solution to the POP.

8 Future Research Issues

There are several avenues for future research from this work and several issues that need to be further investigated. These are as follows :

1. **Sequence based objective functions:** In this work we developed a method for optimizing graph based objective functions. However most problems of real interest are sequence based. For example the assembly line balancing problem and the resource constrained project scheduling problem are classic problems of relevance respectively to assembly lines and 1-of assembly projects. However since one precedence graph yields a multitude of sequences, the ALB and RCPS problems are SOPs and not POPs. So the methods developed here cannot be directly used for integrating assembly sequence generation and line balancing. In Naphade (1997) we provide methods to solve both the SOP and POP for sequence based objective functions such as the ALB and RCPS performance criteria. For sequence based objective functions, the POP is defined as identification of a precedence graph or graphs that *contain* the optimal sequence.

2. **Decomposition Issues:** Issues related to decomposition need to be further investigated. The most important issue is one of global optimality. We demonstrated in Section 7.4 that in order to achieve global optimality for *graph based* performance measures in the presence of decomposition, one may need to enumerate all decomposition schemes, which is an enormous and unrealistically expensive computational effort. In order to capture the global optimum without this huge effort, we need to investigate the possibility of certain decomposition schemes dominating others for certain objective functions.

   Another set of issues related to decomposition are those of redundancy. We proved in Section 7.2 that for the general case, there are no decomposition schemes that yield redundancy free Level II decompositions. We demonstrated certain special cases for which this was not true. More research needs to go into the possible existence of other special cases where redundancy free Level II decompositions are possible.

   We noted that there is a unique Level I decomposition for every problem which is redundancy free. However the number of subproblems generated is significantly larger. A computational study of the tradeoffs between these two decompositions needs to be done. Such a study may evaluate the differences between solving a smaller number of problems with redundancy and a larger number of smaller subproblems without redundancy.

3. **Other graph-based performance measures:** Here we discuss performance measures other than sparsity that can be used to select precedence graphs.

The merits of Sparsity: We mentioned earlier that sparsity is beneficial since it implies a less constrained precedence graph, more scheduling flexibility etc. However, the above statements are only "generally true", in that a sparser graph is not necessarily a better one. An example where sparsity does not benefit, is shown in Figure 7. Graph 1 has six arcs, but tasks A, B, C can all be performed simultaneously if resources permit. On the other hand, graph 2 has only four arcs, but the tasks A, B, C cannot be performed simultaneously. (A path on n nodes is the sparsest connected graph, but it is also the graph which is most heavily constrained and has least scheduling flexibility).

Thus sparsity is not always beneficial even if a sparse graph will "generally" be better (less constrained) than a dense graph. We described the application of our method to maximize sparsity as a demonstration of its computational efficiency. However the method can also be used for other regular performance measures such as the ones described below:

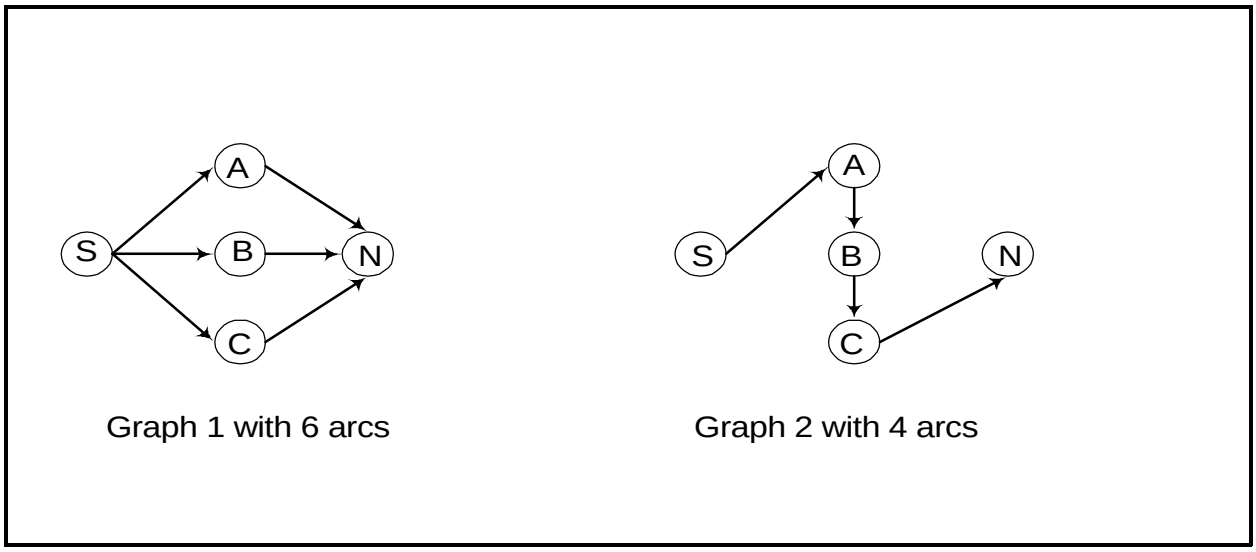

Figure 6
Sparsity is not always beneficial

a) **Transitivity Adjusted Sparsity (TAS):** Suppose, conjunctive arcs exist between nodes A,B and A,C, and suppose the decision graph partitioning process imposes an arc from B to C. In this case, the resultant graph has three arcs : A→B, B→C, A→C. Thus to optimize S, we will try to minimize this number (N). However, note that because of transitivity, the constraint A→C is not necessary, and the "effective" transitivity-adjusted number of arcs N_t is 2. Thus optimizing N_t makes more sense than optimizing N.

b) **Transitivity Adjusted Node Disjoint Paths (TANDP):** A higher number of node disjoint paths in a graph again indicates that there are many activities that can be done in parallel. (Figure 7). Transitivity adjustments are necessary for a more meaningful evaluation.

The graphs in Figures 7 and 8 both have the same number of TANDP, but the graph in 8 is clearly more constrained than the one in 7. This leads us to consider joint optimization of TANDP and TAS.

c) **Weighted TAS & TANDP :** Intuitively, this tries to remove the arcs that are not on the node disjoint (parallel) paths on the graph in Figure 8, allowing any two activities on different node disjoint paths to be executed simultaneously.

d) **Average Nodal Indegree (ANI):** If the average nodal indegree is large, one can say that roughly, an activity has many predecessors. Thus an activity will have to wait for many other activities to be completed, thus minimizing ANI may be beneficial.

With the same rationale, we may want to maximize the Average Nodal Outdegree (ANO). However maximization of ANO is not a regular performance measure, and the methods developed here cannot be used to optimize it.

e) **Average Total Number of Predecessors (ATP):** This measure extends the idea behind ANI to include all predecessors of an activity instead of just the immediate predecessors.

4. **Optimization in the presence of Randomness**: In many cases, the actual execution of an assembly schedule may be subject to randomness in the form of random processing times, breakdowns of machines and so forth. In such cases the robustness of an assembly plan (or schedule) is more important than the deterministic objective function. Thus one direction for future research is generation of assembly precedence graphs that are robust - for which the cost function does not deteriorate greatly in the presence of randomness. A related problem is identification of measures based on the structure of the graph that can predict or are correlated to the robustness of the graph. A specific avenue for research on this problem is to use our method to generate a set of graphs that optimize deterministic measures such as the ones listed above and then use simulation to test the performance of these graphs in the presence of randomness.

9. Contributions, Strengths and Weaknesses.

The methods developed in this paper have the following shortcomings:

1. The starting point is a set of Establishment Conditions. This set of establishment conditions can be arbitrarily complex for large products or subassemblies. The more complex the establishment conditions, the more difficult it is for engineers to extract them accurately based on the product design. Also, the methods developed in this paper are especially applicable to complex and challenging sets of establishment conditions. More research needs to be done on automatic generation of establishment conditions or assembly constraints, so that the extraction of complex establishment conditions becomes easier. Some such research can be found in Delchambre and Wafflard (1991).

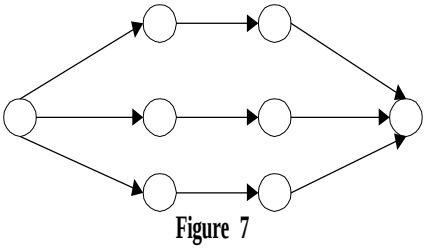

Figure 7
Node Disjoint Paths

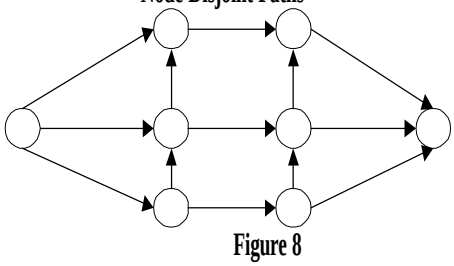

Figure 8
What's wrong with NDP

2. There is an inherent disadvantage to generating precedence graphs instead of generating sequences, due to the fact that the number of precedence graphs is typically a few orders of magnitude greater than the number of sequences. This indicates that one sequence may be generated from many different precedence graphs. However we believe that the computational results demonstrated in this work present a convincing argument for the use of precedence graphs, since for moderately sized problems commonly used in the literature, the computations have been of the order of seconds.

We think that the above drawbacks are more than compensated for by the several contributions and strengths of the methods described below:

1. **Rigorous Optimization Methodology:** The first contribution is the framework itself. To the best of our knowledge, this is the first rigorous methodology that optimally solves the precedence selection problem. This framework can be used by other researchers to develop search algorithms in order to select assembly plans based on quantitative measures that are of relevance to their specific process design or manufacturing scenario.

2. **Computational Efficiency:** We demonstrated that inspite of dealing with a much larger solution space of precedence graphs, our method optimized sparsity within an extremely small computational expense.

3. **Simultaneous connections between more than two parts:** Since we base our analysis on tasks as against parts, our method is applicable to liaisons between any number of parts. If an assembly exists in which connecting more than two parts together seems to be a natural task, that task can be easily represented as a node on a precedence graph. Also, another way in which three parts (e.g. a, b, c) may be connected together, is by simultaneously establishing the liaisons between a , b and b , c. This is equivalent to parallel performance of two tasks modeled in the precedence graphs as nodes - which is quite obviously being incorporated in our method.

4. **Improvements over existing methods for the POP**: There have been several attempts at automating not only the generation but also the selection of assembly plans in the literature. We believe that this paper provides the following contributions in relation to previous literature:

   (i) Chen and Henrioud (1994) outline a method for systematic generation of precedence graphs. However they provide no complexity analysis or computational results.

   (ii) Bonneville et. al. (1995) use a genetic algorithm for integrated generation and evaluation of assembly plans. However they start off with a few feasible assembly plans provided by an expert and generate new plans by identifying common subassemblies within the plans that they have. This method is not a complete plan generation method as it is limited by the subassemblies that exist in the initial set of plans. For this reason and also because the solution methodology is a GA, this method cannot guarantee optimality.

   (iii) Delchambre and Gaspart (1992) develop a prototype user friendly software for generation and evaluation of assembly plans based on the constraint extraction method of Delchambre and Wafflard (1991). However their method does not rigorously deal with disjunctive constraints, nor does it use quantitative methods for selection of Assembly Plans. They develop an interface through which the user can view several assembly plans and evaluate them based on quantitative and/or subjective measures.

References

1. Ahuja R.K., T.L. Magnanti and James B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, NJ. 1993
2. Baldwin D.F., T.E. Abell, M. M. Lui and T. L. DeFazio, "An Integrated Computer Aid for Generating and Evaluating Assembly Sequences for Mechanical Products", *IEEE Jr. of Rob. and Automation*, 7(1), 78-94, 1991.
3. Ben-Arieh D. and B. Kramer, "Computer-Aided Process Planning for assembly: generation of assembly operations sequence",*Int. Jr. of Prod. Res.*, 32(3), 643-656, 1994.
4. Bonneville F., C. Perrard and J. M. Henrioud, "Genetic Algorithm to generate and evaluate assembly plans", *IEEE Symposium on Emerging Technologies and Factory Automation*, 2, 231-239, 1995.
5. Bourjalt A., "Contribution a une approche méthodologique de l'assemblage automatisé : Elaboration automatique de séquences opératoires," Thése d'état, Université de Franche-Comté, Besançon, France, Nov. 1984.
6. Chen Ke and Jean-Michel Henrioud, "Systematic Generation of Assembly Precedence Graphs", *Proc. of the IEEE Int. Conf. on Rob. and Auto.*, 1476-1482, 1994.
7. DeFazio T.L. and D. E. Whitney, "Simplified Generation of All Mechanical Assembly Sequences",*IEEE Jr. of Rob. and Auto.*, 3(6), 640 - 657, 1987
8. Delchambre A., *Computer-Aided Assembly Planning*, Chapman and Hall, London, 1992.
9. Delchambre A. and P. Gaspart, "KBAP: An Industrial Prototype of Knowledge-Based Assembly Planner", *Proc. of the IEEE Int. Conf. on Rob. and Auto*, , 3, 2404-2409, 1992.
10. Harary Frank and Edgar M. Palmer, *Graphical enumeration*, Academic Press, New York, 1973.
11. Homem de Mello Luiz S., and Arthur C. Sanderson, "AND/OR Graph Representation of Assembly Plans", *IEEE Trans. on Rob. and Auto*, 6(2), 188 - 199, 1990.
12. Homem de Mello Luiz S., and Arthur C. Sanderson, "Representations of Mechanical Assembly Sequences",*IEEE Trans. on Rob. and Auto*, 7(2), 211-227, 1991a.
13. Homem de Mello Luiz S., and Arthur C. Sanderson, "A Correct and Complete Algorithm for the generation of Mechanical Assembly Sequences",*IEEE Trans. on Rob. and Auto*, 7(2), 228-240, 1991b.
14. Homem de Mello Luiz S., and Arthur C. Sanderson,"Two Criteria for the Selection of Assembly Plans : maximizing the Flexibility of Sequencing the Assembly Tasks and Minimizing the Assembly Time Through Parallel Execution of Assembly Tasks", *IEEE Trans. on Rob. and Auto*, 7(5), 626-633, 1991c.
15. Horowitz E., S. Sahni and D. Mehta, *Fundamentals of Data Structures in C++*, W.H. Freeman and Co., New York, 1995.
16. Laperriere Luc and Hoda A. ElMaraghy, "Assembly Sequences Planning for Simultaneous Engineering Applications", *The International Journal of Advanced Manufacturing Technology*, 9(4) 231-244, 1994.
17. Lee Jim and Tzvi Raz, "A Branch-and Bound Procedure for Robot Assembly Planning", *Proceedings of the 11th annual conference on Computers and Industrial Engineering*, 17, 215-220, 1989
18. Li R. and C. Hwang, "Framework for automatic DFA system development.", *Comp & IE*, 22(4), 403-411, 1992a.
19. Minzu V. and J. M. Henrioud, "Systematic Method for the Design of Flexible Assembly Systems", *Proceedings of the IEEE International Conference on Robotics and Automation*, 3, 56-62, 1993.
20. Naphade K.S., *A Graph Theoretic Framework for Integrated Assembly Planning*, Ph.D. Dissertation, Department of Industrial and Manufacturing Systems Engineering, Lehigh University, June 1997.
21. Naphade K.S., S.D. Wu and R.H. Storer, "Graph-Theoretic Generation of Assembly Plans, Part I: Correct Generation of Precedence Graphs", (accompanied paper), 1999.
22. Nevins J.L. and D.E. Whitney, *Concurrent Design of Products and Processes*, McGraw-Hill Publishing Company, USA, 1989.
23. Nilsson N.J., *Principles of Artificial Intelligence*, Morgan Kaufman Publishers Inc., USA, 1980.
24. Papadimitrou C.H. and K. Steiglitz, *Combinatorial Optimization : Algorithms and Complexity*, Englewood Cliffs NJ : Prentice Hall USA, 1982.
25. Steen. S.W.P., *Mathematical Logic*, Cambridge University Press, New York, 1972.
26. Wu S.D., E. Byeon and R.H. Storer, "A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness", *Operations Research*, Vol. 47, No. 1, 113-124, 1999.