Graph-Theoretic Generation of Assembly Plans
Part I: Correct Generation of Precedence Graphs

Kedar S. Naphade
Bell Labs, Lucent Technologies

Robert H. Storer
S. David Wu
Lehigh University

Report No. 99T-03

Graph-Theoretic Generation of Assembly Plans
Part I: Correct Generation of Precedence Graphs

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

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

Abstract

Automatic generation and selection of assembly plans is a problem of great practical importance with many significant cost implications. In this paper, we focus on the constraint satisfaction aspects of this problem which involve the determination of a correct assembly precedence graph from an arbitrarily complex set of design constraints, physical restrictions and pre-established design conditions. We prove that this constraint satisfaction problem (CSP) is NP-complete and provide a graph-theoretic scheme for the generation of assembly plans. Our scheme involves decomposing the CSP into polynomially solvable subproblems, mapping the subproblems onto a "decision graph," then partitioning the decision graph to obtain correct assembly plans. We then describe a procedure that systematically generates all assembly sequences. In this paper, we focus on key theoretical properties of the proposed framework including its correctness, completeness and inherent computational complexity. To illustrate main elements of the framework we include a simple example in bicycle assembly. In a subsequent (Part II) paper, we discuss computational issues related to decomposition as well as the generation of *optimal* assembly plans, where optimality is defined in terms of sequencing and scheduling objectives, line balancing metrics, or other graph computable measures.

1. Motivations

Assembly is the process of joining separate components and subassemblies together to form a single final assembled unit (e.g. mechanism, device, building etc.). A single assembly task involves joining two or more components or subassemblies together. In many cases, the order in which these tasks are performed is an important consideration. Firstly, many such orders may not be feasible because of physical constraints such as accessibility and stability of assembly. For example, while changing a car tire, one cannot feasibly install the lug nuts until after the tire has been mounted on to the hub. It is also common that many feasible sequences exist, but some are more desirable than others according to criteria such as the need for jigs or fixtures, the number of tasks that can be performed simultaneously and so forth. We define assembly planning as the process of identifying an assembly plan, which defines either a complete or partial order in which the assembly tasks can be performed.

Assembly Planning problems have received much 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. The sequence generation problem involves generating one or more feasible sequences. These are sequences in which all the tasks in this set can be performed in order to feasibly assemble the product. The precedence graph generation problem (Chen and Henrioud, 1994), involves generating precedence graphs such that all assembly sequences generated from these graphs are feasible.

The sequence in which assembly tasks are performed can have a significant impact on cost and efficiency through both quantitative and qualitative measures. Quantitative measures may include resource allocation, line balancing objectives, scheduling objectives, the number of re-orientations required and the 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. Different assembly sequences may require different resources, may lead to different resource utilizations and different tooling and fixturing requirements.

The assembly planning process is typically performed by industrial or manufacturing engineers. Typically, the decisions made are subjective and depend heavily on the expertise and experience of the person involved. Though most experienced engineers devise good sequences intuitively, much could be gained by the systematic generation of sequences. Systematic generation ensures that no good sequences are overlooked, facilitates automation of the assembly planning process and provides a means to evaluate alternatives which better utilize resources in a flexible environment (Homem de Mello and Sanderson, 1991b).

In this paper, we develop a method which instead of generating fully specified assembly sequences, generates a set of correct and complete precedence graphs. The word *complete* refers to the generation of a set of precedence graphs from which *all* possible assembly sequences can be derived. The word *correct* implies that all these sequences are *feasible*, i.e. they satisfy all assembly constraints.

We believe that the precedence graph has greater merit as an assembly "plan" than a fully specified single sequence, for the following reasons:

1. In many cases, it is possible to perform several assembly tasks simultaneously. In order to take advantage of this, specification of a partial order (not a complete order) of tasks that satisfies all stated assembly constraints is needed. This partial order is represented by a precedence graph. By specifying a full sequence we lose the opportunity for simultaneous execution of tasks.

2. In cases requiring dynamic decision making during actual assembly operations, a precedence graph is a much more valuable plan than just a sequence. We describe two cases where such decision making is required:

   (i) Consider a flexible assembly shop or a flexible job shop where one task can be performed on several different machines. In addition to dispatching decisions, such a shop also introduces routing decisions into the problem. Problems like the job shop scheduling problem are extremely difficult even in the absence of such flexibility - solving them optimally in the presence of flexibility is nearly impossible. Hence they may be solved using heuristic routing rules and dispatching rules in real time. We claim that in such scenarios, the precedence graph is much more valuable as an assembly plan than a single assembly sequence. This is because at any decision point, the precedence graph supplies a set of schedulable tasks as against just the single task that a sequence would supply. This argument is mirrored by Homem de Mello and Sanderson (1991b) where they mention that one of the benefits of generating all sequences is that it provides alternate sequences for better resource utilization in a flexible environment.

   (ii) Consider the common situation in which there is significant uncertainty related to either task durations or resource availabilities. The precedence graph is again a much more valuable "plan". This is because a precedence graph represents in a compact fashion, a large number of possible assembly task sequences. As a result, in the event of a disruption, it is easily possible to switch to a different sequence if the shop floor controller has access to the precedence graph (Wu et al., 1999). Consider the following scenario as an example. Task i is the task supposed to be performed next and the machine m that task i requires, breaks down. If the shop floor controller only knows one feasible sequence, he or she has to wait until the machine is up again. On the other hand, if he or she has access to the precedence graph, several other tasks may be executed while machine m is being repaired.

3. Chen and Henrioud (1994) mention that in order to design an assembly line, a precedence graph is first required. Generation of this precedence graph is handled empirically in most factories, which may be a viable option if the new product is to be manufactured by modifying an existing line or if the new product is simple. However for complex new products, generation of all precedence graphs or at least a set of good precedence graphs is extremely useful information for assembly line design.

4. In many real assembly environments, quantitative assembly line balancing methods are still not being used because it is difficult to come up with *one* precedence graph - different design engineers have different perspectives of the assembly product and come up with different precedence graphs. Here we present a framework that can be used for generating one correct precedence graph or all precedence graphs.

In this paper, we are primarily concerned with the generation of a feasible or correct assembly plan rather than the selection of an "optimal" one. Methods for complete generation and selection of assembly plans are discussed in the subsequent (Part II) paper (Naphade, et. al. 1999). The remainder of the paper is organized as follows: In Section 2, we provide a brief review of the relevant assembly planning literature. In Section 3, we define and characterize the problem of interest. Sections 4,5,6 each discuss various steps of the methodology developed. Section 7 contains an analysis of the computational complexity. Section 8 contains an illustrative example. In Section 9, we conclude the paper with a discussion of the shortcomings and contributions of this work and directions for future research.

2. Review of Literature

Bourjalt(1984) started research in assembly sequence generation. He obtained the constraints or establishment conditions providing the relationships between liaisons through a question and answer method. DeFazio and Whitney(1987) improved Bourjalt's method by reducing the number of questions needed. They devised a "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. This representation required fewer nodes compared to the diamond graph of DeFazio and Whitney (1987) and simplified the search for feasible plans. This representation essentially evaluates the possibility of a transition from a certain assembly to all possible sets of two or more feasible child subassemblies. A similar approach is also provided by Lee et. al. (1993). On this graph, a tree represents one assembly sequence. Homem de Mello and Sanderson (1991b) further present an algorithm for correct and complete generation of assembly sequences using AND/OR graphs. Tsao and Wolter (1993) and Huang and Lee (1991) use predicate calculus methods for assembly sequence generation.

In our opinion, the challenge of assembly planning arises out of the presence of disjunctive constraints ("or" constraints) and the consequent need of predicate calculus to model the constraints. An example of an "or" constraint is "1 or 2 → 3" where 1 ,2 and 3 are assembly tasks. This constraint implies that either task 1 or task 2 *must* be performed before task 3 can be performed. For many simple assemblies, there may be no "or" constraints. There is a significant volume of literature that concentrates on automated sequence generation for assemblies without "or" constraints. The important contributions of such work (e.g. Ben-Arieh and Kramer (1994), Li and Hwang (1992a, 1992b), Lin and Chang (1993) etc.) lie in computer-aided constraint identification and computer-aided sequence generation methods using constraint based modeling, or geometric feature based modeling of the assembly. There has also been a lot of research on designing computer-aided assembly planning systems (expert systems or decision support systems). For instance Ye and Urzi (1996) try to capture the heuristic strategies used by design engineers to generate good assembly plans in their decision support system. Delchambre (1992) give an extensive review of techniques used in such systems. A more recent review can be found in Ye and Urzi (1996). Delchambre and Wafflard (1991) develop software that extracts precedence constraints from a liaison graph. Delchambre and Gaspart (1992) use this method to develop prototype software for generation and selection of assembly plans. They however do not consider the case of complex disjunctive constraints. Also selection of plans is manually done by a user through a user-friendly interface, as opposed to a systematic implicit/explicit enumeration for quantitative selection of assembly plans.

We assume that the establishment conditions which are input for our method are already available. Generation of these conditions has been dealt with in Baldwin et. al. (1991), Bourjalt (1984), DeFazio and Whitney (1987). 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.

3. Problem Definition and Characterization

We first define the constraint satisfaction problem or CSP. The CSP determines a "correct" precedence graph - a precedence graph given a specified set of assembly constraints. Note that given a correct precedence graph, it is easy to extract from it at least one, and typically a multitude of correct sequences. Two related problems can be defined in the context of assembly planning - Precedence-graph Optimization Problems (POPs) and Sequence Optimization Problems (SOPs). The POPs deal with Precedence graph Optimization - obtaining assembly plans or precedence graphs that optimize a certain performance measure *in addition* to satisfying all constraints. The SOPs are concerned with identification of optimal assembly sequences for specified performance measures such as typical resource constrained project scheduling objectives or assembly line balancing objectives. POP and SOP are the subjects of discussion in the Part II paper and in (Naphade, 1997). We now define the CSP more formally. Suppose 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 us assume that there are n such liaisons or assembly tasks that need to be performed to fully assemble the product. However, all possible sequences in which n tasks may be performed or all precedence graphs that can be drawn on n nodes are not always feasible. Precedence constraints arise from issues such as accessibility, stability of the assembly during the process etc. and may render many of the sequences infeasible.

For an assembly of n tasks numbered 1 through n, the question and answer procedures of Bourjalt (1984) and DeFazio and Whitney (1987) yield a set of logical statements that completely represent the complex system of precedence constraints among these n tasks. These statements are called Establishment Conditions (Homem de Mello and Sanderson, 1991a). Given a certain task k to be performed, these conditions specify what combinations of tasks must be performed or cannot be performed before task k. A feasible assembly sequence must satisfy all establishment conditions specified. For instance consider the following fictitious establishment condition for a certain task 3 :

$$(1 \text{ } and \text{ } (2 \text{ } or \text{ } 4)) \text{ } or \text{ } (5 \text{ } and \text{ } 6) \rightarrow 3 \tag{1}$$

From the above condition we infer that at least one of the following sets of tasks need to be completed before task 3 can be performed : {1,2},{1,4},{5,6}. Such establishment conditions may exist for each of the tasks to be performed.(Note that for feasibility, there must be at least one task without any predecessors). Any assembly sequence (complete order) or precedence graph (partial order) describing a feasible assembly plan must satisfy all establishment conditions. Thus the CSP can be defined as follows:

CSP: *Given a set of establishment conditions on the n tasks of an assembly, generate a sequence for performing these tasks or a precedence graph of these tasks that satisfies all the establishment conditions.*

Establishment conditions such as 1 can be transformed through rules of predicate calculus into standard forms to obtain the sets of tasks mentioned above. However before we comment on what standard forms are of interest and how the transformation can be made, we must first introduce the precedence operator (→) (read "precedes") into our system of predicate calculus. For this operator we define several properties :

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).

Using the distributive properties defined above, any establishment condition can be transformed so that the precedence operator has a single task on either side rather than a logical combination of tasks. For instance equation 1 above can be written as :

$$((1\rightarrow3) \text{ and } ((2\rightarrow3) \text{ or } (4\rightarrow3)) \text{ or } ((5\rightarrow3) \text{ and } (6\rightarrow3))) \qquad \textbf{(2)}$$

We look upon the expressions (1→3), (2→3) (4→3) etc. as *variables* (or *literals*) that can take truth values T (true) and F(false). If the variable ($1 \rightarrow 3$) takes a truth value T, then in the sequence that results task 1 must precede task 3. For an assembly consisting of n tasks, there are n(n-1)/2 such variables. Substituting x_1, x_2, x_3 etc. for the expressions in (2), the establishment condition now looks like :

$$(x_1 \textit{ and } (x_2 \textit{ or } x_3)) \textit{ or } (x_4 \textit{ and } x_5) \qquad \textbf{(3)}$$

A variable (x_1, x_2, x_3 etc.) or its negation ($\sim x_1$, $\sim x_2$, $\sim x_3$ etc.) is called a literal. An expression like (3) formed with literals and connectives (*and, or*) is called a *formula.* A formula assumes a truth value that is dependent on the truth values of the literals that it consists of.

It is well known that every formula can be converted into its conjunctive and disjunctive normal forms (Nilsson(1989), Steen(1972)). For this work we are interested in the conjunctive normal form (CNF) of the establishment conditions. The CNF of a formula is a conjunction of a finite set of clauses, where each clause is a disjunction of literals. Any logical formula can be converted into its CNF in polynomial time (Cormen et. al., 1990). This is done by repeatedly replacing expressions of the form x_1 *or* (x_2 *and* x_3) by (x_1 *or* x_2) *and* (x_1 *or* x_3). For instance, the CNF of formula (3) is :

$$(x_1 \textit{ or } x_5) \textit{ and } (x_2 \textit{ or } x_3 \textit{ or } x_5) \textit{ and } (x_1 \textit{ or } x_4) \textit{ and } (x_2 \textit{ or } x_3 \textit{ or } x_4) \qquad \textbf{(4)}$$

The reader may verify that any set of truth values for the variables that satisfies (3) will satisfy (4) and vice-versa. Thus every establishment condition can be represented as a formula in its CNF. A feasible assembly sequence must satisfy all the establishment conditions. As all conditions need to be satisfied, they can all be concatenated by the *and* connective leading to a larger formula in its CNF. We now need to assign truth variables to the literals in this formula such that the formula assumes a truth value T. This is identical to the Satisfiability problem which is NP-complete (Papadimitrou and Steiglitz, 1982). We have thus shown that CSP reduces to SAT and proved the following -

Theorem: Identifying a feasible assembly sequence is NP-complete.

This is a somewhat paradoxical result especially since the difficulties in assembly planning typically arise out of an unmanageably large number of assembly sequences: finding one or more feasible sequences is rarely difficult. This is because, for many products, the establishment conditions are short, constituting a system of constraints that is a polynomially solvable special case of the general problem (Details in the next section). Hence at first glance the above result appears to have only theoretical value. However this result has a very important algorithmic consequence: Any procedure or algorithm that guarantees to generate a feasible assembly sequence (or a complete and correct set of assembly sequences) ***must*** have a worst case complexity that is not polynomial.

The method developed in this paper solves this NP-complete CSP using three basic steps: Problem Decomposition, Mapping and Partitioning (Figure 1). In the Problem Decomposition step, the CSP (3-SAT) problem is decomposed into a number of linearly solvable subproblems. These subproblems are a generalization of the 2-SAT problem, but are still linearly solvable. In order to solve the subproblems, we map each subproblem on to a graph, which we call the "decision graph". This mapping is the second step in the solution methodology. On the decision graph, the subproblem is posed as a graph partitioning problem. The third step involves partitioning this decision graph to obtain a set of decisions which represents a feasible precedence graph for the original CSP. In the subsequent sections, we describe these three steps in detail.

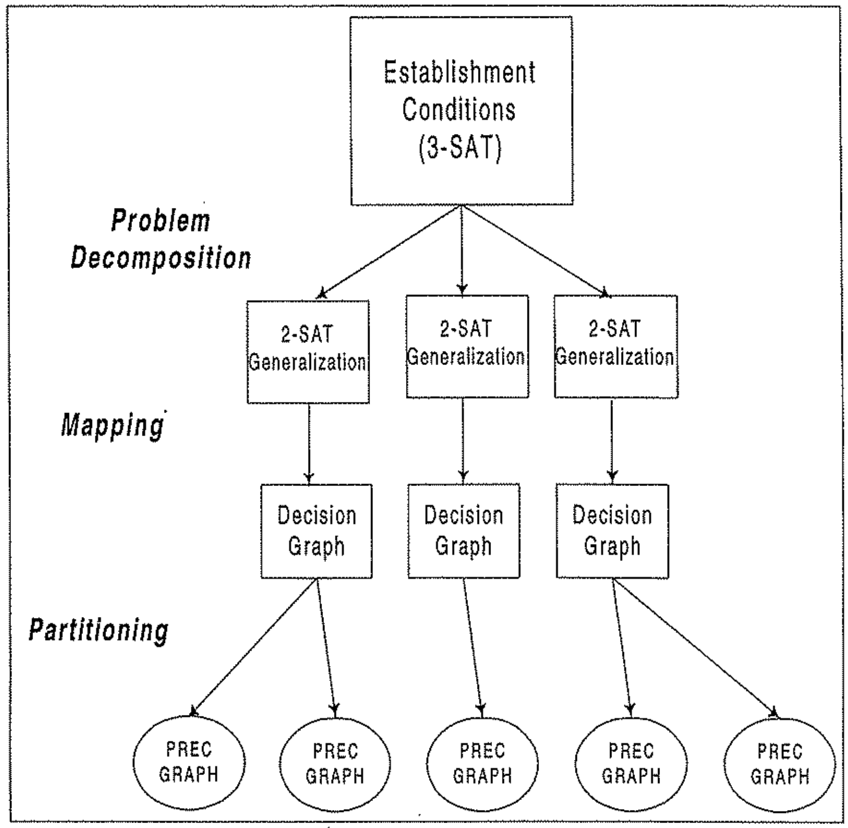

Figure 1
The Hierarchical Framework

4. Problem Decomposition

The 2-SAT problem is a special case of the SAT in which every clause in the CNF is of length two - two literals connected by the *"or"* connective. It is well known (Papadimitrou and Steiglitz, 1982) that the 2-Satisfiability problem is a polynomially solvable special case of the general satisfiability problem. In this section we decompose the CSP or the satisfiability problem *(P)* into *N* simpler instances (Q_i) of a generalized 2-Satisfiability problem. In the 2-Satisfiability problem, all clauses in the CNF are either single literals or a disjunction of two literals. Our problem is a generalization of the 2-SAT for the following reason: In the 2-SAT problem, a literal can have only two values, 0 or 1. However the methods developed in this paper can be used to solve a problem in which a literal can take an arbitrary number of values. This will become clearer in subsequent sections.

It is important to note that this decomposition is necessary only if the original CSP contains one or more establishment conditions that have disjunctions of 3 or more literals in their CNF. If this is not the case, the CSP is already a 2-SAT and hence does not need to be decomposed. In such a case, the CSP is linearly solvable as we will prove in the next section.

The decomposition satisfies the following properties:

*(a) A solution that is feasible for **P** is feasible for at least one* Q_i. $i \in \{1,...,N\}$

(b) A solution that is feasible for any Q_i *is feasible for **P**.* $i = 1,....,N$

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. Consider the following instance of problem ***P*** :

$$(x_1 \textit{ or } x_5) \textit{ and } (x_2 \textit{ or } x_8 \textit{ or } x_5) \textit{ and } (x_3 \textit{ or } x_6) \textit{ and } (x_2 \textit{ or } x_4 \textit{ or } x_6 \textit{ or } x_9) \qquad \textbf{(5)}$$

Clause I **Clause II** **Clause III** **Clause IV**

There are two clauses in ***P*** (II and IV) with more than two literals. In order to decompose the problem, each clause containing three or more literals is broken up into several sub-clauses each of which contains at most two literals. This is done such that a literal in the original clause is present in one and only one of the new subclauses. For instance the clause (x_2 *or* x_8 *or* x_5) is broken up into two sub-clauses : (x_2 *or* x_8) and (x_5). Now suppose the problem ***P*** is replaced by two problems ***P′*** and ***P′′***. The problem ***P′*** contains clauses I, III and IV of ***P*** and the sub-clause (x_2 *or* x_8). The problem ***P′′*** contains clauses I, III and IV of ***P*** and the sub-clause (x_5).

P' : (x_1 *or* x_5) *and* (x_2 *or* x_8) *and* (x_3 *or* x_6) *and* (x_2 *or* x_4 *or* x_6 *or* x_9) **(6)**

P'' : (x_1 *or* x_5) *and* (x_5) *and* (x_3 *or* x_6) *and* (x_2 *or* x_4 *or* x_6 *or* x_9) **(7)**

It is clear that any solution that is feasible for either P' or P'' is feasible for P. It is also fairly simple to see that all feasible solutions of P can be obtained by solving both P' and P''. This is because any solution of P that has a truth value T for literal x_2 or literal x_8 will be generated as a feasible solution of problem P' and any solution of P that has a truth value T for literal x_5 will be generated as a feasible solution of problem P''. Thus the problems P' and P'' satisfy properties (a) and (b) above. Clause IV of problem P can be similarly decomposed into two sub-clauses. To now obtain a set of 2-Sat problems that satisfy properties a and b, we take all possible combinations of the sub-clauses as follows :

Q_1 : (x_1 *or* x_5) *and* (x_2 *or* x_8) *and* (x_3 *or* x_6) *and* (x_2 *or* x_4) **(8)**

Q_2 : (x_1 *or* x_5) *and* (x_5) *and* (x_3 *or* x_6) *and* (x_2 *or* x_4) **(9)**

Q_3 : (x_1 *or* x_5) *and* (x_2 *or* x_8) *and* (x_3 *or* x_6) *and* (x_6 *or* x_9) **(10)**

Q_4 : (x_1 *or* x_5) *and* (x_5) *and* (x_3 *or* x_6) *and* (x_6 *or* x_9) **(11)**

The above four problems satisfy properties (a) and (b) with respect to problem P. In summary, any clause j of P that contains k_j (k_j>2) literals is decomposed into $\lceil k_j/2 \rceil$ sub clauses, such that all literals x_i from the original clause are part of one and only one subclause. If k_j is even, each subclause is a disjunction of two literals. If k_j is odd, all but one subclause are a disjunction of two literals and one subclause contains one literal. All clauses of P that contain more than two literals are decomposed as mentioned above. A problem Q_i is formed by taking a combination of subclauses one subclause derived from each of the parent clauses. The total number of such combinations and hence the total number of problems Q_i equals $\eta = \prod_j(\lceil k_j/2 \rceil)$. As properties (a) and (b) are satisfied by this decomposition scheme, solving all Q_i leads to correct and complete generation of all the assembly sequences.

Several insights are noted here:

1. The concept of *constraint elimination* may be used to simplify a problem. Assume that each establishment condition is a disjunction of several literals. If there are two establishment conditions with the same task on the right hand side and if the literals on the left hand side of condition 1 are a subset of the literals on the left hand

side of condition 2, then condition 2 can be eliminated, as condition 1 automatically satisfies condition 2.

e.g. condition 1 : (3 or 4) $\rightarrow$ 6

condition 2 : (3 or 4 or 5) $\rightarrow$ 6

condition 1 : 1 $\rightarrow$ 8

condition 2 : (1 or 3) $\rightarrow$ 8

Condition 2 can be eliminated in both above cases. This is a useful property since the longer establishment condition can be eliminated whenever possible. If the condition eliminated has a length of 3 or more literals, there is a significant reduction in the number of subproblems obtained after decomposition.

2. The decomposition scheme may lead to a non-polynomial number of subproblems. To solve a subproblem, we provide a method which has a worst case complexity that is linear. In other words, for every subproblem, we obtain a feasible solution (or establish the fact that none exists) in linear time. However, in the worst case, all subproblems may need to be investigated before one feasible solution is found. Thus for the overall problem, finding a feasible solution may take non-polynomial time - which is consistent with the NP-completeness result.
3. The decomposition method involves partitioning a constraint of length > 2 into constraints of length ≤ 2. However there is no restriction on which literals need to be in the same subproblem or which literals need to be in different subproblems. A given constraint may be split in several different manners, and so for a given problem, we may have several *alternative* decompositions (possibly an exponential number), each of which is correct and complete. In this paper we arbitrarily select any correct and complete decomposition. Choice of decomposition has little effect on determination of a correct solution, however it has a significant effect on generation of all solutions or identification of optimal solutions (POP, SOP etc.). These issues are discussed in further discussed in the Part II paper.

We have decomposed the CSP into a number of subproblems. In subsequent sections, we describe the mapping and partitioning steps that provide a solution to these subproblems.

5. Mapping the Subproblem onto a "Decision Graph"

Any disjunction of two literals can be represented as a set of if ... then statements. For instance the clause (x_1 *or*

x_5) can also be written as : if ($\sim x_1$) then (x_5); if ($\sim x_5$) then (x_1). For the assembly planning problem, each of the literals is a precedence relation. We introduce the expression ($a \equiv b$) to indicate the absence of any precedence constraints between tasks a and b. In other words, if $a \equiv b$, tasks a and b may be performed in any order *or performed simultaneously.* Suppose x_1 represents $1 \rightarrow 3$ and x_5 represents $5 \rightarrow 3$ then the clause (x_1 *or* x_5) can also be expressed as the following set of if .. then constraints:

if ($3 \rightarrow 1$) then ($5 \rightarrow 3$) ; if ($3 \rightarrow 5$) then ($1 \rightarrow 3$)

if ($1 \equiv 3$) then ($5 \rightarrow 3$) ; if ($3 \equiv 5$) then ($1 \rightarrow 3$)

Thus the subproblems Q_i can now be represented as a combination of two types of constraints. The first type is straight precedence constraints of the form "$A \rightarrow B$" ; and the second type which we shall call the ***decision dependent constraints*** are of the form :

"if ($A \ \mathfrak{R} \ B$) then ($C \rightarrow D$)"

where A, B, C etc. are tasks and $\mathfrak{R}$ is either a precedence relationship($\rightarrow$) or a temporal equivalence relationship($\equiv$). Before a solution to the problem is obtained, the relative sequence of tasks a,b,c,d is not fixed: they are decisions that need to be taken. However the decision associated with the sequence of a and b influences the decision associated with sequencing tasks c and d. Hence the term ***decision dependent constraint.***

The subproblem that results from the decomposition can be represented on a disjunctive graph G as in Figure 2. The nodes on the graph represent tasks. There are three types of arcs on the graph : Conjunctive Arcs, Disjunctive arcs and Hyperarcs. The conjunctive arcs represent fixed precedence relations between tasks : precedence constraints that have to be met and are not subject to the decision making process. For instance in any assembly sequence that results from the graph in Figure 2, Task 2 shall always precede task 5.

The disjunctive arcs have a slightly different interpretation than that of the disjunctive graph representation of machine scheduling problems. In machine scheduling problems, disjunctive arcs represent a resource conflict - the constraint that the two tasks connected by a disjunctive arc cannot be performed at the same time. Hence resolving a disjunctive arc will result in a conjunctive arc in one of the two possible directions. In this case, the resolution of a disjunctive arc between tasks a and b may lead to three results: a conjunctive arc from A to B ($A \rightarrow B$), a conjunctive arc from B to A ($B \rightarrow A$) or no arc between A and B ($A \equiv B$). This is because the constraint here is *not* a resource constraint. It may arise out of physical factors such as stability, reachability and so on. Hence it may indeed be possible to perform both tasks a and b simultaneously. Thus the "no-arc" resolution of a disjunctive arc is very much an alternative that needs to be actively considered.

The "≡" relationship plays an important role in generating precedence graphs. Explicit inclusion of this relationship allows us to omit an arc from the precedence graph if feasible. In general a precedence graph that is sparse or has a small number of precedence arcs makes it easier to use resources efficiently and lends more flexibility to the scheduling problems that may need to be solved once the precedence graph is designed. Also in this case, the disjunctive arcs may not be independent of each other. Resolution of one arc may affect the resolution of another through if...then constraints. This dependence is indicated by the hyperarc between the two disjunctive arcs.

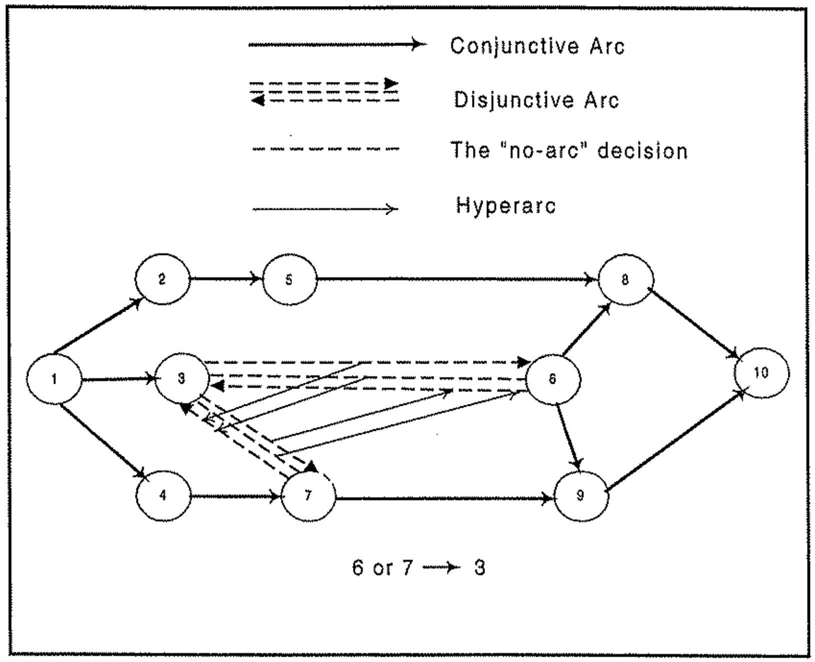

Figure 2
Disjunctive Graph Representation

In order to identify a feasible solution (i.e. a completely conjunctive precedence graph), the disjunctive arcs must be resolved such that all the decision dependent constraints are satisfied. We represent these decision dependent or "if - then" constraints on a decision graph which is later partitioned in order to obtain a feasible set of alternatives. The problem is mapped on to a decision graph G^* from a disjunctive graph G in the following manner:

For every disjunctive arc (A↔B) in G there are three nodes in G^*. Each of these nodes represents a different resolution of the disjunctive arc (A≡B, A→B, B→A). Thus these nodes represent three mutually exclusive alternatives. If there are d disjunctive arcs in G, there are 3d nodes in G^*.

The arcs in G^* represent the decision dependent precedence constraints as follows : If an alternative A→B forces an alternative C→D, then there is an arc in G^* from the node representing A→B to the node representing C→D. Consider as an example the following set of constraints for the disjunctive graph in Figure 2:

$$\text{if } (1\rightarrow 2) \text{ then } (5\rightarrow 8) \tag{12}$$

$$\text{if } (3\rightarrow 7) \text{ then } (6\rightarrow 3) \tag{13}$$

$$\text{if } (3\rightarrow 6) \text{ then } (7\rightarrow 3) \tag{14}$$

$$\text{if } (3\equiv 7) \text{ then } (6\rightarrow 3) \tag{15}$$

$$\text{if } (3\equiv 6) \text{ then } (7\rightarrow 3) \tag{16}$$

$$\text{if } (3\rightarrow 6) \text{ then } (8\rightarrow 5) \tag{17}$$

$$\text{if } (8\rightarrow 5) \text{ then } (9\rightarrow 6) \tag{18}$$

In the above set of constraints, the constraints (13)- (16) are the result of a single logical constraint: (6 *or* 7)→3. The other constraints are not seen to be a result of an "*or*" constraint. However we include these constraints here simply to illustrate that the solution method is capable of handling any decision dependent constraints and is not restricted to those that are derived from an "*or*" constraint.

In Figure 3 we show an illustration of the decision graph G^* for constraints (12) through (18). As mentioned above, every disjunctive arc between tasks a and b can be resolved in three possible ways and hence represents a decision with three

Table I
Decision Indices for example in figure 2

Alternative	Decision Index
1→2	A
3→7	B
3→6	C
5→8	D
6→9	E

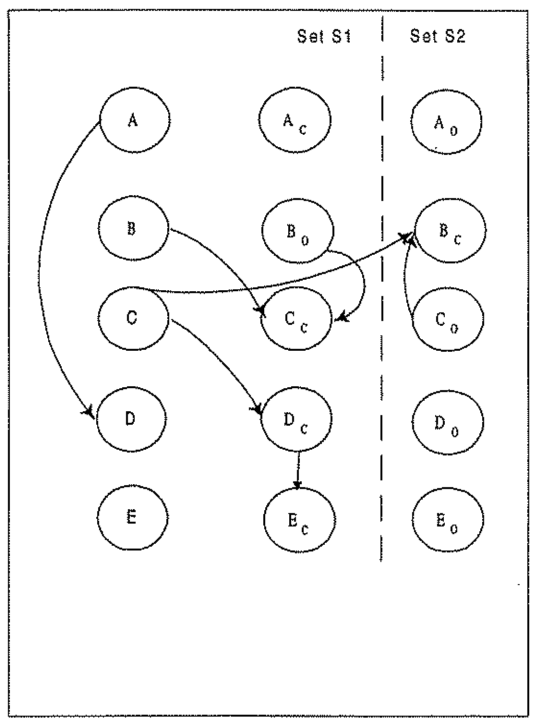

Figure 3
Partitioned Decision Graph

alternative resolutions. Table I lists the correspondence between the disjunctive arcs and the decision indices. For instance if the decision index "A" represents the alternative $1 \rightarrow 2$, we call the alternative $2 \rightarrow 1$ "A_c" and the alternative $1 \equiv 2$ "A_0". Note that any two of these three nodes form the complement set of alternatives for the third node. E.g. the complement of alternative A_c is the set of alternatives $\{A_0, A\}$. Each of these three alternatives is represented by a node on the decision graph G^*.Thus nodes on G^* represent alternatives and arcs on G^* represent implications or consequences of those decision alternatives (Decision Dependent Constraints). The objective of the problem is to select a feasible and "self-consistent" set of alternatives that resolves all the arcs on G. This involves partitioning the decision graph into sets R and A such that set A contains all the accepted alternatives (nodes) and set R contains all alternatives that are rejected. We have thus posed the problem as a graph-partitioning problem.

For an alternative I, we define a complement set C_I. This set contains the "complements" of that alternative. For instance the complement set for node I contains the nodes I_0 and I_c, the complement set for node I_0 contains the nodes I and I_c and so on.

Proposition 1

A feasible and "self-consistent" partition of G^* into sets R and A must satisfy the following properties:

*Property 1 : From the set of nodes $\{I, I_0, I_c\}$ one and only one node must be in set **A**. The other two nodes must be in set **R***

Property 2 : $\nexists$ arc $IJ \in G^*$ with $I \in A$ and $J \in R$

Proof:

Property 1 simply states that every decision must be resolved, and only one alternative from a set of mutually exclusive alternatives can be accepted. The second property ensures that all effects of an accepted decision are accepted in the following manner: If there is an arc in G^* from node I to node J, it means that alternative I forces alternative J. Hence if alternative I is accepted ($I \in A$), alternative J also must be accepted ($J \in A$).

Q.E.D.

The graph partitioning problem can now be stated as follows:

*Partition graph G^* into two sets R and A **(rejected and accepted nodes respectively)**, 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. In this case, set **A** is a feasible set of alternatives.*

We have thus mapped the subproblem on to a decision graph and posed it as a graph partitioning problem. In the next section we provide a method to solve this graph partitioning problem.

6. Partitioning the Decision Graph to obtain a Feasible Solution.

For every node I on graph G^*, let P_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 P'_I as the union of the complement sets of all nodes J in set P_I.

Proposition 2

If I $\in A$ then $P_I \subset A$ and hence $P'_I \subset R$.

Proof:

Let node J $\in P_I$. Assume that J $\in R$ Hence there is a directed path from I to J. Since I $\in A$ and J $\in R$, there must be a pair of nodes K,L on the path from I to J, such that there is an arc from node K to L, K $\in A$ and L $\in R$. This violates property 2 in proposition 1 for nodes K and L. Hence node J must belong to set A and $P_I \subset A$. Since $P_I \subset A$, $P'_I \subset R$ in order to satisfy property 1 in proposition 1.

Note that because of this, the decisions associated with nodes in P_I are completely resolved : They are automatically forced because of inclusion of node I into the accepted set A of alternatives. Similarly, let the set T_I be the set of all nodes K such that there is a directed path from node K to node I.

Proposition 3

If I$\in R$, then $T_I \subset R$.

Proof:

Let K $\subset T_I$. Hence there is a directed path from K to I. Using the same logic as in proposition 2, if I$\in R$ then K $\in R$ and hence $T_I \subset R$.

Note however, that unlike the previous case, here the decisions associated with nodes in T_I are not completely resolved. By placing a node in set R, we are eliminating that alternative from being selected. There may be two more alternatives left, from which the accepted alternative still needs be selected.

Proc_Part(I):

1. Place node I in set $\boldsymbol{A}$.
2. Place nodes belonging to $\boldsymbol{P_I}$ in set $\boldsymbol{A}$ and nodes belonging to $\boldsymbol{P'_I}$ 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.

Figure 4
Partitioning Procedure

We can use these results to construct an algorithm to generate a feasible partition. We first define a partitioning procedure Proc_Part(I) as shown in figure 4. This partitioning procedure takes as an argument a node I. It arbitrarily places node I in set A and repeatedly applies the conclusions of propositions 2 and 3 until one of two termination conditions is met.

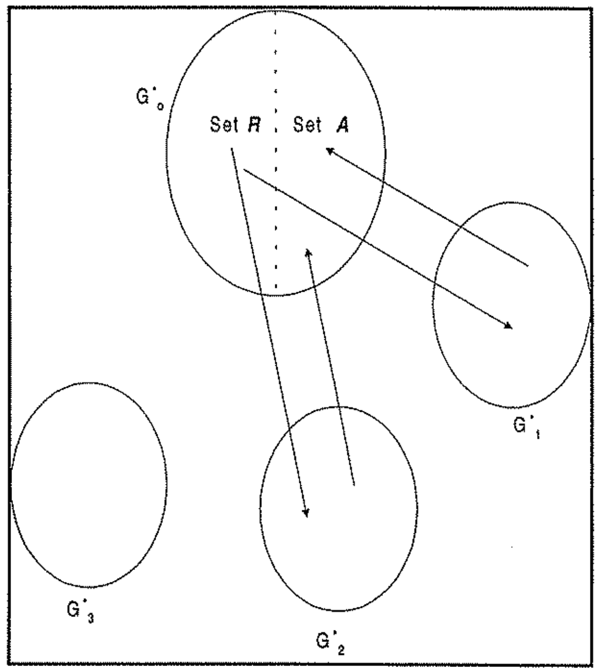

Figure 5
Independence of Unpartitioned Subgraphs

The first condition is an inconsistency : a node previously placed on one side now must be placed on the other side; or both a node and one of its complement nodes need to be placed in set $\boldsymbol{A}$.

The second termination condition for the partitioning procedure is if all nodes that need to be partitioned (assigned to sets) can be partitioned without an inconsistency. This means we now have a partition which satisfies properties 1 and 2 of proposition 1.

Proposition 4

Proc_Part(I) starts by placing a node I in set $\boldsymbol{A}$. This has three possible outcomes.

Case 1: This results in an inconsistency. Hence placing node I in set *A* is not a feasible decision.

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

Case 3: The graph is partially partitioned without any inconsistency. In this case the remaining unpartitioned subgraph is independent of the partitioned subgraph and the partitioning procedure can be independently re-executed on the remaining subgraph.

Proof:

Proc_Part(I) places node I into set *A*. Given that input, the procedure merely enforces all the other alternatives that are forced by this decision (accepting alternative I), and eliminates alternatives that cannot co-exist with alternative I. In other words, the partitioning procedure propagates the effect of accepting alternative I onto the other decisions that are to be made. There is no other independent decision being made in the procedure. All decisions made are consequences of the decision to accept alternative I. After this procedure terminates, there can be three resulting possibilities :

Case 1 :

An Inconsistency: If there is a directed path node I to node L, and also from node I to node L_0 (or L_c), both node L and L_0 (or L_c) need to be placed in set **A**, violating property 1. This means that there is no feasible solution with node I in set **A**. Hence node I cannot be placed in **A**. In this case, we place a complement node of I (a node belonging to set $\mathbf{C}_I$) into set **A**, and re-execute the partitioning procedure. If none of the nodes in $\mathbf{C}_I$ can be feasibly placed in set **A**, then there is no feasible solution to the problem.

Case II :

Either node I or a node from its complement set can be placed in set **A** and as a result of the partitioning procedure, all the nodes of the graph are partitioned. In this case we have a complete and feasible partition.

Case III

Either node I or a node from its complement set can be placed in set **A**. But after implementing the partitioning procedure, only part of the graph is partitioned, and there is a remaining subgraph or mutually disconnected subgraphs that still need to be partitioned (Figure 5). In this case, for each remaining subgraph, we select arbitrarily a new node L, place it in set **A** and repeat the above procedure.

In order to prove that for Case III the algorithm will not cycle, we will now prove that the unpartitioned subgraphs are independent of the partitioned subgraph. Let us designate the partitioned subgraph by $\mathbf{G}^*_o$ and the unpartitioned subgraphs by $\mathbf{G}^*_u$, u=1,2,3 etc. We say that $\mathbf{G}^*_u$ is independent of $\mathbf{G}^*_o$ if the following properties are satisfied:

(i) Nodes in $\mathbf{G}^*_u$ can be partitioned regardless of which set nodes in $\mathbf{G}^*_o$ have been placed.

(ii) Nodes in $\mathbf{G}^*_o$ will not be affected because of the partitioning of $\mathbf{G}^*_u$.

Consider a partially partitioned graph $\mathbf{G}^*$ as in Figure 3.5. The unpartitioned graph may consist of one or more mutually disconnected subgraphs. Some of them may also be disconnected from the partitioned subgraph (e.g. $\mathbf{G}^*_3$) and others which are connected to the original subgraph ($\mathbf{G}^*_1$, $\mathbf{G}^*_2$).

For totally disconnected subgraphs, the independence (properties 1,2) mentioned above is obvious. For subgraphs that are connected, note that any arcs that connect them to the partitioned graph $\mathbf{G}^*_o$ can be of only two types: Those that have a root node in $\mathbf{G}^*_u$ and a tail node in set **A**, and those that have a root node in set **R** and tail node in $\mathbf{G}^*_u$. There can be no arc with a root node in set **A** and tail node in $\mathbf{G}^*_u$: If there was such an arc, the partitioning procedure would

Step 0 Arbitrarily order the decisions to be taken.
Set I = 1.
Step 1 Execute Proc_Part(I)
If node I is the only possible remaining alternative for the associated decision, and if partitioning procedure does not yield feasible solution, STOP. No feasible solution to the problem
Step 2 If Feasible partition obtained
GOTO Step 3
Else
Select an unpartitioned node I′ from complement set of I.
Set I = I′, GOTO Step 1.
Step 3 If no more unpartitioned nodes
STOP. (Feasible solution obtained).
Else
select an unpartitioned node with the lowest cardinality k from the remaining nodes.
Set I=k.
GOTO Step 1.

Figure 6
FEAS_SOL Algorithm for identifying a feasible solution

have placed the tail node in set **A** and it would not be part of the unpartitioned subgraph. Similarly, there can be no arc with root node in the $\mathbf{G}^*_u$ and tail node in set **R** as the partitioning procedure would already have placed the root node in set **R**.

Knowing that there is no arc that starts in $\mathbf{G}^*_u$ and ends in set **R** of the partition, the nodes in $\mathbf{G}^*_u$ can be placed on any side of the partition without violating property 2 for a feasible partition. Because of this relationship between $\mathbf{G}^*_u$ and the partitioned subgraph $\mathbf{G}^*_o$, it is clear that the nodes in $\mathbf{G}^*_u$ can be placed on either side of the partition without affecting the feasibility considerations for nodes that are already partitioned.
Similarly, as there are no arcs that have their root in set **A** and head in $\mathbf{G}^*_u$, nodes in the $\mathbf{G}^*_u$ can be partitioned without considering the partitioned nodes in the partitioned subgraph. Hence each subgraph $\mathbf{G}^*_u$ can now be looked upon as an independent graph that has to be partitioned. The partitioning procedure can be applied to these subgraphs and recursively to the subgraphs that form within them until the entire graph is feasibly partitioned or inconsistency is proven.

In the case of assembly sequence determination, every decision represented by a disjunctive arc (A↔B) has three possible alternative resolutions : (A→B), (B→A) or (A≡B). However, the proof and procedure detailed above, does not depend on the fact that every decision has three alternative resolutions. In other words the cardinality of the complement set of any node is not restricted to a fixed integer for the proof to be valid. Thus the theoretical development made above is applicable to any general scenario where the decision maker has to make several decisions with complex decision-dependencies and where each decision has several different mutually exclusive alternatives. Even for a general case such as this, the procedure outlined above identifies a feasible and self-consistent set of decisions in polynomial time as we will see in section 7. Figure 6 contains a pseudo-code describing the algorithm for obtaining a feasible solution. The first step is arbitrarily ordering the decisions. Different orderings may produce different feasible solutions (or the outcome that there is no feasible solution to the problem). Different feasible solutions can also be obtained by selecting different nodes at Step 4. These issues are more relevant for the POP and the SOP and are discussed the Part II paper..

By completely partitioning the decision graph, we have resolved every decision. In doing so, we have assigned a direction to every disjunctive arc from Figure 2, or removed it from the precedence graph by accepting the "no-arc" alternative. We thus have a completely conjunctive precedence graph that satisfies all establishment conditions. We have completed the description of all three steps of the hierarchical graph theoretic method for solving the CSP. In the next section we will analyze the computational complexity of the method.

7. Computational Complexity

In this section we investigate the worst case and best case computational complexity involved in generating a feasible solution using the hierarchical procedure described in this paper. The first step is conversion of an establishment condition into its CNF. As shown in Cormen et. al. (1990), the length of the CNF that results is a polynomial function of the length of the original formula and also the computation required is polynomial (say P). If there are n_1 establishment conditions, then the time required to convert all of them into the CNF is $O(P(n_1))$. The next step is decomposition of the problem into subproblems. Recall that the entire problem can be converted into a satisfiability problem representation by concatenating all the establishment conditions ($O(n_1)$) in the CNF. Let m be the number of disjunctive clauses in this representation. Selecting a subproblem requires selecting one subclause of length 1 or 2 literals from each disjunctive clause. This is O(m). Mapping the subproblem onto a decision graph is again O(m) as every subclause is converted into an if - then constraint represented by four arcs on the decision graph. The FEAS_SOL algorithm also terminates in O(m) time as proved below :

Since every length 2 establishment condition leads to four if-then constraints, the number of arcs in $\mathbf{G}^*$ is 4m. Let n be the number of nodes in $\mathbf{G}^*$. In the first step the algorithm randomly selects a node, places it in set **A** and all its successors in set **A**. The task of finding all successors can be achieved through a breadth first search that has a worst case complexity of O(m). The validity of property 1 needs to be checked for each node, which is O(n). If feasibility condition 1 is violated, then the complements of the selected node are in turn placed in the set **A** and the same steps are repeated. As mentioned above there are three possible outcomes of this process. In the first case, neither the original

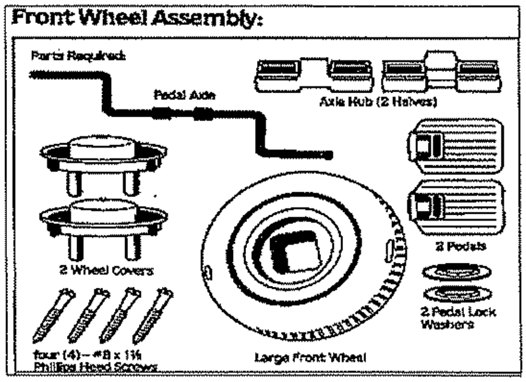

Figure 7
Front Wheel Assembly

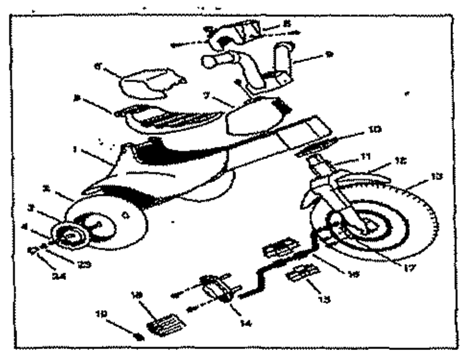

Figure 8
Toy Tricycle Assembly

node nor any of its complements can be placed in set **A**. In this case, no feasible partition exists, and the algorithm terminates with this outcome in O(m+n) time. In the second case, the node or its complement are placed in set **A** without violating feasibility and after all the predecessors and their complements are placed on appropriate sides, no further nodes remain to be partitioned. In this case, the algorithm has identified a feasible partition in O(m+n) time. In the last case, the algorithm may recursively partition subgraphs. Even if the number of subgraphs is large, the computational time of the algorithm will still remain O(m+n). This is because the worst case complexity for partitioning any subgraph is $O(m_s + n_s)$ where m_s and n_s are respectively the number of arcs and nodes that within that subgraph. Summing these up for the whole algorithm would lead to a complexity of O(m+n) where m and n are the number of arcs and nodes in the entire graph $\mathbf{G}^*$.

Hence in all three cases, the algorithm terminates in O(m+n) time to either generate a feasible partition or establish that none exists. The 2-Satisfiability problem is solvable in linear time (e.g. Papadimitrou and Steiglitz, 1982). The problem solved by the above algorithm is a generalization of the 2-Sat problem, where a literal can take more than two values. (A decision may have several different mutually exclusive alternatives available).We have thus proved that the above generalization of the 2-sat problem is also solvable in linear time.

In the best case, the procedure will identify a feasible sequence in $O(m+n+P(n_1))$ time. However if the FEAS_SOL algorithm terminates with the result that no feasible partition exists, a new subproblem has to be selected and solved. As the number of subproblems (η) could be potentially exponential, in the worst case the procedure could take an exponential amount of time : $O(\eta \cdot (m+n+P(n_1))$ to identify a feasible sequence or to establish that none exists.

Node	Decision
0	A≡F
1	A→F
2	F→A
3	B≡F
4	B→F
5	F→B
6	D≡F
7	D→F
8	F→D
9	E≡F
10	E→F
11	F→E

Table II
Decisions represented by nodes on Decision Graph in Figure 9

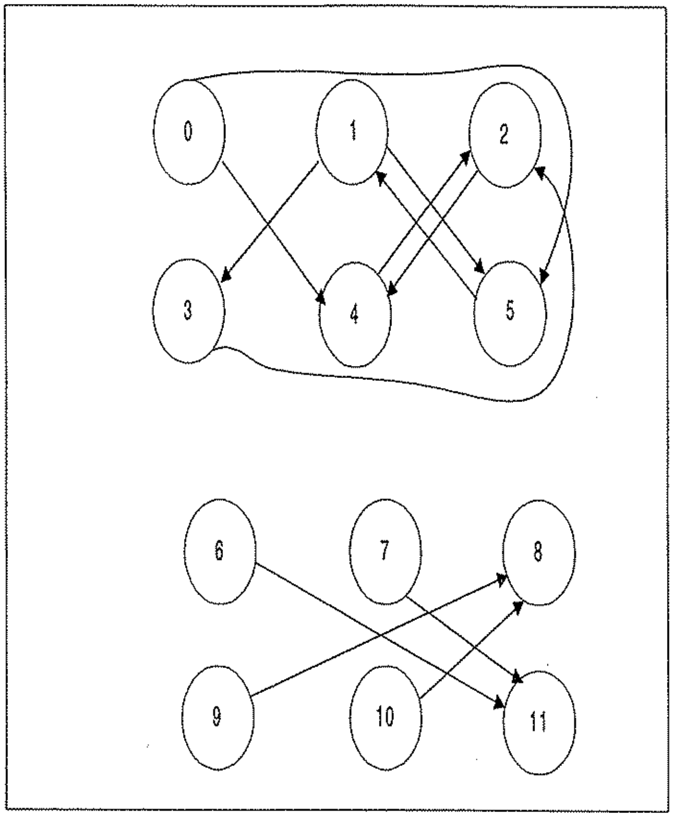

Figure 9
Decision Graph for Front Wheel Assembly

8. A Simple Illustrative Example

Figure 7 shows the components for the Front Wheel Assembly of the tricycle in Figure 8. These are the parts and liaisons that constitute the assembly :

Parts:
1. Front Axle
2. Axle Hub
3. Wheel
4. Left Wheel Cover
5. Right Wheel Cover
6. Left Pedal
7. Right Pedal

Liaisons:
A. Left Wheel Cover - Hub
B. Right Wheel Cover- Hub
C. Hub - Axle
D. Left Pedal - Axle
E. Right Pedal - Axle
F. Wheel - Axle
G. Washer - Left Pedal

8. Left Pedal Washer — H. Washer - Right Pedal

9. Right Pedal Washer — I. Left Wheel Cover- Right Wheel Cover

Using the question and answer method provided by De Fazio and Whitney (1987) the following establishment conditions can be obtained:

$C \rightarrow A$ $\quad$ $C \rightarrow B$ $\quad$ $F \rightarrow I$ $\quad$ $G \rightarrow D$ $\quad$ $H \rightarrow E$

$(A \text{ or } B) \rightarrow F$ $\quad$ $(F \rightarrow A) \text{ or } (F \rightarrow B)$ $\quad$ $(F \rightarrow D) \text{ or } (F \rightarrow E)$

Since this example does not contain constraints with length more than 3 clauses, the decomposition phase is not necessary here. For this example, we demonstrate in Figure 9 the decision graph associated with the assembly constraints stated above. The graph has 4 decisions (12 nodes) and 12 arcs or decision dependent constraints. Table II lists for each node of the decision graph, the alternative that it corresponds to. In order to obtain a feasible partition to this graph, we need to execute the Feas_Sol algorithm of figure 6 as follows:

We first arbitrarily place node 0 in set **A**. This forces both nodes 4 and 5 to be placed in set **A** which is infeasible because it implies acceptance of mutually disjoint alternatives. Hence we try to place node 1 in set **A**. This is also infeasible as it implies placing both nodes 3 and 5 in set **A**. Now we consider node 2. This forces node 4 to be placed in set **A** which produces a feasible partial partition of the decision graph. The next step is selecting an arbitrary node from the remaining subgraph. We arbitrarily select node 6 which forces node 11 for set **A**. This results in a feasible partition of the entire graph. Thus one feasible set of alternatives is 2,4,6,11, forcing the constraints F→A, B→F and F→E (in addition to pre-existing conjunctive constraints) to yield the feasible conjunctive precedence graph shown in figure 10. Note that since the precedence operator is transitive, the arc from C to A is no longer required. S and N are the dummy start and end activities respectively.

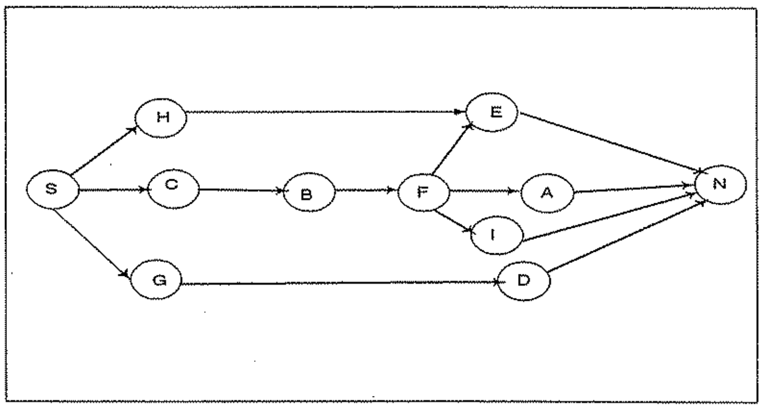

Figure 10
Feasible Precedence Graph (Solution to CSP)

9. Discussions and Conclusions

a) In this paper we developed a graph-theoretic framework for the solution of constraint satisfaction problems encountered in Assembly Planning. This framework forms the basis for development of optimization methods which can be used to select assembly plans which are not only feasible, but also optimal. The framework is generic in that it can be used for optimization of a variety of objective functions.

b) The new concept of decision dependent constraints was introduced and applied to assembly planning problems. This concept provides us a new way of looking at the Satisfiability problems. Using this concept, we proved that the following generalization of the 2-SAT problem is polynomially solvable. Consider literals X_{ij} where $i \in \{1,2,...,M\}$ and $j \in \{1,2,...,N_i\}$. Any 2-SAT problem with literals X_{ij} will remain polynomially solvable even if the following constraints are added:

$$\sum_{j=1}^{j=N_i} X_{ij} = 1$$

c) This method represents an improvement over existing methods for precedence graph generation. Existing methodologies for generating precedence graphs are either not "complete " (Bonneville et. al., 1995) or not accompanied by a rigorous analysis of computational complexity (Chen and Henrioud, 1994). Delchambre and Gaspart (1992) develop a prototype user-friendly software for generation and evaluation of assembly plans, but their method does not rigorously deal with disjunctive constraints.

d) In the context of the existing methodologies of generating feasible assembly sequences, we make the following comments: Homem de Mello and Sanderson (1990, 1991a,b,c) , Lapierre and ElMaraghy (1994) etc. use the AND/OR graph representation. The advantage of the AND/OR graph are that an initial specification of establishment conditions is not required - the transition feasibility is evaluated for each possible transition from an assembly to a pair (or more) of subassemblies. This eliminates the possibility of erroneous specification of assembly constraints that exists in the use of establishment conditions. This graph is also more compact than the diamond graph of De Fazio and Whitney (1987). We offer the following comparisons between our method and the AND/OR graph with regard to generation of a feasible assembly sequence

(i) AND/OR graphs evaluate the feasibility of one transition - from an assembly to two subassemblies- at a time. The question asked is simple - "Is this transition feasible." However, the disadvantage is that every possible transition from an assembly to two subassemblies must be tested for feasibility. In the worst case this is an exponential number and implies asking an exponential number of questions to the design engineer or an exponential number of calls to a feasibility evaluation routine (Homem de Mello and Sanderson (1990). Thus the tradeoff between using AND/OR graphs and establishment conditions is having to ask a few complex questions or a large number of simple questions.

(ii) As far as generation of one feasible assembly sequence is concerned, our method is much more efficient than the AND/OR graph in generating a feasible sequence. We justify this statement as follows: In our method, once we decompose the assembly planning problem, we are left with a certain number (say NS) of subproblems. For each of the NS subproblems we generate a feasible precedence graph, or verify that there is none in linear time. Even though NS could be exponential in the worst case, in most real cases it is a small number. (E.g. NS=4 for AFI of De Fazio and Whitney (1987)). Thus in the worst case it will take us a reasonable number of linear computations to find one feasible precedence graph. On the other hand, for the AND/OR graph approach, a tree in the graph that represents a feasible assembly sequence cannot be identified unless the entire graph is generated. This is a much costlier computation than ours since the entire AND/OR graph contains all possible subassemblies of the product. Also note that decomposition is required only when long establishment conditions exist. If there are no long establishment conditions, our method identifies a feasible assembly sequence in linear time. The AND/OR graph will still need to be fully generated (or until at least one tree of the graph is completely generated).

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Cormen T.H., C.E. Leiserson and R.L. Rivest, Introduction to Algorithms, MIT Press, 1990.
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 A. Wafflard, "An Automatic, Systematic and User Friendly Computer-Aided Planner for Robotized Assembly", *Proc. of the IEEE Int. Conf. on Rob. and Auto*, 1, 592-598, 1991.
10. 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.
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. Huang Y.F. and C.S.G. Lee, "A Framework of Knowledge-Based Assembly Planning",*Proc. of the IEEE Int. Conf. on Rob. and Auto*, 599-604, 1991.
16. Lee Sukhan, G.J. Kim and G.A. Bekey, "Combining Assembly Planning with Redesign: An Approach for more effective DFA.", *Proc. of the IEEE Int. Conf. on Rob. and Auto*, 319-325, 1993
17. Li R. and C. Hwang, "Framework for automatic DFA system development.", *Comp & IE*, 22(4), 403-411, 1992a.
18. Li R. and C. Hwang, "Assembly code generation from a feature-based geometric model.",*Int. Jr. of Prod. Res.*, 30(3), 627-646, 1992b.
19. Lin A.C. and T.C. Chang, "An Integrated Approach to Automated Assembly Planning for Three-Dimensional Mechanical Products", *Int. Jr. of Prod. Res.*, 31(5), 1201-1227, 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 II: Problem Decomposition and Optimization Algorithms", (accompanied paper), 1999.
22. Nilsson N.J., *Principles of Artificial Intelligence*, Morgan Kaufman Publishers Inc., USA, 1980.
23. Papadimitrou C.H. and K. Steiglitz, *Combinatorial Optimization : Algorithms and Complexity*, Englewood Cliffs NJ : Prentice Hall USA, 1982.
24. Steen. S.W.P., *Mathematical Logic*, Cambridge University Press, New York, 1972.
25. Tsao Jungfu and Jan D. Wolter, "Assembly Planning with Intermediate States",*Proc. of the IEEE Int. Conf. on Rob. and Auto*, 71-76, 1993.
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, Number 1, 113-124, 1999.
27. Ye N. and D.A. Urzi, "Heuristic Rules and Strategies of Assembly Planning: Experiments and Implications in the Design of Assembly Decision Support Systems.", *Int. Jr. of Prod. Res.*, 34(8), 2211-2288, 1996.