Proceedings of the 2007 IEEE
International Symposium on Assembly and Manufacturing
Ann Arbor, Michigan, USA, July 22-25, 2007

Sequence Planning to Minimize Complexity in Mixed-Model Assembly Lines

Xiaowei Zhu, S. Jack Hu, Yoram Koren, Samuel P. Marin, and Ningjian Huang

Abstract— **Sequence planning is an important problem in assembly line design. It is to determine the order of assembly tasks to be performed sequentially. Significant research has been done to find good sequences based on various criteria, such as process time, investment cost, and product quality. This paper discusses the selection of optimal sequences based on complexity introduced by product variety in mixed-model assembly line. The complexity was defined as operator choice complexity, which indirectly measures the human performance in making choices, such as selecting parts, tools, fixtures, and assembly procedures in a multi-product, multi-stage, manual assembly environment. The complexity measure and its model for assembly lines have been developed in an earlier paper by the authors. According to the complexity models developed, assembly sequence determines the directions in which complexity flows. Thus proper assembly sequence planning can reduce complexity. However, due to the difficulty of handling the directions of complexity flows in optimization, a transformed network flow model is formulated and solved based on dynamic programming. Methodologies developed in this paper extend the previous work on modeling complexity, and provide solution strategies for assembly sequence planning to minimize complexity.**

I. Introduction

As an important step in assembly system design, sequence planning, or sequence analysis [1] is to determine which assembly task should be done first, which should be done later. Proper determination of the sequence may help balance the line, reduce equipment investment, and ensure better product quality. Significant research has been done to find effective methodologies in search of good sequences.

The process of assembly sequence planning (ASP) begins with the representation of an assembled product, for example, by a graph or adjacency matrix. One type of graph, *liaison graph*, was first introduced by Bourjault in [2] to establish the relationships among component parts in an assembly. The liaison graph is a graphical network in which nodes represent parts and lines (or arcs) between nodes represent liaisons. Each liaison represents an assembly feature where two parts join. The process of realizing such a liaison or liaisons is referred as an *assembly task*. Hence, the problem of ASP is to find a proper order of realizing the liaisons, i.e., the sequence of tasks to completely assemble the product.

This work was supported by the Engineering Research Center for Reconfigurable Manufacturing Systems of the National Science Foundation and the General Motors Collaborative Research Laboratory in Advanced Vehicle Manufacturing, both at The University of Michigan.

X. Zhu, S. J. Hu, and Y. Koren are with the Department of Mechanical Engineering at The University of Michigan, Ann Arbor, MI 48109, USA Email: xwzhu@umich.edu

S. P. Marin and N. Huang are with the Manufacturing Systems Research Lab at the General Motors R&D Center, Warren, MI 48090, USA

Another assembly representation is the precedence graph. A precedence graph is a network representation of *all* precedence relations among *all* assembly tasks. A sample graph is shown in Fig.1. In the graph, nodes represent tasks, and there exists an arc (i, j) if task i is an *immediate predecessor* of task j.

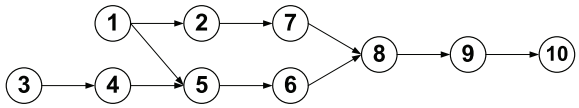

Fig. 1. Precedence Graph of a Ten-Task Assembly (From [3], pp.5)

An immediate predecessor is defined as follows [4]. If $i \prec j$, then task i is known as a *predecessor* or *ancestor* of task j; and j is known as a *successor* or *descendent* of i. If i is a predecessor of j, and there is no other task which is a successor of i and predecessor of j, then i is known as an immediate predecessor of j. A task may have more than one immediate predecessor and can be started as soon as all its immediate predecessors have been completed. By definition, if a task has two or more immediate predecessors, every pair of them must be *unrelated* (i.e., no precedence relationship) in the sense that neither of them is a predecessor of the other. Moreover, if i is a predecessor of j, and j is a predecessor of k, then obviously i is a predecessor of k. In notation, we write: $i \prec j, j \prec k \Rightarrow i \prec k$. This property of precedence relationships is called *transitivity*. By transitivity, therefore, we can determine the set of predecessors, or the set of successors of any task from the set of immediate predecessors of each task (i.e., from the precedence graph).

Typically, one precedence graph corresponds to multiple, sometimes, a large number of sequences. With those candidate sequences, engineers select the best sequences according to a certain criterion. Among the criteria, one of the earliest attempts is to balance the line. Scholl [3] presented a thorough treatment of assembly line balancing. Other engineering knowledge are also incorporated into the selection process [1], for example, removing unstable subassembly state to avoid awkward assembly procedures and improve quality; eliminating refixturing & reorientation to reduce non-value added costs; imposing a subassembly to allow parallel processing; and so on.

Besides the attentions on a single product, researchers have also developed sequence planning methodologies for a group of similar products in a family. Gupta and Krishnan [5] showed that careful assembly sequence design for a product family helped to create genetic subassemblies which can reduce subassembly proliferation and the cost of offering

product variety. Rekiek et al. [6] and Lit et al. [7] developed an integrated approach for designing product family (including assembly sequences) and the assembly system at the same time so that multiple products could be assembled in the same line. Hence, according to which factor is critical to a specific line design problem, new criteria are often developed for the selection of good sequences.

In this paper, we discuss sequence planning to reduce manufacturing complexity in manual, mixed-model assembly line design. The criterion is called operator choice complexity, which measures the uncertainty presented to human operators when they are making choices, such as selecting parts, tools, fixtures, and assembly procedures in a multi-product, manual assembly environment. The complexity measure and its model for assembly line was developed in an earlier paper by the authors [8] and will be reviewed in the next section. According to the complexity models developed, assembly sequence determines the directions in which complexity flows and thus proper assembly sequence planning can reduce complexity.

The objective of this paper is to develop methodologies of finding the optimal assembly sequences to minimize system complexity. The paper is structured as follows. Section II provides background information on complexity measure and model, and shows the opportunity of minimize complexity by assembly sequence planning. Section III discusses the problem formulation and the preliminary attempts to solve the problem based on an integer program (IP). Due to the difficulties of handling constraints in the IP, Section IV presents a network flow program formulation, which transforms the original problem into a solvable traveling salesman problem with precedence constraints expressed by an extended precedence graph. Then, procedures are developed to solve the transformed problem using dynamic programming. Session V demonstrates a numerical example for the ten-task case study shown in Fig.1. Finally, Section VI concludes the paper and suggests the future work.

II. Background on Manufacturing Complexity

In this section, we introduce the complexity model developed for mixed-model assembly lines in an earlier paper [8]. The model considers the product variety induced manufacturing complexity in manual assembly lines where operators have to make choices according to the variants of parts, tools, fixtures, and assembly procedures.

A complexity measure called "Operator Choice Complexity" (OCC) was proposed to quantify the uncertainty in making the choices. The OCC takes an analytical form as an information-theoretic entropy measure of the average randomness (uncertainty) in a choice process. It is assumed that the more certain the operator is about what to choose in the upcoming assembly task, the less the complexity is, and the less chance the operator would make mistakes. Reducing the complexity may help to improve assembly system performance. In fact, the definition of OCC is also similar to that of the cognitive measure of human performance in the Hicks-Hyman Law [9], [10].

A. Measure of Complexity

In a *very* general form, the measure of complexity, OCC is a linear function of the *entropy rate* of a stochastic process (choice process). The choice process consists of a sequence of random choices with respect to time. The choices are modeled as a sequence of random variables, each of which represents choosing one of the possible alternatives from a choice set. In fact, the choice process can be considered as a discrete time discrete state stochastic process $X' = \{X_t, t = 1, 2, \ldots\}$, on the state space (the choice set) $X_t \in \{1, 2, \ldots, M\}$, where t is the index of discrete time period, M is the total number of possible alternatives which could be chosen during each period. More specifically, $X_t = m, m \in \{1, 2, \ldots, M\}$, is the event of choosing the m^{th} alterative during period t. With the above notation, the general form of OCC is:

$$\begin{aligned} H(X') &= \lim_{N\to\infty} \frac{1}{N} H(X_1, X_2, \ldots, X_N) \\ &= \lim_{N\to\infty} H(X_N | X_{N-1}, X_{N-2}, \ldots, X_1) \quad (1) \end{aligned}$$

The second equivalence sign holds if the process is stationary [11].

In the simplest case, if the sequence is independent, identically distributed (*i.i.d.*), Eqn.(1) can be reduced to an analytical entropy function H. The H function takes the following closed form:

$$H(X) = H(p_1, p_2, \ldots, p_M) = -C \cdot \sum_{m=1}^{M} p_m \cdot \log p_m \quad (2)$$

where $p_m \triangleq \mathbf{P}(X = m)$, for $m = 1, 2, \ldots, M$, which is the probability of choosing the m^{th} alternative if the distribution of X is given; $(p_1, p_2, \ldots, p_M)$ describes the distribution of X, which is also known as the *mix ratio* of the (component) variants associated with the choice process.

B. Complexity Propagation

Base on the idea that variety causes complexity in a multi-stage manufacturing system, we define two types of complexity for each station:

- **Feed complexity:** Choice complexity caused by the feature variants added at the current station.
- **Transfer complexity:** Choice complexity of the current station caused by the feature variants previously added at an upstream station.

The propagation scheme of the two types of complexity is depicted in Fig.2, where, for station j, the feed complexity is denoted as C_{jj} (with two identical subscripts), and the transfer complexity is denoted as C_{ij} (with two distinct subscripts to represent the complexity of station j caused by the variants added at an upstream station i). Transfer complexity exists because the feature variant added on the upstream station i has been carried down to station j, and may affect the process of realizing other feature at station j. The effects can cause, for example, accessory part selections, tool changeovers, fixture conversions, or assembly procedure changes. By definition, the transfer complexity can *only*

flow from upstream to downstream, but not in the opposite direction. In contrast, the feed complexity can *only* be added at the current station with no "transfer" behavior.

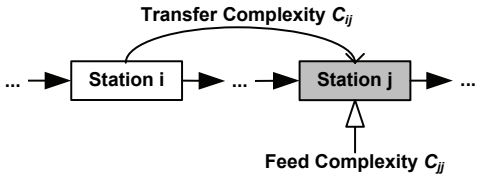

Fig. 2. Types of Complexity

C. System Level Complexity Model

With the two types of complexity defined, we can derive a system level complexity model to characterize the interactions among multiple sequentially arranged stations. Consider an assembly line having n workstations shown in Fig.3. The stations are numbered 1 through n sequentially from the beginning of the line to the end. The mix ratio, i.e., the percentages of component variants added at each station, is known. Using Eqn.(2), we can obtain the entropy H for the variants at each station according to their mix ratios (by assuming an *i.i.d.* component build sequence).

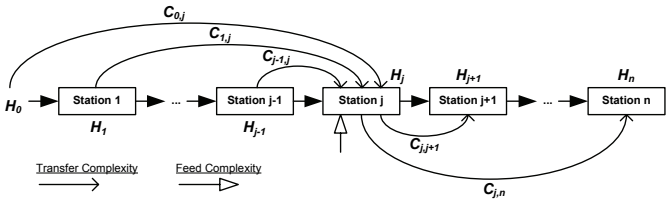

Fig. 3. Propagation of Complexity at the System Level in a Multi-Stage Assembly System

In Fig.3, each directed arc stands for a stream of transfer complexity, C_{ij}, flowing from station i to j (C_{ij} can be zero). Hence the total complexity at a station is simply the sum of the feed complexity at the station and the transfer complexity from all the upstream ones. For station j, the total complexity is:

$$C_j = C_{jj} + \sum_{\forall i: i \prec j} C_{ij} \tag{3}$$

According to the definition of transfer complexity, if component variants added at station i cause choices during the assembly operations at station j, we have:

$$C_{ij} = a_{ij} \cdot H_i, \text{ for } i = 0, 1, 2, \ldots, n-1; j = 1, 2, \ldots, n \tag{4}$$

where,

- H_i – Entropy of component variants added at station i;
- H_0 – Entropy of variants of the base part;
- a_{ij} – Coefficient of interaction between assembly operations at station j and variants added at station i, i.e.,

$$a_{ij} = \begin{cases} 1 & \text{Variants added at station } i \text{ has an impact on station } j, \text{ and } i < j \\ 0 & \text{Otherwise} \end{cases}$$

Therefore, the values of C_{ij}'s are determined by the following two steps.

Step 1: Determine the value of H_i, which depends on the mix ratio of component variants added at station i. As we have mentioned earlier, for component variants with an *i.i.d.* build sequence, Eqn.(2) can be used to calculate the H_i levels. In other words, H_i is determined by the assignment of the assembly task at station i.

Step 2: Determine the value of a_{ij}, which depends on the relationship between the component variants added at station i and the process requirements at station j, which, in turn, is related with the assignment of assembly task at station j.

The two-step procedure of determining C_{ij} makes it difficult to formulate an easy-to-solve optimization problem to minimize total system complexity since different sequences will result in different C_{ij}'s. In addition, the number of candidate sequences can be quite high and it is computationally prohibitive to exhaustively evaluate all of them to find the one with minimum system complexity. Therefore, methodologies and algorithms are needed to search for the optimal sequences.

To begin with, we set up a sequence planning problem as follows. We consider an assembly system, having n assembly tasks, denoted as 1 to n. Tasks are to be arranged sequentially in an order subject to precedence constraints, such as the constraints expressed by the precedence graph in Fig.1, where $n = 10$. Additionally, to make the problem comparable to the original problem in Fig.3, we assume each task corresponds to one and only one station, and vice versa.

According to the multi-stage complexity model in Fig.3, transfer complexity may be found between every two tasks. The complexity becomes effective only from upstream tasks to downstream ones. For example, in Fig.4, when task i precedes task j (also denoted as $i \prec j$), there is transfer complexity flowing from task i to j (denoted as an arc from node i to j). In other words, when task j is performed after task i, it is also possible that the assembly process for task j requires choices in parts/tools/fixtures/assembly procedures according to the variants previously installed by task i. Using the notation of transfer complexity in Fig.2, we know that the amount of complexity incurred in the above scenario is C_{ij}. On the other hand, it is also possible for transfer complexity, C_{ji}, to exist and flow from task j to i if $j \prec i$. Obviously, only one of the two scenarios will take place at one time. Thus, although transfer complexity can exist in either directions, one and only one of the values in the pair (C_{ij}, C_{ji}) is effective for each assembly sequence.

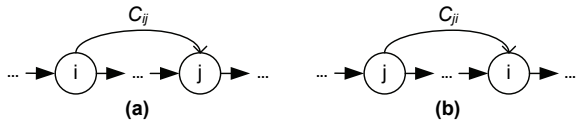

Fig. 4. Transfer Complexity between Two Assembly Tasks i and j, (a) C_{ij} if $i \prec j$, (b) C_{ji} if $j \prec i$

Hence, the assembly sequence determines the directions in which transfer complexity flows, and hence the total system complexity. In the following, we discuss a simplified assembly sequence planning (ASP) problem to minimize complexity.

III. Problem Formulation

In this section, we discuss the assumption and formulation of the sequence planning problem proposed above. In addition, an integer program has been formulated as the first attempt to solve the problem. However, the attempt was not successful, which opened the further discussions in Session IV.

A. Assumption: Position Independent Choice Complexity

For simplicity and practical reasons, we propose an assumption of *position independence* for transfer complexity. That is, the values of C_{ij} and C_{ji} depend *solely* on the relative positions of the two tasks i and j, not the tasks in between, nor their absolute positions in the assembly sequence. Although the assumption seems quite restrictive, it is applicable for assembling customized products with highly modularized components such as the final assembly process of automobiles, home appliances, and electronics. Because of the simplification, the computational effort to find optimal solution is greatly reduced and the problem becomes manageable as well.

Under the above assumption, we are able to determine all the values of transfer complexity between every pair of assembly tasks. For the example of the ten-task assembly system described in Fig.1, a node-node cost array can be formed as shown in Fig.5 to collect all the transfer complexity values, where C_{ij} corresponds to the cell (i, j) at row i and column j. For each feasible assembly sequence, the system complexity then can be found by the summation of all the effective complexity, i.e., $\sum C_{ij}$, where $(i, j) \in \{(i, j)|i \prec j\}$. If no precedence constraint is present, there exist $n!$ feasible assembly sequences. It is obvious that when the constraints are less stringent, the number of feasible sequences can become quite large.

	1	2	3	4	5	6	7	8	9	10
1		C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	$C_{1,10}$
2	C_{21}		C_{23}	C_{24}	C_{25}	C_{26}	C_{27}	C_{28}	C_{29}	$C_{2,10}$
3	C_{31}	C_{32}		C_{34}	C_{35}	C_{36}	C_{37}	C_{38}	C_{39}	$C_{3,10}$
4	C_{41}	C_{42}	C_{43}		C_{45}	C_{46}	C_{47}	C_{48}	C_{49}	$C_{4,10}$
5	C_{51}	C_{52}	C_{53}	C_{54}		C_{56}	C_{57}	C_{58}	C_{59}	$C_{5,10}$
6	C_{61}	C_{62}	C_{63}	C_{64}	C_{65}		C_{67}	C_{68}	C_{69}	$C_{6,10}$
7	C_{71}	C_{72}	C_{73}	C_{74}	C_{75}	C_{76}		C_{78}	C_{79}	$C_{7,10}$
8	C_{81}	C_{82}	C_{83}	C_{84}	C_{85}	C_{86}	C_{87}		C_{89}	$C_{8,10}$
9	C_{91}	C_{92}	C_{93}	C_{94}	C_{95}	C_{96}	C_{97}	C_{98}		$C_{9,10}$
10	$C_{10,1}$	$C_{10,2}$	$C_{10,3}$	$C_{10,4}$	$C_{10,5}$	$C_{10,6}$	$C_{10,7}$	$C_{10,8}$	$C_{10,9}$	

Fig. 5. Transfer Complexity Values between any Two of the Ten Tasks

Among all the feasible assembly sequences, our objective is to find an optimal one with minimum system complexity. Briefly, the optimization problem for the ASP is the following.

Program 1:

Minimize: System complexity $Z = \sum_{(i,j)\in\{(i,j)|i\prec j\}} C_{ij}$
With respect to: Assembly sequence
Subject to: Precedence constraints

B. Problem with Integer Program Formulation

Because of the imposed precedence constraints, Program 1 can be viewed equivalently as finding a node covering chain (or tour) in the precedence graph of Fig.1 with minimum summation of effective complexity flowing from node i to j, where $i \prec j$. Accordingly, an integer program can be formulated as follows.

Program 2:

Decision variables:

$$x_{ij} = \begin{cases} 1 & \text{Task } i \text{ is assigned prior to task } j \\ 0 & \text{Otherwise} \end{cases}$$

Minimize $Z(\mathbf{x}) = \sum_{\forall i,j} C_{ij} x_{ij}$

Subject to:

(i) $x_{ij} + x_{ji} = 1, \forall i, j$

(ii) $x_{ij} \in \{0, 1\}$

(iii) If $(i_1, i_2), i_2, (i_2, i_3), \ldots, i_{n-1}, (i_{n-1}, i_n)$ forms a tour in the precedence graph, then

$$\begin{cases} x_{i_1,i_2} = x_{i_1,i_3} = \ldots = x_{i_1,i_{n-1}} = x_{i_1,i_n} = 1 \\ x_{i_2,i_3} = \ldots = x_{i_2,i_{n-1}} = x_{i_2,i_n} = 1 \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ x_{i_{n-1},i_n} = 1 \end{cases}$$

Constraints (i) and (ii) ensure one and only one task is assigned to a station, constraint (iii) determines the directions of complexity flows. Because of the difficulties in handling constraint (iii) of Program 2, the ASP problem in the above direct formulation is difficult to solve. Attempts have been made to use the methods from network flow modeling to simplify the above problem. That is, we add all the complexity values as flows on the arcs of the precedence graph, such as the one in Fig.1, then solve a minimum flow problem accordingly.

IV. A Network Flow Program Formulation

To begin with, we assume the values of transfer complexity between every pair of assembly tasks are known. A node-node cost array for complexity as in Fig.5 can be formed to collect all these values, where C_{ij} corresponds to the cell (i, j) at row i and column j. However, due to precedence constraints, some of the cells in the array are inadmissible, i.e., if it is infeasible for task i to precede task j, we denote the cell (i, j) as inadmissible, and assign it an ∞ complexity value. Finding all these inadmissable cells and purging them can greatly simplify the original problem. Thus, a procedure is developed below.

A. Procedure of Purging Inadmissible Cells

It takes three steps to find and purge inadmissible cells using the transitivity property of precedence mentioned in Section I.

Step 1:For row i, mark with X in the j^{th} column for all j, where $j \in J_i = \{j|i \prec j\}$ (J_i is the set of

nodes with a precedence relationship with node i, including implicit transitivity relationships).

Step 2: For row i, $\{i, j\}$ is a pair of unrelated elements if the j^{th} row is unmarked. The meaning of unrelated pair is that either task i could be assigned before task j, or vice versa. The incurred complexity of these two scenarios is C_{ij} or C_{ji} respectively. Find out both cell (i, j) and (j, i) and mark them with Y. Fig.6 shows the resulting array after Steps 1 and 2.

	1	2	3	4	5	6	7	8	9	10
1		X	Y	Y	X	X	X	X	X	X
2			Y	Y	Y	Y	X	X	X	X
3	Y	Y		X	X	X	Y	X	X	X
4	Y	Y			X	X	Y	X	X	X
5		Y				X	Y	X	X	X
6		Y					Y	X	X	X
7			Y	Y	Y	Y		X	X	X
8									X	X
9										X
10										

Fig. 6. Resulting Array after Steps 1 and 2

Step 3: By now, all the unmarked cells are inadmissible cells. Mark them with ∞ and restore the corresponding cost coefficients with appropriate subscripts to the rest of cells, and shadow the cells which were marked with X for later use.

B. Equivalent Network Flow Model

It is observed that the complexity costs in the non-shadowed cells are formed in pairs; in each one of the feasible solutions, one and only one of them are included in the total system complexity cost function. However, the complexity values in the shadowed cells are imposed by precedence constraints either explicitly or implicitly; therefore, all of them are by all means counted in every feasible solution.

The above observation implies one of the ways to simplify the complexity cost array without changing the original problem. That is, we simply set all the shadowed cells to zero, see Fig.7. Then the only change to the objective function of the original optimization problem in Program 2 is of a constant, which does not affect the optimal solutions. In fact, the equivalent argument for the simplification is to set complexity from i to j to zero if the precedence constraint requires i to precede j either explicitly or implicitly (through transitivity), i.e., $C_{ij} = 0$ for $i \prec j$.

The cells with denoted complexity cost in Fig.7 are the ones forming an unrelated pair. Draw directed, dotted arcs between i and j in both directions if $\{i, j\}$ is such an unrelated pair, and assign flow values C_{ij}, C_{ji} to the associated arcs respectively. An extended precedence graph is obtained as shown in Fig.8. Notice that the flows on the solid arcs are the complexity values between two tasks with explicit precedence relationships, which are *all zero* due to the simplification stated in the previous paragraph. However the flows on the dotted arcs are the complexity

	1	2	3	4	5	6	7	8	9	10
1		0	C_{13}	C_{14}	0	0	0	0	0	0
2	∞		C_{23}	C_{24}	C_{25}	C_{26}	0	0	0	0
3	C_{31}	C_{32}		0	0	0	C_{37}	0	0	0
4	C_{41}	C_{42}	∞		0	0	C_{47}	0	0	0
5	∞	C_{52}	∞	∞		0	C_{57}	0	0	0
6	∞	C_{62}	∞	∞	∞		C_{67}	0	0	0
7	∞	∞	C_{73}	C_{74}	C_{75}	C_{76}		0	0	0
8	∞	∞	∞	∞	∞	∞	∞		0	0
9	∞	∞	∞	∞	∞	∞	∞	∞		0
10	∞	∞	∞	∞	∞	∞	∞	∞	∞	

Fig. 7. Reduced Complexity Cost Array

values between every *unrelated* pair of tasks, which may take on *non-zero* values.

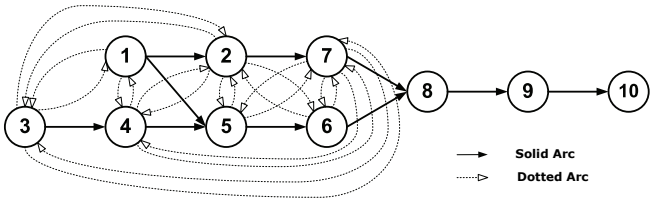

Fig. 8. Extended Precedence Graph

The graph is an equivalent network flow model where the objective is to find a tour which has the least flow cost. This is because each of the feasible solutions corresponds to a path satisfying the following properties in the extended precedence graph.

1) The path is directed, i.e., the travel must follow the direction of the arrows;
2) The path must visit every node one and only once;
3) The path must have the solid arcs directed forward.

Properties 1 and 2 are required simply due to the definition of a precedence graph. In other words, the path is Hamiltonian. Property 3 is imposed because the solid arcs are the explicit precedence relationships (ten in total for the example) which must be satisfied. Once these precedences are satisfied, all the implicit precedence relations will hold automatically, i.e., the precedence constraints specified by the original precedence graph are satisfied. For example, by inspection, one of the feasible solutions is 1-2-3-4-5-6-7-8-9-10. By stretching the path from the extended precedence graph, we find all the solid arcs are directed forward, see Fig.9(a). However, an infeasible solution violating property 3 is also illustrated in Fig.9(b): a path satisfying properties 1 and 2 is taken with the sequence of nodes being 1-4-7-3-2-5-6-8-9-10. By stretching the path again, the solid arcs of $(2, 7)$ and $(3, 4)$ are found to be in the reverse direction. Thus property 3, i.e., the constraint in the original precedence graph, has been violated.

Once the constraints are satisfied, the objective value is computed by counting all the active flows. The active flow is defined as the flow on the dotted arc whose direction is the same as that of the path. In fact, the active flow represents the effective transfer complex in the unrelated pair. Take the

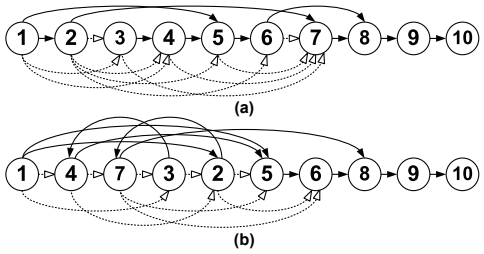

Fig. 9. Stretched Path of (a) a Feasible Solution, (b) an Infeasible Solution

example in Fig.9(a), the active flows are C_{13}, C_{14}, C_{23}, C_{24}, C_{25}, C_{26}, C_{37}, C_{47}, C_{57}, and C_{67}. Therefore, the objective value of the solution is,

$$Z = C_{13} + C_{14} + C_{23} + C_{24} + C_{25} + C_{26} + C_{37} + C_{47} + C_{57} + C_{67} + \text{constant} \quad (5)$$

The reason for adding the constant in the above expression follows from the arguments (of simplification) in the use of the reduced complexity cost array in Fig.7.

In conclusion, by combining properties 1 and 2 with property 3, the equivalent network flow model transforms the original formulation (Program 2) to a problem of finding a Hamiltonian tour in the *extended precedence graph*, and at the same time, subject to the constraints imposed by the *original precedence graph*. The problem is then similar to what is widely known as the traveling salesman problem with precedence constraints (TSP-PC) [12], which can be solved using Dynamic Programming (DP) with some modifications.

In addition, by using the transformation, we gain the advantage of greatly reducing the number of non-zero, finite cells in the complexity cost array. For the purpose of assembly sequence planning, this reduction significantly simplifies the work of evaluating the transfer complexity values by the procedures discussed about Eqns.(1) and (2). For the ten-task example, only the transfer complexity of the pairs $\{1,3\}$, $\{1,4\}$, $\{2,3\}$, $\{2,4\}$, $\{2,5\}$, $\{2,6\}$, $\{3,7\}$, $\{4,7\}$, $\{5,7\}$, and $\{6,7\}$ needs to be evaluated. Moreover, as we shall see shortly, the transformation also provides the means of finding state transition costs in DP, which makes our problem comparable to the classical TSP-PC with C_{ij} being the arc length.

C. Solution Procedures by Dynamic Programming

Here we reformulate our problem and restate it with the notations similar to that of the classical TSP-PC. First, we append a "dummy" node 0 that has no transfer complexity from or to the other nodes, and connect it with the starting nodes (ending nodes) with forward (backward) arcs according to the precedence constraints. The starting node (ending node) is defined as the node having no predecessor (successor). In our example, the starting nodes are 1 and 3, and the ending node is 10. Thus, we add node 0 with forward arcs $(0,1)$, $(0,3)$, and backward arc $(10,0)$ to the graph in Fig.8. Next, let the extended precedence graph be $G = (\mathcal{N}, \mathcal{A})$, which is a directed graph, where $\mathcal{N} = \{0, 1, 2, \ldots, n\}$ is the node set, $\mathcal{A}$ is the arc set. $C_{ij} \geq 0, (i,j) \in \mathcal{A}$ is the complexity incurred if task i precedes task j, i.e., node i is visited prior to j (denoted as $i \prec j$). By convention, let $C_{ii} = \infty, \forall i \in \mathcal{N}$ to eliminate self-loops. Finally, for each node $i \in \mathcal{N}$, precedence relationships defined by the original precedence graph (or, equivalently the solid arcs in Fig.8) can be expressed by means of a set of nodes ($\Pi_i^{-1} \subset \mathcal{N}$) that must be visited *before* node i, or a set of nodes ($\Pi_i \subset \mathcal{N}$) that must be visited *after* node i.

The ASP problem is then cast as one of the variants of the classical TSP-PC in [12] as to find a Hamiltonian tour starting from node 0, visiting every node in $\Pi_i \subset \mathcal{N}$ before entering node i ($i \in \{1, 2, \ldots, n\}$), and finally returning to node 0. The objective is to find a feasible tour that minimizes the sum of the complexity incurred. However, it is important to note that instead of calculating the sum of the costs on its arcs (along the traveling path) as in the classical problem, we compute the sum of C_{ij}'s, where $(i,j) \in \mathcal{A}$ and $i \prec j$. This presents difficulties in handling the state transition cost in developing DP procedures. For that, we need to ensure the following two conditions:

- **Condition 1:** The decision space for going from a state (say $State^*$) to another state (called state transition) depends on $State^*$, not the path coming into $State^*$.
- **Condition 2:** The state transition cost depends on $State^*$, not the path coming into $State^*$.

We will show how to satisfy these conditions in the following discussions.

The DP procedures are developed as follows.

Define state (S, i) as the state being at node $i (i \in S)$ and visited every node in Π_j^{-1} before passing through node $j (\forall j \in S)$, and further define the objective value function $f(S, i)$ to be the least complexity cost "determined" (explained later on the state transition cost structure) on a path starting at node i, legitimately visiting the rest of $(n+1-|S|)$ nodes, i.e., all the nodes in the set $\mathcal{N} \backslash S$, and finally finishing at the dummy node 0.

State transition takes place from state (S, i) to $(S \bigcup \{j\}, j)$ by visiting node j (where $j \in D(S)$) at the next step, where $D(S)$ is the decision space, consists of the set of nodes that can be visited after acquiring state (S, i).

Theorem 1: $D(S)$ is a function of solely S.

Proof: First, denote $Y_k = \{i : |\Pi_i^{-1}| + 1 \leq k \leq n + 1 - |\Pi_i|\}$ as the set of nodes that may stay in position k ($k \in \{1, 2, \ldots, n+1\}$) in any feasible tour. Next, note that, if $|S| = k$, then $j \in Y_k$, i.e., by definition, $D(S) = Y_k \backslash S$. Since Y_k is determined by the precedence graph $G = (\mathcal{N}, \mathcal{A})$, thus $D(S)$ relies only on S.

Put alternatively, $D(S)$ is determined by the set of the nodes we have visited not the node where we are at, nor the path coming into the node. Thus, Theorem 1 shows that Condition 1 has been satisfied.

As we have mentioned earlier, the state transition cost structure of our ASP problem is *quite* different from that of the classical TSP-PC, which causes the difficulty of solving

the problem. However the equivalent transformation in the previous section gains us the insight to transfer the state transition cost to that of the classical TSP-PC as being the arc length from node i to node j.

Theorem 2: The complexity incurred by choosing to visit node j at state (S, i) is a function of the state and node j.

Proof: First of all, we notice that by choosing node j as the next node to be visited, we "determine" the complexity flowing from node j to all the other nodes in the set $\mathcal{N}\backslash(S\bigcup\{j\})$ but not in the opposite direction. The "determined" complexity flows are the finite C_{jk}'s with node $k \in \mathcal{N}\backslash(S\bigcup\{j\})$. To express explicitly, the transition cost from state (S, i) to $(S\bigcup\{j\}, j)$ is $\sum_{\forall k\in K(S,j)} C_{jk}$, where $K(S, j) = \{\mathcal{N}\backslash(S\bigcup\{j\})\}\bigcap\{k|0 \leq C_{jk} < \infty\}$. Therefore, the state transition cost depends only on state (S, i) and node j.

Put alternatively, the active flows are "determined" by selecting finite values in the columns corresponding to the un-visited nodes, and in row j of the reduced complexity array in Fig.7. Therefore, by Theorems 1 and 2, Condition 2 has been satisfied.

Corollary 3: The state transition cost from state (S, i) to $(S\bigcup\{j\}, j)$ is zero, if j is the only candidate decision in $D(S)$, i.e., $D(S) = \{j\}$.

Proof: Since $D(S) = \{j\}$, $\forall k \in K(S, j)$ we have strictly $j \prec k$, which is imposed by precedence constraints. Because of the simplification shown in Fig.7 (where $C_{ij} = 0$ if $i \prec j$), we have $C_{jk} = 0$. Therefore, by Theorem 2, the state transition cost $\sum_{\forall k\in K(S,j)} C_{jk}$ is zero.

The result of Corollary 3 helps to simplify the calculation of state transition costs in DP recursions.

Now the functional equation for the exact solution follows. Moreover, to fully utilize the easily-found decision space $D(S)$, the DP recursion is intentionally developed to be backwards.

Program 3:

$$f(S,i) = \begin{cases} \min_{j|j\in D(S)}\{\sum_{\forall k\in K(S,j)} C_{jk} + f(S\bigcup\{j\}, j)\}, \\ \quad \text{for } \{0\} \subseteq S \subset \mathcal{N}, i \in S; \text{for } S = \mathcal{N}, i \in S\backslash\{0\} \\ \\ 0, \text{ for } S = \mathcal{N}, i = 0 \end{cases}$$

Answer: $f(\{0\}, 0)$

The TSP-PC is known to be NP-hard. Based on DP, the computational complexity of the unconstrained TSP is $o(2^n)$, which is exponentially growing with the number of nodes. Here, the reason that DP is still favorable is because the number of assembly tasks to plan is moderate, ranging from 100 to 200 for a typical automobile plant. Thus it is practically manageable to solve the problem in a reasonable amount of time for the long-term strategic planning, such as ASP. If further computational improvements are needed, heuristics in [12] can be investigated.

V. Numerical Example

By continuing the ten-task example, we demonstrate the numerical results solved by Program 3. We examine the original precedence relationships (network of the solid arcs in Fig.8), and by Theorem 1, we can find the decision space $D(S)$ for every feasible node set S, see Tab.I.

Then the complete Dynamic Programming Network (DPN) is drawn in Fig.10, where nodes represent states (refer to Tab.I for details of the states), and arcs represent possible state transitions (refer to Tab.II for transition costs, where node in the first column (rows) is the starting (ending) node of the arc, and an ∞ value denotes no arc between the two nodes).

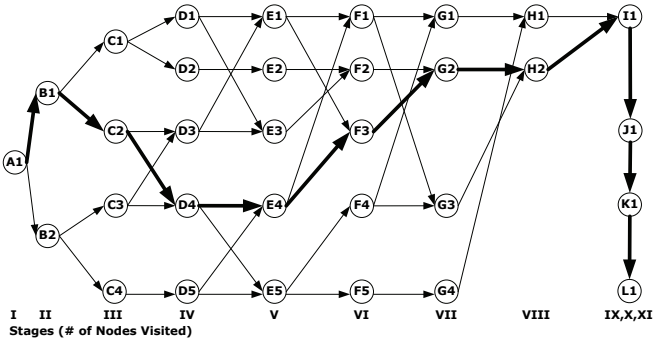

Fig. 10. Complete DPN for the Ten-Task Example

TABLE I
Label, State, and Corresponding Decision Space $D(S)$ for the Nodes of the DPN

Label	State	D(S)	Label	State	D(S)
A1	({0},0)	1,3	E5	({0,1,3,4,5},5	2,6
B1	({0,1},1)	2,3	F1	({0,1,2,3,4,5},5)	6,7
B2	({0,3},3)	1,4	F2	({0,1,2,3,4,7},4)	5
C1	({0,1,2},2)	3,7	F3	({0,1,2,3,4,7},7)	5
C2	({0,1,3},3)	2,4	F4	({0,1,2,3,4,5},2)	6,7
C3	({0,1,3},1)	2,4	F5	({0,1,3,4,5,6},6)	2
C4	({0,3,4},4)	1	G1	({0,1,2,3,4,5,6},6)	7
D1	({0,1,2,3},3)	4,7	G2	({0,1,2,3,4,5,7},5)	6
D2	({0,1,2,7},7)	3	G3	({0,1,2,3,4,5,7},7)	6
D3	({0,1,2,3},2)	4,7	G4	({0,1,2,3,4,5,6},2)	7
D4	({0,1,3,4},4)	2,5	H1	({0,1,2,3,4,5,6,7},7)	8
D5	({0,1,3,4},1)	2,5	H2	({0,1,2,3,4,5,6,7},6)	8
E1	({0,1,2,3,4},4)	5,7	I1	({0,1,2,3,4,5,6,7,8},8)	9
E2	({0,1,2,3,7},3)	4	J1	({0,1,2,3,4,5,6,7,8,9},9)	10
E3	({0,1,2,3,7},7)	4	K1	({0,1,2,3,4,5,6,7,8,9,10},10)	0
E4	({0,1,2,3,4},2)	5,7	L1	({0,1,2,3,4,5,6,7,8,9,10},0)	0

For illustration, numerical values are selected for the cost array by assigning ones to the cells $\{(2, 3), (2, 4), (3, 1), (3, 7), (4, 1), (5, 2), (5, 7), (6, 2), (6, 7), (7, 4)\}$, and zeros to the remaining cells in Fig.7. By calculating the state transition cost according to Tab.II, we obtain the numerical values for every arc of the DPN. Then, we find one shortest path, which has been denoted

TABLE II
STATE TRANSITION COSTS ON THE ARCS OF THE DPN

	B1	B2				
A1	$C_{13}+C_{14}$	$C_{31}+C_{32}+C_{37}$				
	C1	C2	C3	C4		
B1	$C_{23}+C_{24}+C_{25}+C_{26}$	$C_{32}+C_{37}$	∞	∞		
B2	∞	∞	C_{14}	$C_{41}+C_{42}+C_{47}$		
	D1	D2	D3	D4	D5	
C1	C_{37}	$C_{73}+C_{74}+C_{75}+C_{76}$	∞	∞	∞	
C2	∞	∞	$C_{24}+C_{25}+C_{26}$	$C_{42}+C_{47}$	∞	
C3	∞	∞	$C_{24}+C_{25}+C_{26}$	$C_{42}+C_{47}$	∞	
C4	∞	∞	∞	∞	0	
	E1	E2	E3	E4	E5	
D1	C_{47}	∞	$C_{74}+C_{75}+C_{76}$	∞	∞	
D2	0	∞	∞	∞	∞	
D3	C_{47}	∞	$C_{74}+C_{75}+C_{76}$	∞	∞	
D4	∞	∞	∞	$C_{25}+C_{26}$	$C_{52}+C_{57}$	
D5	∞	∞	∞	$C_{25}+C_{26}$	$C_{52}+C_{57}$	
	F1	F2	F3	F4	F5	
E1	C_{57}	∞	$C_{75}+C_{76}$	∞	∞	
E2	∞	0	∞	∞	∞	
E3	∞	0	∞	∞	∞	
E4	C_{57}	∞	$C_{75}+C_{76}$	∞	∞	
E5	∞	∞	∞	C_{26}	$C_{62}+C_{67}$	
	G1	G2	G3	G4		
F1	C_{67}	∞	C_{76}	∞		
F2	∞	0	∞	∞		
F3	∞	0	∞	∞		
F4	C_{67}	∞	C_{76}	∞		
F5	∞	∞	∞	0		
	H1	H2	I1	J1	K1	L1
G1	0	∞	∞	∞	∞	∞
G2	∞	0	∞	∞	∞	∞
G3	∞	0	∞	∞	∞	∞
G4	0	∞	∞	∞	∞	∞
H1	∞	∞	0	∞	∞	∞
H2	∞	∞	0	∞	∞	∞
I1	∞	∞	∞	0	∞	∞
J1	∞	∞	∞	∞	0	∞
K1	∞	∞	∞	∞	∞	0

with thick arcs, see Fig.10. The corresponding optimal solution is 1-3-4-2-7-5-6-8-9-10, and the objective value is one (bit of complexity). As a comparison and thanks to the special selection of numerical values, the infeasible solution (1-4-7-3-2-5-6-8-9-1) illustrated in Fig.9(b) gives the objective value a zero. However the solution violates the precedence constraints as we have pointed out.

VI. CONCLUSION

In this paper, we have demonstrated the opportunity of minimizing complexity for manufacturing systems by assembly sequence planning. The complexity is defined as operator choice complexity, which indirectly measures the human performance in making choices, such as selecting parts, tools, fixtures, and assembly procedures in a multi-product, multi-stage, manual assembly environment.

Methodologies developed in this paper extend the previous work on modeling complexity and provide solution strategies for assembly sequence planning to minimize complexity. The solution strategies overcome the difficulty of handling the directions of complexity flows in optimization and effectively simplify the original problem through equivalent transformation into a network flow model. This makes the problem comparable to the traveling salesman problem with precedence constraints. By a careful construction of the state transition cost structure, we obtain the exact optimal solution through recursions based on dynamic programming. Such solution strategy is also generally applicable to problems in multi-stage systems where complex interactions between stages are considered.

However, due to the restrictive assumption on position independence for complexity, the application of the methodology is still limited. Moreover, the exponentially growing computational complexity is also not satisfactory for large problems with the number of assembly tasks far beyond 200. Hence, the future work should address the above limitations by developing approximations and heuristics without significantly sacrificing the accuracy of the solutions.

ACKNOWLEDGMENT

The authors gratefully acknowledge the financial support from the Engineering Research Center for Reconfigurable Manufacturing Systems of the National Science Foundation under Award Number EEC-9529125, and the General Motors Collaborative Research Laboratory in Advanced Vehicle Manufacturing, both at The University of Michigan.

REFERENCES

[1] D. E. Whitney. *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development*, chapter 7, pages 180–210. Oxford University Press, 2004.

[2] A. Bourjault. *Contribution a une approche mthodologique de assemblage automatis: Elaboration automatique des squences opratoires*. PhD thesis, Universit de Franche-Comt, Besanon, France, 1984.

[3] A. Scholl. *Balancing and Sequencing of Assembly Lines*. Heidelberg; New York: Physica-Verlag, 1999.

[4] K. G. Murty. *Network Programming*, chapter 7, pages 405–435. Englewood Cliffs, N.J.: Prentice Hall, 1992.

[5] S. Gupta and V. Krishnan. Product family-based assembly sequence design methodology. *IIE Transactions*, 30:933–945, 1998.

[6] B. Rekiek, P. De Lit, and A. Delchambre. Designing mixed-product assembly lines. *IEEE Transactions on Robotics and Automation*, 16(3):268–280, 2000.

[7] P. De Lit, A. Delchambre, and J. Henrioud. An integrated approach for product family and assembly system design. *IEEE Transactions on Robotics and Automation*, 19(2):324–334, April 2003.

[8] X. Zhu, S. J. Hu, Y. Koren, and S. P. Marin. Modeling of manufacturing complexity in mixed-model assembly lines. In *Proceedings of 2006 ASME International Conference on Manufacturing Science and Engineering*, Ypsilanti, MI, USA, October 2006.

[9] W. E. Hick. On the rate of gain of information. *Journal of Experimental Psychology*, 4:11–26, 1952.

[10] R. Hyman. Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*, 45:188–196, 1953.

[11] T. M. Cover. *Elements of Information Theory*, chapter 4, pages 60–77. New York: Wiley, 1991.

[12] L. Bianco, A. Mingozzi, S. Ricciardelli, and M. Spadoni. Exact and heuristic procedures for the traveling salesman problem with precedence constraints based on dynamic programming. *INFOR*, 32(1):19–31, 1994.