Proceedings of the 5th IEEE
International Symposium on Assembly and Task Planning
Besançon, France• July 10-11, 2003

A Scheduling Approach to Assembly Sequence Planning

Carmelo Del Valle† Miguel Toro† Eduardo F. Camacho* Rafael M. Gasca†

† Universidad de Sevilla
Dept. Lenguajes y Sistemas Informáticos
Avda. Reina Mercedes s/n
41012 Sevilla, Spain
{carmelo,mtoro,gasca}@lsi.us.es

* Universidad de Sevilla
Dept. Ingeniería de Sistemas y Automática
Camino de los Descubrimientos s/n
41092 Sevilla, Spain
eduardo@cartuja.us.es

Abstract

This paper presents a model for the selection of optimal assembly sequences for a product in multi-robot systems. The objective of the plan is the minimization of the total assembly time (makespan). To meet this objective, the model takes into account the assembly times and resources for each task, the times needed to change tools in the robots, and the delays due to the transportation of intermediate subassemblies between different workstations. The model can be used in different stages of the process planning. The paper includes two algorithmic approaches for solving the scheduling problem: a genetic algorithm intended for the earlier stages, and an A algorithm for the final ones.*

1 Introduction

Assembly planning is a very important problem in the manufacturing of products. It involves the identification, selection and sequencing of assembly operations, stated as their effects on the parts. The identification of assembly operations is done through the analysis of the product structure, using interactive planners [1] [2], or automatically from a geometric and relational model of the assembly [3] and from a CAD model and other non-geometric information [4] [5]. The identification of assembly operations usually leads to the set of all feasible assembly plans. The number of them grows exponentially with the number of parts, and depends on other factors, such as how the single parts are interconnected in the whole assembly, represented in the graph of connections. In fact, this problem has been proved to be NP-complete [6].

The representation of assembly plans is an important is-sue within this scope. The use of And/Or graphs for this purpose [7] has became one of the most standard ways of representing all possible assembly plans. The result is a representation which is adequate for a goal-directed approach. Moreover, this structure is more efficient in most cases than other enumerative ones [7] [8].

Two kinds of approaches have been used for searching the optimal assembly plan. One, the more qualitative, uses rules in order to eliminate assembly plans that include difficult tasks or awkward intermediate subassemblies. A more quantitative approach uses an evaluation function that computes the merit of assembly plans. Several of these proposals can be found in [9] and [10].

The criterion followed in this work is the minimization of the total assembly time (*makespan*) of the plan executed in a multi-robot system. To meet this objective, we present a scheduling-based model which takes into account all factors having an effect on the makespan: an estimation of the duration of tasks; the resources used for them (robots and tools); the times needed for changing tools in the robots; and the delays due to the transportation of intermediate subassemblies between different workstations.

This model completes the one presented in [11], and it can be used in different stages of the process planning: in the first stages, for the design of the assembly and manufacturing system, evaluating the times and costs of different alternatives, and in the final steps of the planning, for determining the adequate assembly sequences. In the first case, the space of solutions is prohibitive for complete and deterministic algorithms. Instead, a genetic algorithm (GA) approach can be used for obtaining good solutions, which can help in the selection of the configuration of the assembly system. Moreover, the solutions obtained by the genetic algorithm can be used for discarding some of the alternative assembly operations, so that we would obtain a reduced set of feasible assembly plans, more suitable for an A* algorithm [12] to obtain an optimal solution, as the one presented in this paper.

The rest of the paper is organized as follows: section 2 describes the assembly sequence planning problem, and section 3 the model proposed. The GA and A* algorithm approaches are presented in section 4 and 5 respectively. Some final remarks are made in the concluding section.

2 Assembly Sequence Planning

The process of joining parts together to form a unit is known as assembly. An assembly plan is a set of assembly tasks with ordering amongst its elements. Each task consists of joining a set of sub-assemblies to give rise to an ever larger sub-assembly. A sub-assembly is a group of parts that can be assembled independently of other parts of the product. An assembly sequence is an ordered sequence of the assembly tasks satisfying all the ordering constraints. Each assembly plan corresponds to one or more assembly sequences.

An And/Or graph is a representation of the set of all assembly plans for a product. The Or nodes correspond to sub-assemblies, the top node corresponds to the whole assembly, and the leaf nodes correspond to the individual parts. Each And node corresponds to the assembly task joining the sub-assemblies of its two final nodes producing the sub-assembly of its initial node. In the And/Or graph representation of assembly plans, an And/Or path whose top node is the And/Or graph top node and whose leaf nodes are the And/Or graph leaf nodes is associated to an assembly plan, and is referred to as an assembly tree. An important advantage of this representation, used in this work, is that the And/Or graph shows how different assembly tasks can be executed in parallel. Figure 1 shows an example of this representation. And nodes are represented as hyperarcs.

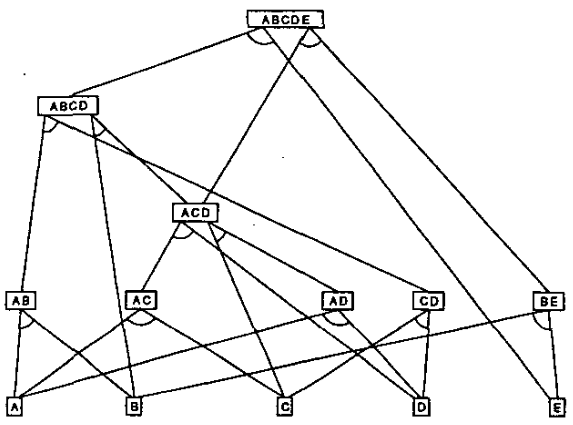

Figure 1: *And/Or* graph of product ABCDE

This work is about the selection of the best assembly plan, that is, one of the And/Or trees of the And/Or graph. Most of approaches used up to now make this selection in a planning phase in which neither the assembly system, nor how the assembly tasks within it will be materialized, is taken into account.

3 The Scheduling-Based Model

This work takes into account the physical realization of the assembly. It is assumed that the assembly tasks corresponding to the And/Or graph have been evaluated separately, in the sense of estimating the resources necessary for their realization (robots, tools, fixtures...) as well as their approximate duration times. For an And/Or graph with a large number of nodes this is not an easy task, and the help of a computer-aided system is necessary. The nodes corresponding to tasks which are not realizable as the adequate tools are not available are eliminated from the And/Or graph.

Another factor taken into account is the time necessary for changing the tools in the robots, which is of the same order as the execution time of the assembly tasks and therefore cannot be disregarded as in Parts manufacturing. $\Delta_{chg}(R,T,T')$ denotes the time needed for installing the tool T in the robot R if the tool T' was previously installed. Notice that any change of configuration in the robots can be modeled in this way.

Another question is the transportation of parts and subassemblies, that could affect the total assembly time. The proposed model supposes a well-dimensioned system, with a perfect planning when executing the assembly plan, so that, when a part would be required in a robot for executing an assembly operation, it will be present there. The same cannot be guaranteed for an intermediate subassembly, because it could be built in a robot and required immediately in another one to form another subassembly. $\Delta_{mov}(SA,R,R')$ denotes the time needed for transporting the subassembly SA from robot R to robot R'.

The model considers only one combination *duration-robot-tool* for an assembly task. However, it can be extended easily when there are various ways for assembling a subassembly from the same components. It is enough to suppose that they corresponds to different assembly tasks, that is, we would add some alternative *And* nodes into the *And/Or* graph for the product.

In order to an easier reasoning, we will suppose that the precedence constraints are in the opposite direction, so that we will refer to a task preceding another one if the first one appears higher in the And/Or graph. This is as if we think about the opposite problem, that of the disassembly. To get the correct solution of the problem, we must only reverse the sequence given by the algorithms.

With this model, the choice is not limited to the assembly plan, but also it can be specify when each task is to be carried out in order to minimize the makespan (some tasks which could potentially be carried out in parallel have to be delayed because they need common resources).

The results derived from this model can be used in different stages of the whole planning process, from the design of the product and of the manufacturing system, to the final execution of the assembly plan.

4 The GA Approach

Genetic Algorithms (GAs) have been used to solve a large variety of combinatorial optimization problems with some success [13] [14]. The nature of the Assembly Sequence Planning problem entails a great difficulty in applying GAs: a sequence of tasks forms a correct solution if all of them belong to an assembly plan, i.e. an assembly tree of the And/Or graph, and they are ordered according to the precedence constraints imposed by the plan. An assembly task is defined by the subassemblies used to form a greater subassembly, and by the resulting assembly. Thus, the presence of a task in a solution is strongly conditioned by the presence of tasks related to these subassemblies.

The first issue in applying GAs is the chromosomal encoding. A natural way of representing a solution is through a sequence of tasks, compatible in order with the constraints imposed by the And/Or graph (ordering and assembly plans). So, not all the tasks sequences form a legal solution. Figure 2 shows how a chromosome is decoded to produce a schedule. A schedule builder transforms the chromosome into a legal assembly schedule, taking into account the precedence constraints and the shared resources to be used (machines and tools). This translation is made directly because of the simplicity of that representation. The result could be visualized as a Gantt chart, and it allows the fitness function (the makespan) to be calculated. Note that, depending on the assignation of resources to tasks and their durations, different chromosomes could be mapped into an only schedule. It will happen when tasks do not share the same resources and could be executed in parallel.

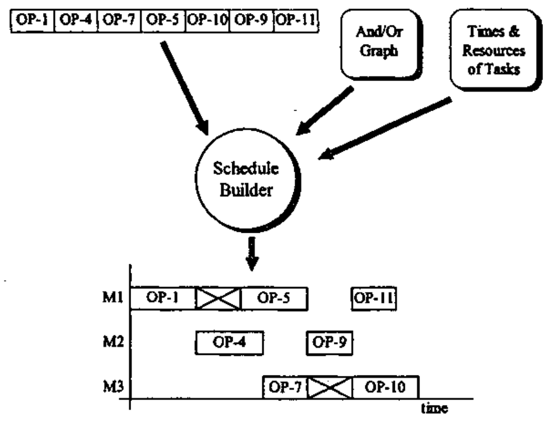

Figure 2: The Schedule Builder.

Two families of genetic operators have been defined for searching the whole solution space. The first includes operators that search locally for new sequences in a predetermined assembly plan, that of parent chromosomes. These operators, referred to below as Re-Ordering Tasks operators, are similar to those used for other sequencing problems in the literature [13] [14], but obviously result to be insufficient to find the optimum. The other family of operators is intended to search for sequences in other assembly plans, and are referred to as Re-Planning operators. This is basically made by introducing a new task in a solution, and substituting certain tasks for others in order to maintain the soundness of the chromosomes.

4.1 Re-Ordering Tasks (ROT) Operators

This kind of operator is intended to search for new sequences in a predetermined assembly plan. Because of the improbability of two sequences of the same assembly plan coinciding in a population, they are implemented as mutation operators. They operate in a chromosome by selecting a random task in the sequence and attempting to move it to another random position. Their predecessor or successor tasks might be also involved in the movement, so that they may keep in their positions or move with the selected task. Those possibilities give us four different genetic operators. The transposition of tasks is performed so that the resultant individual is legal.

4.2 Re-Planning Tasks (RP) Operators

This kind of operator is intended to search for sequences in other assembly plans. The resultant individuals will contain new assembly tasks, coming from another individual present in the population (crossover operators) or generated randomly (mutation operators). Some other new tasks are required in order to complete a correct chromosome, so that they substitute some others.

The sequences generated by these genetic operators will maintain the position of tasks they had in the parents, and the new task will fill the blanks, at some compatible order with the precedence constraints.

RP Crossover (RP-C) operators take two individuals (parents) and generate two children, trying to merge genetic information from the two parents. Because of the nature of Assembly Planning there will be little chance of constructing a new solution from significant parts of any two solutions. The generation of children is made by selecting one task in one of the parents, so that their successor tasks in that parent are also selected. The remaining tasks in the new individual will be selected from the other parent, whenever possible, or randomly, in order to complete a legal chromosome.

RP Mutation (RP-M) takes an individual and modifies it by changing a random sub-tree of the assembly plan for another, selected randomly and according to the constraints imposed by the And/Or graph. The positions of the new tasks in the sequence will be the same that held the substituted tasks.

4.3 Results from the GA

A hypothetical product has been used in order to evaluate the genetic operators described in the previous section. That product is formed by 30 parts, and the number of connections among them is the minimum. The product includes in its And/Or graph various alternative tasks for each Or node, and contains 396 Or nodes and 764 And nodes. There are about 10^{21} possible individuals.

The values corresponding to the higher part of figure 3 represent the average of 50 trials. The lower part represents the best result in all trials. Moreover, all values represent the average of 10 different distributions of durations and shared resources among the tasks. They show the best solution found until the number of evaluations indicated. The graphics include also the value of the optimum solution (OPT) and the performance of a random algorithm (RND).

Figure 3 shows the operation of the specific genetic operators. The high difficulty for merging genetic information from two any individuals could explain the relatively poor results obtained by RP-C in comparison with those expected from typical crossover operators. In fact, RP-M obtains slightly better results, maybe because it preserves more genetic information in the individuals. Moreover, ROT operators improve more quickly at first. At last, their performance is conditioned by the assembly plans generated in the initial population. A last curve is generated in figure 3. It shows the results obtained by a GA with all referred operators working together (ALL). A quite improvement can be observed. This reflects the combination of the two effects: ROT operators optimize assembly plans that have been generated, and RP operators obtain new assembly plans.

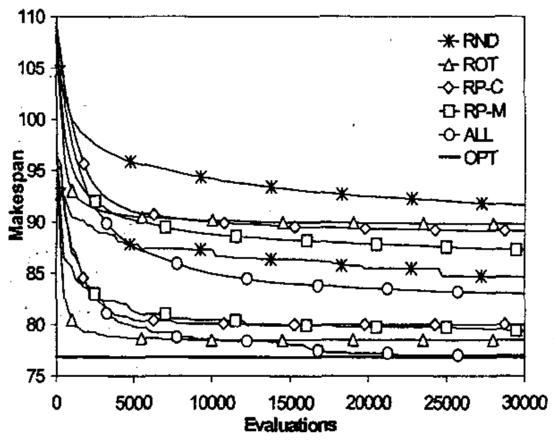

Figure 3: Results from the GA.

5 The A* Algorithm Approach

Another algorithm, based on the A* search [12], has been developed to solve the problem stated in section 3. The algorithm has two well-differentiated parts: one of them considers the sequential execution of assembly tasks imposed by the precedence constraints defined in the And/Or graph, i.e. in the high part of the And/Or graph. The other solves the parallel execution of assembly tasks (the representation through the *And/Or* graph allows a natural study of this stage). This is actually the most complex section, because the execution of tasks on one side of the global assembly is not independent of the rest, and can influence the execution of tasks in the other part of assembly.

The sequential part of the algorithm is used while the tasks considered involves only a non-trivial sub-assembly below the corresponding And node. When an assembly task takes two non-trivial sub-assemblies, the parallel part of the algorithm is used for obtaining the solution from the node in the search tree. In that moment, the algorithm generates all the assembly trees from that And node. Each of them is used for obtaining an optimal order for the tasks included in them, through a separate A* algorithm. The global algorithm orders all these trees using an estimation of the time needed for the execution of its tasks, so that not all the trees must been completely solved, because of the pruning of the search.

In the search tree of the algorithm, a node represents a state corresponding to the execution of the set of tasks that have been included into the solution in the previous steps. The order of including tasks specifies the order of execution of them. So, a task will not be included until all its predecessor tasks in the And/Or graph have been included. This strategy allows to verify the precedence constraints of the problem. The state of a node *n* can be obtained also through the set of tasks *cand*(*n*) that can be included in the next expansion step, denoted *candidates*. For each candidate task *O* we take the earliest start time, *est*(*O*), if it were introduced in the next step. The description of the state of *n* is completed with the time corresponding to the last used of each robot *R* due to the tasks that have been included, *lastTime*(*n*, *R*), and the last tool used in each robot, *lastTool*(*n*, *R*).

The objective function, *f*(*n*), is given by the time needed for the execution of the tasks included in *n*, *g*(*n*), plus an estimation of the time needed to complete a solution, *h*(*n*). Function *g*(*n*) can be defined as:

$$g(n)=\max\left(\max_{O_i\in cand(n)}\left(est(n,O_i)\right),\max_{R_i\in robots}\left(lastTime(n,R_i)\right)\right) \quad (1)$$

Some different heuristic functions can be defined for *h*(*n*). In order to maintain the admissibility of the A* algorithm, *h*(*n*) must be an optimistic estimation of the remaining time for an optimal solution from *n*. In order to calculate properly the objective function *f* from *g* and *h*, two different types of slack have been defined, one for the candidate tasks, *e*(*n*, *O*) = *g*(*n*) – *est*(*n*, *O*), and another one for the robots, *e*(*n*, *R*) = *g*(*n*) – *lastTime*(*n*, *R*).

5.1 Heuristic Functions

For the sequential part of the algorithm, *h*(*n*) can be defined as:

$$h(n)=\min_{O_i\in ntOr(n)}\left(hs(O_i)\right) \quad (2)$$

ntOr(*n*) denoting the non trivial Or node below the last task introduced in *n*, and

$$hs(O)=dur(O)+\max\left(\min_{O_i\in Or_1(O)}\left(hs(O_i)\right),\min_{O_i\in Or_2(O)}\left(hs(O_i)\right)\right) \quad (3)$$

where *dur(O)* is the duration of task *O* and $Or_1(O)$ and $Or_2(O)$ are the Or nodes corresponding to the initial subassemblies involved in task *O*.

Notice that only the precedence constraints have been used in the definition of *h(n)* for the sequential part of the algorithm. For the parallel part, the constraints due to the use of resources can be taken into account, because of the separation of the assembly trees, so that for each subproblem all tasks are defined, i.e. there is no alternative tasks, and then the amount of uses for the different resources are known.

Two basic heuristic functions can be defined for the parallel part of the A* algorithm, considering separately the two types of constraints: the precedence of tasks, and the use of resources.

5.1.1 The heuristic function h_1: precedence of tasks

It corresponds to an estimation of the time remaining if the interdependencies between different branches in the tree are not taken into account. It is looked at only in depth. It can be defined with following equations:

$$h_1(n) = \max\left(0, \max_{O_i \in cand(n)} \left(h_1(O_i) - e(n, O_i)\right)\right) \tag{4}$$

$$h_1(O) = dur(O) + \max_{O_i \in suc(O)} \left(h_1(O_i) + \tau_{mov}(O_i, O)\right) \tag{5}$$

$$\begin{aligned}\tau_{mov}(O_i, O) = \max\big(&\tau(O_i, R(O), H(O)),\\ &\Delta_{mov}(sa(O_i), R(O_i), R(O))\big)\end{aligned} \tag{6}$$

$$\begin{aligned}\tau(O, R, T) = \max\Big(0, \max_{O_i \in suc(O)} \big(\big(h_1(O_i) + {}\\ \tau(O_i, R(O), T(O)\big) - h_1(O)\big)\Big)\end{aligned} \tag{7}$$

In the above expressions, *R*(*O*) and *T*(*O*) are the robot and tool necessary for the execution of task *O*. t(*O*, *R*, *T*) is the added delay, due to the fact that the tool *T* is being used by robot *R* in task *O* and *successors*, because of the necessary tool changes. The equation (7) defines t(*O*, *R*, *T*) when $R \neq R(O)$. In the case $R=R(O)$, t(*O*, *R*, *T*) is defined as $\Delta_{ch}(R, T(O), T)$ (that could be zero if $T=T(O)$). Finally, $t_{mov}(O, O')$ is the delay considering the possible transportation of the intermediate subassembly generated between the execution of *O* and *O'*, and that of the possible change of tools.

Notice that $h_1(O)$ does not depend on the expansion nodes, so that it can be calculated for each task prior to using the A* algorithm for the assembly trees.

5.1.2 The heuristic function h_2: use of resources

It corresponds to an estimation of the time needed if only the remaining usage times of the tools in each robot are taken into account, further supposing the number of tool changes to be at a minimum. It can be defined as follows:

$$h_2(n) = \max_{robots} \left(h_2(n, R_i) - e(n, R_i)\right) \tag{8}$$

where $h_2(n, R_i)$ is the minimum time of use of robot R_i without considering the task precedence constraints. If each tool is associated with only one robot, the calculation of $h_2(n, R)$ is equivalent to the traveling salesman problem, when considering the tools not yet used and an initial node corresponding to the last-used tool in the robot *R*:

$$h_2(n, R) = \left(\sum_{T_j \in R} \sum_{O_i \in cand(n)} h_2(O_i, T_j)\right) + \Sigma\left(n, \Delta_{chi}(R)\right) \tag{9}$$

with $h_2(J, T)$ the remaining time of usage of tool *T* by task *O* and its successors. The term $\Sigma\left(n, \Delta_{chi}(R)\right)$ refers to the time needed for the tool changes. In the usual case that tool-changing times do not depend on the type of tool, it can be calculated easily. Without any precedence information, an in order to maintain the admissibility of the heuristic, it must be supposed that the remaining tools will be installed only once.

5.1.3 Combination of heuristics

The heuristic functions h_1 and h_2 present two different effects in calculating *h*(*n*). The estimation made from the first one is due to the most unfavorable candidate task. In the other hand, h_2 shows an additive effect, because of the uses of robots by all candidate tasks. Therefore, a new heuristic function can be defined from the combination of both, taking the most realistic estimation, $\max(h_1(n), h_2(n))$.

5.2 Results from the A* Algorithm

The use of A* algorithms presents some problems. The most important is the storage space that could be occupied. In order to improve this issue, the algorithm was adapted so that it uses a depth-first search periodically for finding a new solution whose value could be used for pruning the search tree. Another improvement was done about detecting symmetries, so that redundant nodes are avoided in the expansion.

The algorithm was used to solve the same 10 problems than those referred in section 4.3. Table 1 shows how many times the optimal solution was found by a depth-first movement (N-Pr), how many times the algorithm did not find the optimal solution in 30 seconds, when the available memory was exhausted (N-F), and the error rate. It can be observed that in most cases the algorithm found the optimal solution quickly, but in some cases, the high consumption of memory makes it very inefficient. It is easy to conclude that, for larger problems, there will be a lower chance of running efficiently.

Heuristic	Time (ms)			N-Pr	N-F	% Error
	Ave	Max	Min			
h_1	19429	30590	180	4	6	0,990
h_2	1422	6420	0	5	0	0,000
$\max(h_1, h_2)$	4291	30050	0	4	1	0,248

Table 1: Results from the A* algorithm

6 Conclusions

A model for the selection of optimal assembly sequences for a product in a generic multi-robot system has been presented. The objective of the plan is the minimization of the total assembly time. To meet this objective, the model takes into account, in addition to the assembly times and resources for each task, the times needed to change tools in the robots, and the delays due to the transportation of intermediate sub-assemblies between different workstations.

Two algorithmic approaches have been used to solve the resultant scheduling problem. GAs are used to find a sub-optimal solution, but they have not problem of memory. The A* approach allows finding an optimal solution in a more efficient way, but limited to the use of memory. So, the first approach can be used in the earlier stages of the planning, when designing the manufacturing system. The A* algorithm can be used in the final stages, when the space of solutions have been reduced after discarding most of uninteresting alternatives.

References

[1] A. Bourjault. *Contribution à une Approche Méthodologique de Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires.* Thèse d'état, Uni-versité de Franche-Comté, Besançon, France, 1984.

[2] T.L. De Fazio and D.E. Whitney. Simplified Generation of All Mechanical Assembly Sequences. *IEEE Journal of Robotics and Automation*, Vol. 3, No. 6, 1987, pp. 640-658. Also, Corrections, Vol. 4, No. 6, pp. 705-708.

[3] L.S. Homem de Mello and A.C. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation.* Vol. 7, No. 2, 1991, pp. 228-240.

[4] T. L. Calton. Advancing design-for-assembly. The next generation in assembly planning. *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning*, pp. 57-62, Porto, Portugal.

[5] B. Romney, C. Godard, M. Goldwasser, G. Ramkumar. An Efficient System for Geometric Assembly Sequence Generation and Evaluation. *Proceedings 1995 ASME International Computers in Engineering Conference*, pp. 699-712.

[6] R.H. Wilson, L. Kavraki, T. Lozano-Pérez and J.C. Latombe. Two-Handed Assembly Sequencing. *International Journal of Robotic Research.* Vol.14, 1995, pp. 335-350.

[7] L.S. Homem de Mello and A.C. Sanderson. And/Or Graph Representation of Assembly Plans. *IEEE Transactions on Robotics and Automation.* Vol. 6, No. 2, 1990, pp. 188-199.

[8] J. Wolter. A Combinatorial Analysis od Enumerative Data Structures for Assembly Planning. *Journal of Design and Manufacturing.* Vol 2, No. 2, June 1992, pp. 93-104.

[9] M.H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *International Journal of Computational Geometry and Applications*, 9:371-418, 1999.

[10] L.S. Homem de Mello and S. Lee (eds.) *Computer-Aided Mechanical Assembly Planning.* Kluwer Academic Publishers, 1991.

[11] C. Del Valle and E.F. Camacho. Automatic Assembly Task Assignment for a Multirobot Environment. *Control Engineering Practice*, Vol. 4, No. 7, 1996, pp. 915-921.

[12] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.

[13] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley. A Comparison of genetic sequencing operators. In R.K. Belew and L.B. Booker, eds. *Proceedings of the Forth International Conference on Genetic Algorithms, ICGA-91*, pp. 69-76. Morgan Kaufmann, 1991.

[14] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, ed. *The Handbook of Genetic Algorithms*, pp. 332-349. Van Nostram Reinhold, 1990.