Binary vs Symbolic Chromosomal Encoding in GA-based Selection of Optimal Assembly Sequences

CARMELO DEL VALLE[*], EDUARDO F. CAMACHO[**]
[*]Dept. Lenguajes y Sistemas Informáticos
[**]Dept. Ingeniería de Sistemas y Automática
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla
SPAIN

Abstract: - This work presents an application of genetic algorithms to assembly sequence planning. This problem is more difficult than other sequencing problems that have already been tackled with success using these techniques, such as the classic Travelling Salesperson Problem (TSP) or the Job Shop Scheduling Problem (JSSP). It not only involves the arranging of tasks, as in those problems, but also the selection of them from a set of alternative operations. And/Or graphs are used as an efficient structure for the representation of all the possible assembly plans. Thus the selection process results in a search in the And/Or graph of the product. Two representations of solutions are compared in this work. The first one uses a binary chromosome that corresponds to an ordering of the enumeration of all possible assembly sequences. Conventional genetic operators are used with this representation. In the other one a chromosome represents a sequence of tasks, compatible with the ordering constraints imposed by the And/Or graph. Two families of genetic operators have been used for searching the whole solution space. The first includes operators that search for new sequences locally in a predetermined assembly plan, that of parent chromosomes. These operators are similar to those used in TSP and JSSP in the literature. The other family of operators introduces new tasks in the solution, replacing others to maintain the validity of chromosomes, and it is intended to search for sequences in other assembly plans.

Key-Words: - genetic algorithms, optimization, scheduling, sequencing, assembly planning, manufacturing systems, robotics IMACS/IEEE CSCC'99 Proceedings, Pages:7341-7346

1 Introduction

Genetic Algorithms (GAs) have been used to solve a variety of optimization problems with some success. Combinatorial problems are a class of problems particularly difficult to solve, sequencing problems included. Many of them have been studied using evolutionary techniques, such as TSP and JSSP, well known as NP-complete problems. Starkweather, *et al.* [11] and Syswerda [12], as well as many others, study these problems using genetic operators that perform permutations in the solutions.

This paper presents an application of GAs to the problem of selecting and sequencing assembly operations. This is a more difficult planning problem than TSP and JSSP. It involves not only the optimal arrangement of tasks, as in those problems, but also the selection of them from a set of alternative operations, and taking into account the constraints imposed to build a feasible assembly plan.

Assembly planning is a very important problem in the manufacturing of products. It involves the identification, selection and sequencing of assembly operations, specified by their effects on the parts. The identification of assembly operations has been tackled by analysing the product structure, either using interactive expert systems [2] [3] or, more recently, through planners working automatically from geometric and relational models [6] and from CAD models and other non-geometric information [1] [10].

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 within the whole assembly, i.e. the structure of the graph of connections. In fact, this problem has been proved to be NP-complete in both the two-dimensional [9] and three-dimensional [8] [13] cases.

The representation of assembly plans is an important issue within this scope. The use of And/Or graphs for this purpose [5] [6] is becoming one of the most standard ways of representing all possible assembly plans. They can be obtained by studying the opposite problem, that of disassembly, but maintaining the constraints of assembly. Most automatic plan-

ners work by this strategy. The result is a representation, which is adequate for a goal-directed approach. Moreover, Homem de Mello and Sanderson [5] and Wolter [14] showed that this structure is more efficient in most cases than other enumerative ones.
An optimum assembly plan is now sought, selected from the set of all feasible assembly plans. Two kinds of approaches have been used for choosing an optimal one. One, the more qualitative, uses rules in order to eliminate assembly plans that includes difficult tasks or awkward intermediate subassemblies. Another approach, the more quantitative, uses an evaluation function that computes the merit of assembly plans. There are various of these proposals in the book edited by Homem de Mello & Lee [7], a monograph of the subject.

The criterion followed in this work is the minimization of the total assembly time (*makespan*) in the execution of the plan. To meet this objective, the algorithm starts from the And/Or graph (compressed representation of all feasible assembly plans) and the information about each assembly task (robot and tool needed and estimation of assembly time). This is the approach followed by Del Valle & Camacho [4], that apply A* algorithms to solve the problem.

The rest of the paper is organized as follows: Section 2 describes the problem of selection and sequencing of assembly tasks. The description of the two representational models proposed for the GA is found in Section 3 and some of the results obtained are presented in Section 4. 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. The joining process results in the connection of one part with parts already assembled. A sub-assembly is a group of parts having the property of being able to be assembled independently of other parts of the product. 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. 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 possible 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

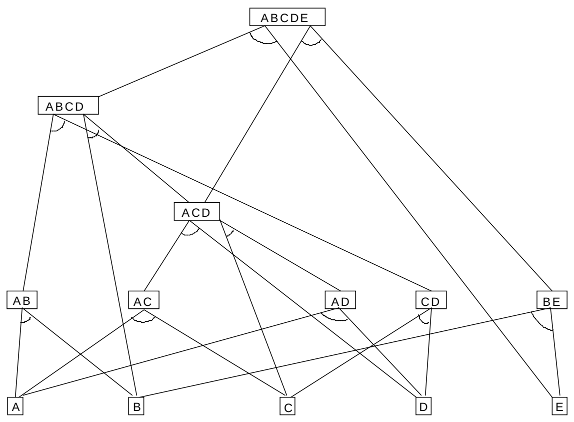

Fig. 1. And/Or graph of product ABCDE

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 the independence of assembly tasks that can be executed in parallel. Figure 1 shows an example of this representation. And nodes are represented as hyperarcs.

The problem is focused on searching an optimal assembly sequence, an ordering of an assembly plan (one of the And/Or trees of the And/Or graph). The evaluation of solutions implies a previous estimation for the times and resources (robots, tools, fixtures...) needed for each assembly task in the And/Or graph. These times should include an estimation for the times needed for other operations, such as transportation of parts and subassemblies. Another factor taken into account here, 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.

3 The Genetic Algorithm

The nature of the Assembly Sequence Planning problem imposes 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.

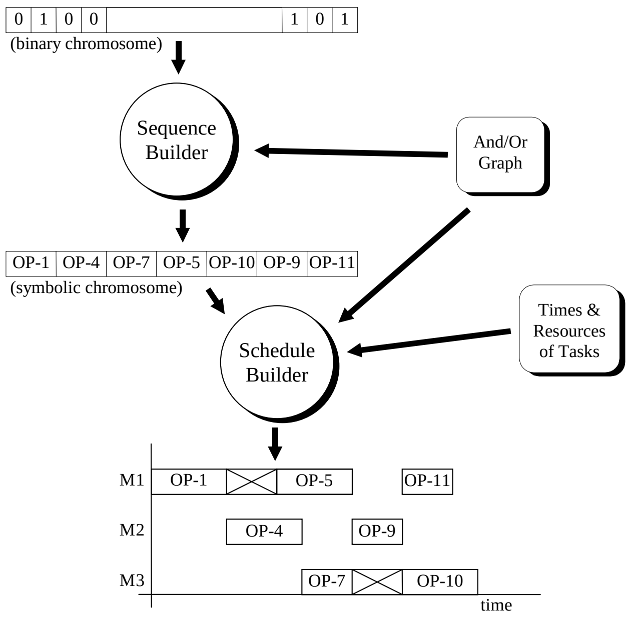

Fig. 2. Decoding of chromosomes.

The first issue in applying GAs is the chromosomal encoding. Two ways of representing a solution have been studied in this work: binary and symbolic. The first one allows the use of conventional crossover and mutation operators, but genetic information transferred would be poor. A symbolic encoding would require specific genetic operators, but it would be more effective.

Figure 2 shows how a chromosome is decoded to produce a schedule. In the case of binary representation, a *sequence builder* transforms it into a symbolic one, that represents 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 constitute a valid symbolic representation. The sequence builder is described in the next subsection.

A schedule builder transforms the symbolic representation 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 easily 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 symbolic chromosomes could be mapped into a unique schedule. It will happen when tasks do not share the same resources and could be executed in parallel.

3.1 Binary Representational Model

An individual, represented as a binary chromosome, corresponds to a valid sequence of tasks. This encoding is based in numbering all the possible individuals. For any non trivial subassembly *SA* in the *And/Or* graph, the number of assembly sequences to build it is:

$$NS(SA) = \sum_{t_i \in T(SA)} NS(SA_{i1}) \cdot NS(SA_{i2}) \cdot \begin{pmatrix} NP_{i1} + NP_{i2} - 2 \\ NP_{i1} - 1 \end{pmatrix}$$

SA_{i1} and SA_{i2} being the subassemblies used by task t_i, one of the alternative tasks to build *SA*, and NP_{i1} and NP_{i2} being the number of parts in those subassemblies. The combinatorial number takes into account the different ways to form a sequence from two subsequences maintaining the relative order for the tasks belonging to the same original subsequence. Note that $NP_{i1} - 1$ is the number of necessary tasks to build SA_{i1}. For any trivial subassembly (one part), the number of sequences *NS* must be defined to be 1.

The enumeration of all the sequences will allow encoding all the individuals, so that each sequence will have a different integer, whose binary code is the chromosome. The number of sequences of the complete product give the length of the chromosome, $length = \lceil \log_2 NS\,(PRODUCT) \rceil$. The enumeration is made so that for any *Or* node, corresponding to a subassembly *SA*, all the sequences formed from the same initial task (departing from the Or node) are distributed consecutively. If *NAND*(SA) is the number of alternative task departing from the Or node corresponding to *SA*, and these tasks are arranged, $t_1,..., t_{NAND(SA)}$, then the distribution of integers for the sequences starting with the task t_i are in the interval

$$\left[\sum_{j<i} NS_j , \sum_{j \le i} NS_j \right)$$

with

$$NS_j = NS(SA_{j1}) \cdot NS(SA_{j2}) \cdot \begin{pmatrix} NP_{j1} + NP_{j2} - 2 \\ NP_{j1} - 1 \end{pmatrix}$$

The information to be saved is the lower bound of the interval, named *base*(t_i). The sequence builder (see Fig. 3) is based in all that information to construct a sequence from a binary coded integer in the interval [0..*NS*(*PRODUCT*)). The algorithms use the operations *first*, *rest* and **::** for obtaining the first task of a sequence, the sequence minus the first task, and the sequence resultant from a first task and a sequence; comb(*m*, *n*) denotes the combinatorial number, and div(*a*, *b*) returns the quotient and the rest of the integer division.

Ideally, if all numbers in the expressions would be 2^k for any integer *k*, the most significant bits would represent the first task selected, the next bits would represent the sequence of tasks from the first subassembly, and that of the second subassembly, Finally,

```
Algorithm SEQUENCE ($x$, $SA$)
    if $SA$ non trivial
        $< task, SA_1, SA_2 > \leftarrow (x, SA)$
        $x \leftarrow x - base(task)$
        $< q_1, r_1 > \leftarrow$ div ($x$, comb ($NP_1$, $NP_2$))
        $< q_2, r_2 > \leftarrow$ div ($q_1$, $NS$ ($SA_2$))
        $s_1 \leftarrow SEQUENCE (q_2, SA_1)$
        $s_2 \leftarrow SEQUENCE (r_2, SA_2)$
        $s \leftarrow task :: MERGE (s_1, s_2, NP_1, NP_2, r_1)$
    else $s \leftarrow [\,]$
    return $s$

Algorithm MERGE ($s_1$, $s_2$, $NP_1$, $NP_2$, $x$)
    if $s_1 = [\,]$ return $s_2$;
    if $s_2 = [\,]$ return $s_1$
    else if $x <$ comb ($NP_1 + NP_2 - 3$, $NP_1 - 2$)
        $s \leftarrow MERGE$ (rest($s_1$), $s_2$, $NP_1 - 1$, $NP_2$, $x$);
        return first ($s_1$) :: $s$
    else
        $x \leftarrow x -$ comb ($NP_1 + NP_2 - 3$, $NP_1 - 2$);
        $s \leftarrow MERGE$ ($s_1$, rest($s_2$), $NP_1$, $NP_2 - 1$, $x$);
        return first ($s_2$) :: $s$
    endif

Algorithm SEQUENCE-BUILDER ($x$, $PRODUCT$)
    return SEQUENCE ($x$, $PRODUCT$)
```

Fig. 3. The Sequence Builder.

the less significant bits would represent the merge of the sequences. That scheme would repeat recursively for each sequence, i.e. the second and the third sequence of bits (see Fig. 4).

Actually, the distinction of bits is no clear, and the genetic information transferred will not be so high. It would happen that two consecutive integers with the same more significant bits refer to two very different assembly sequences, even with different assembly tasks. Note that less significant bits will lose more genetic information than most significant ones.

3.1.1 Genetic Operators

Conventional crossover and mutation operators have been used with the binary representation. In the case of crossover, only simple crossover has been considered, because of the poor genetic information that less significant bits could transfer, as indicated before.

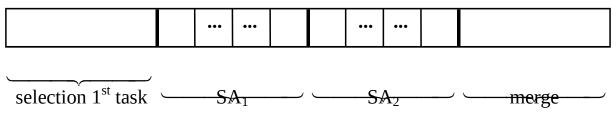

Fig. 4. Binary encoding of individuals.

3.2 Symbolic Representational Model

As indicated before, a symbolic encoding of an individual represents a sequence of tasks, compatible in order with the constraints imposed by the And/Or graph (ordering and assembly plans), but without taking into account the resources to be used. That was the input to the schedule builder (Fig. 2).

Two families of genetic operators have been used 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 TSP and JSSP in the literature, 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 validity of the chromosomes.

3.2.1 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.

3.2.2 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 subtree 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 Experiments and Results

As indicated above, numerous factors have an influence in the complexity of the proposed problem. Some of the most important factors are the number of parts of the product to be assembled, the size and structure of the And/Or graph, and the distribution of durations and shared resources (robots, tools, fixtures...) among all possible assembly tasks that could compose the solution.

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. Note that the number of different schedules depends not only on the number of sequences, but also on the distribution of shared resources (and their number) and durations among all tasks. Thus, various individuals could be transformed in a unique schedule.

The values corresponding to the higher part of the graphics in figures 5, 6 and 7 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 5 shows that there is no substantial difference in the results obtained by the standard crossover and mutation operators in the binary representation model working alone (C and M, C slightly worse than M) or together (ALL, slightly better than M). In all cases, they improve the results from the random algorithm amply.

Figure 6 shows the operation of the specific genetic operators in the symbolic representation model. 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 Fig. 6. 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.

The better results from the two models studied, binary vs symbolic representation are resumed in Fig.

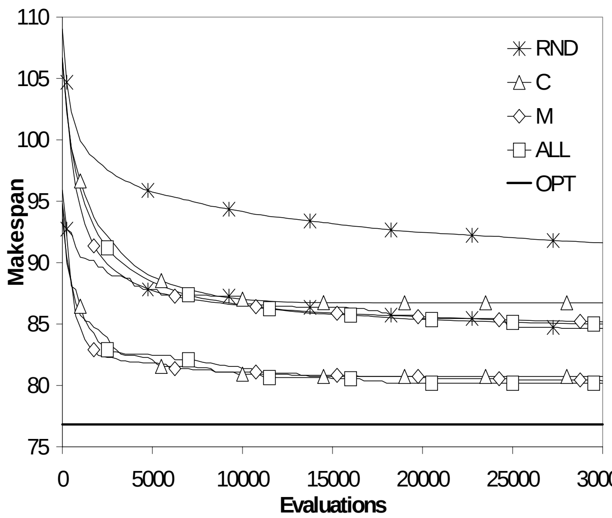

Fig. 5. Results from the binary model.

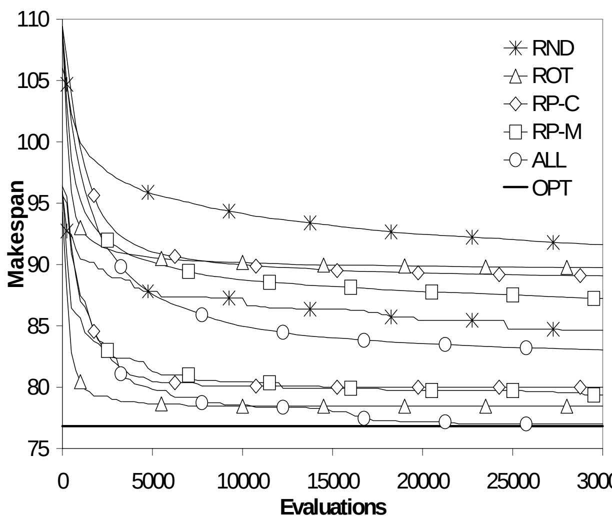

Fig. 6. Results from the symbolic model.

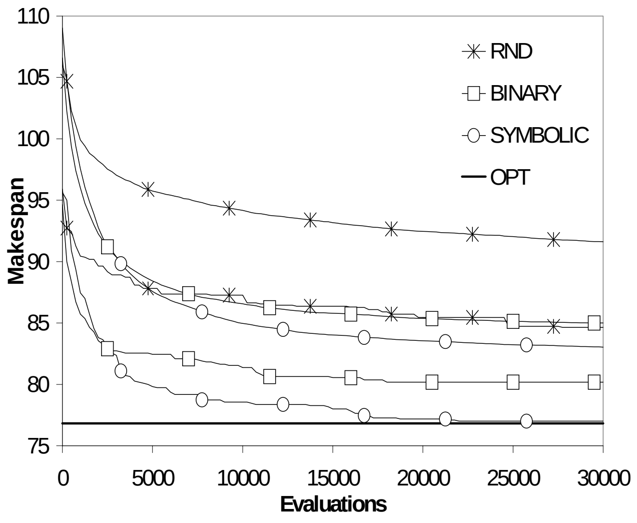

Fig. 7. Binary vs Symbolic.

7. It shows that symbolic representation obtains better results, as was to be expected.

4 Conclusions

A GA has been proposed to solve an assembly sequence problem, a much more difficult problem than other sequencing problems that have already been tackled using similar techniques, such as TSP and JSSP. It involves the selection of assembly operations that will form the assembly plan, in addition to their optimal arrangement. Two ways of representing a solution have been studied in applying GAs: a binary encoding based in numbering all the possible solutions, and a symbolic one that represents an ordered sequence of tasks. The results obtained show that the symbolic representation with the specific genetic operators used is more adequate for finding a better solution than conventional crossover and mutation operators working with the binary encoding.

References:

[1] Ames, A.L., T.L. Calton, R.E. Jones, S.G. Kaufman, C.A. Laguna and R.H. Wilson. Lessons Learned from a Second Generation Assembly Planning System. *Proc. 1995 IEEE Intl. Symp. on Assembly and Task Planning*, pp. 41-47.

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

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

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

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

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

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

[8] Kavraki, L., J.C. Latombe and R.H. Wilson. On the Complexity of Assembly Partitioning. *Information Processing Letters*. Vol. 48, 1993, pp. 229-235.

[9] Kavraki, L. and M. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*. Vol. 55, 1995, pp. 156-165.

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

[11] Starkweather, T., S. McDaniel, K. Mathias, D. Whitley, C. Whitley. A Comparison of Genetic Sequencing Operators. In R.K. Belew and L.B. Booker, *editors*. *Proceedings of the Forth Intl. Conf. on Genetic Algorithms*, ICGA-91, pp. 69-76. Morgan Kaufmann, 1991.

[12] Syswerda, G. Schedule Optimization Using Genetic Algorithms. In L. Davis, editor. *The Handbook of Genetic Algorithms*, pp. 332-349. Van Nostram Reinhold, 1990.

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

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