A Genetic Algorithm for Assembly Sequence Planning

Carmelo Del Valle[1], Rafael M. Gasca[1], Miguel Toro[1], and Eduardo F. Camacho[2]

[1]Dept. Lenguajes y Sistemas Informáticos, Univ. Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{carmelo, gasca, mtoro}@lsi.us.es
[2]Dept. Ingeniería de Sistemas y Automática, Univ. Sevilla,
Camino de los Descubrimientos s/n, 41092 Sevilla, Spain
eduardo@cartuja.us.es

Abstract. This work presents a genetic algorithm for 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 Traveling 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. 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. 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. Furthermore, some problem-based heuristics have been used for generating the individuals in the population.

1 Introduction

Genetic Algorithms 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 [1] [2].

This paper presents an application of genetic algorithms 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 [3] [4] or, through planners working automatically from geometric and relational models [5] and from CAD models and other non-geometric information [6] [7].

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. In fact, this problem has been proved to be NP-complete [8].

An optimum assembly plan is now sought, selected from the set of all feasible assembly plans. Most approaches used for choosing an optimal one employ different kind of rules in order to eliminate assembly plans including difficult tasks or awkward intermediate subassemblies [9] [10].

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 takes into account the information about each assembly task (robot and tool needed and estimation of assembly time) [11]. This approach allows applying the results to different stages of the process planning, from the design of the product and of the manufacturing system, to the final execution of the assembly plan.

The rest of the paper is organized as follows: Section 2 describes the problem of selection and sequencing of assembly tasks. The proposed genetic algorithm is described 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

And/or graphs have been used to represent the set of all feasible assembly plans for a product [12]. In this representation, 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 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. Another factor considered 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. $\Delta_{cht}(M, C, C')$ will denote the time needed for installing the tool C in the robot (machine) M if the tool C' was previously installed. Notice that any change of configuration in the robots can be modeled in this way.

Another issue is the transportation of parts and sub-assemblies, that could affect the total assembly time. Ideally, it would be supposed 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. But 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 subas-

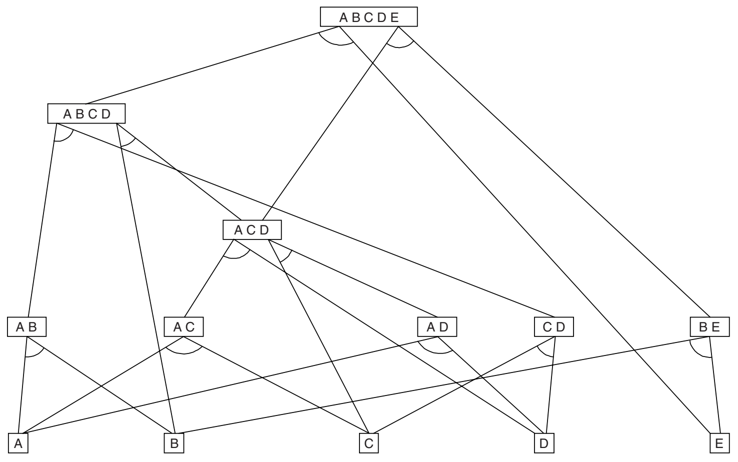

Fig. 1. *And/Or* graph of product ABCDE.

sembly. $\Delta_{mov}(SA, M, M')$ will denote the time needed for transporting the subassembly *SA* from robot *M* to robot *M'*.

3 The Genetic Algorithm

Genetic Algorithms (GAs) have been used to solve a large variety of combinatorial optimization problems with some success [1] [2]. 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. There will be little chance of constructing a new solution from significant parts of any two solutions. It will be necessary to add (probably not a few) other tasks to complete the solution, and delete some others.

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.

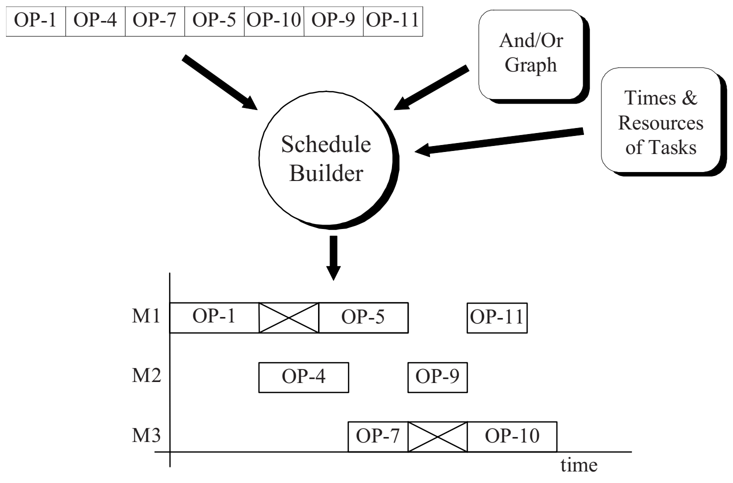

Fig. 2. The Schedule Builder.

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, such as TSP and JSSP, in the literature [1] [2], 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.

3.1 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 Re-Planning (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. This is done in different ways, depending on the distance of the search for a predecessor task of the selected task in the And/Or graph.

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.

3.3 A heuristic for generating solutions

An estimation of the time needed for the execution of each task and their successors in the And/Or graph have been used in order to generate the solutions in the initial population. An optimistic way of doing this estimation is done by the heuristic function *ht*, defined by the equations below:

$$ht(T) = dur(T) + \max\left(\min_{T_i \in Or1}(htc(T_i, T)), \min_{T_j \in Or2}(htc(T_j, T))\right) \quad \textbf{(1)}$$

$$htc(T_i, T) = ht(T_i) + \max\left(\tau\left(T_i, M(T), C(T)\right), \Delta_{mov}\left(sa(T_i), M(T_i), M(T)\right)\right) \quad \textbf{(2)}$$

$$\tau(T, M, C) = \begin{cases} \Delta_{cht}\left(M, C(T), C\right) & \text{if } M = M(T) \\ \max\left(0, \tau_1(T, M, C)\right) & \text{if } M \neq M(T) \end{cases} \quad \textbf{(3)}$$

$$\tau_1\left(T, M, C\right) = \max\left\{\min_{T_i \in Or_1}\left(\tau_2\left(T, T_i, M, C\right)\right), \min_{T_j \in Or_2}\left(\tau_2\left(T, T_j, M, C\right)\right)\right\} \quad \textbf{(4)}$$

$$\tau_2\left(T, T_i, M, C\right) = \tau\left(T_i, M, C\right) - \left(ht(T) - ht(T_i)\right) \quad \textbf{(5)}$$

This calculations for *ht*(*T*) take into account not only the duration *dur*(*T*) of each task *T* and the precedence constraints defined in the And/Or graph for the tasks, but also the delays corresponding to the change of tools (configurations) in the machines and to the transportation of intermediate sub-assemblies between different machines.

A solution is built by ordering the tasks of a tree of the And/Or graph. In order to construct a tree, an And node is selected randomly from the set of alternative task for each Or node, so that the probability of selection is inverse to the value of *ht* for the tasks. The sequence of tasks is formed selecting consecutively a task which its immediate predecessor have been introduced in the sequence. When a set of candidate tasks may be selected, a probability directed proportional to *ht* is used for a random selection, so that the tasks (and its successors) with more estimated time will have more chance of being selected.

4 Experiments and Results

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 an unique schedule.

The values corresponding to the higher part of figures 3 and 4 represent the average of 25 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).

Figures 3 and 4 show 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 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 influence of having an initial population that have been generated randomly or using any other informed method can be seen comparing the two figures. Figure 3 shows how the GA works when starting from a random initial population, and figure 4 when the initial population is generated using the heuristic *ht* presented in section 3.3. The curve RND in figure 4 corresponds to the use of *ht* in a probabilistic algorithm that continuously generates new solutions, which substitutes the random algorithm used in figure 3. We can see that the probabilistic algorithm improves the random one, and the average best solutions are near the solutions obtained by some of the genetic operators working alone. The results are better in general in figure 4, but the improvements are different for each operator. For example, RP-C operators working alone obtain similar results, RP-M improves slightly, and ROT operators improves more, because of the influence of the initial population in his behavior. The GA with all the operators working together also improves the solution, but only a little for the best trial.

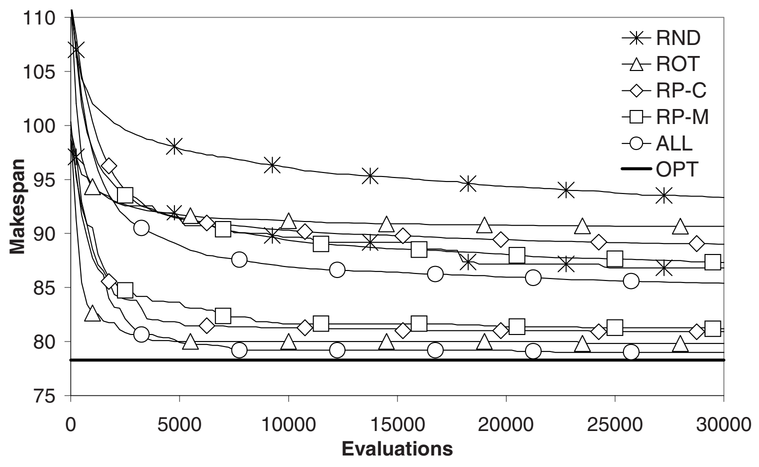

Fig. 3. Results for random initial populations.

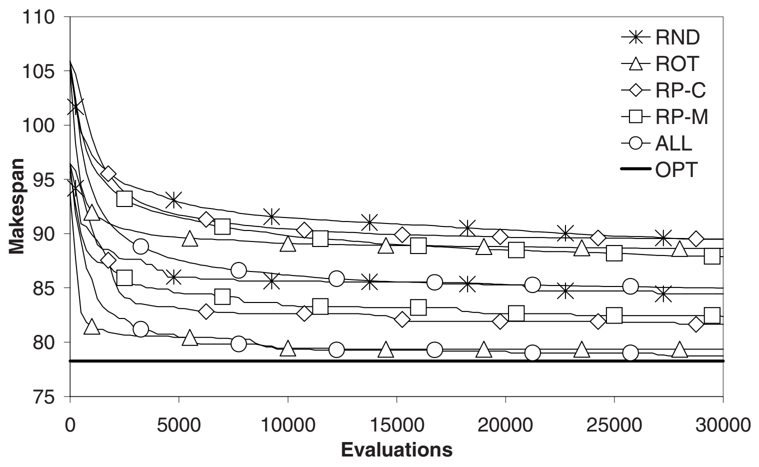

Fig. 4. Results for heuristic initial populations.

The use of more heuristic methods, such as the heuristic presented, but also during the search, is expected to improve the GA, so that a natural extension of this work must be in that sense.

5 Conclusions

A genetic algorithm has been presented for solving the assembly sequence planning problem, a much more difficult problem than other sequencing problems that have already been tackled using similar techniques, such as TSP and JSSP, because it involves also the selection of assembly operations that will form the assembly plan from a set of alternatives. Two families of genetic operators have been used in order to search through the whole solution space. RP operators have been proposed for generate new assembly plans (with different assembly tasks) from others. On the other hand, there is the ordering of assembly tasks in the sequence to obtain a good schedule. ROT mutation operators have been used for this goal.

Although the genetic information used by these operators could seem insufficient, the combined operation of them has been quite satisfactory. The behavior of the GA algorithm has been improved by using a initial population that have been generated using a heuristic corresponding to an estimation of the time needed for executing each task and its successors in the And/Or graph of the product to be assembled.

References

1. T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley (1991). A Comparison of Genetic Sequencing Operators. *Proceedings of the Forth Intl. Conf. on Genetic Algorithms*, ICGA-91, pp. 69-76. Morgan Kaufmann.
2. G. Syswerda (1990). Schedule Optimization Using Genetic Algorithms. In L. Davis, ed.. *The Handbook of Genetic Algorithms*, pp. 332-349. Van Nostram Reinhold.
3. Bourjault, A. (1984). *Contribution à une Approche Méthodologique de l'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires*. Thèse d'état, Université de Franche-Comté, Besançon, France.
4. De Fazio, T.L. and D.E. Whitney (1987). Simplified Generation of All Mechanical Assembly Sequences. *IEEE J. Robotics and Automat.*, Vol. 3, No. 6, pp. 640-658. Also, Corrections, Vol. 4, No. 6, pp. 705-708.
5. L.S. Homem de Mello and A.C. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Trans. Robotic and Automation*.Vol 7(2), 1991, pp. 228-240.
6. B. Romney, C. Godard, M. Goldwasser, G. Ramkumar (1995). An Efficient System for Geometric Assembly Sequence Generation and Evaluation. *Proc. 1995 ASME International Computers in Engineering Conference*, pp. 699-712.
7. T. L. Calton. Advancing design-for-assembly. The next generation in assembly planning. *Proc. 1999 IEEE Int. Symp. on Assembly and Task Planning*, pp. 57-62.
8. Wilson, R.H., L. Kavraki, T. Lozano-Pérez and J.C. Latombe (1995). Two-Handed Assembly Sequencing. *International Jour. Robotic Research*. Vol. 14, pp. 335-350.
9. Homem de Mello, L.S. and S. Lee, *eds.* (1991b). *Computer-Aided Mechanical Assembly Planning*. Kluwer Academic Publishers.
10. M.H. Goldwasser and R. Motwani (1999). Complexity measures for assembly sequences. *Intern. Joual of Computational Geometry and Applications*, 9:371-418.
11. C. Del Valle and E.F. Camacho (1996). Automatic Assembly Task Assignment for a Multirobot Environment. *Control Eng. Practice*, Vol. 4, No. 7, pp. 915-921.
12. Homem de Mello, L.S. and A.C. Sanderson (1990). And/Or Graph Representation of Assembly Plans. *IEEE Trans. Robotics Automat*. Vol. 6, No. 2, pp. 188-199.