A Pomset-Based Model for Estimating Workcells' Setups in Assembly Sequence Planning

Carmelo Del Valle, Miguel Toro, Rafael Ceballos, Jesús Aguilar

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

Abstract. This paper presents a model based on pomsets (partially ordered multisets) for estimating the minimum number of setups in the workcells in Assembly Sequence Planning. This problem is focused through the minimization of the makespan (total assembly time) in a multirobot system. The planning model considers, apart from the durations and resources needed for the assembly tasks, the delays due to the setups in the workcells. An A* algorithm is used to meet the optimal solution. It uses the *And/Or* graph for the product to assemble, that corresponds to a compressed representation of all feasible assembly plans. Two basic admissible heuristic functions can be defined from relaxed models of the problem, considering the precedence constraints and the use of resources separately. The pomset-based model presented in this paper takes into account the precedence constraints in order to obtain a better estimation for the second heuristic function, so that the performance of the algorithm could be improved.

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 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, i.e. the structure of the graph of connections. In fact, this problem has been proved to be NP-complete [1]. The identification of assembly tasks is carried out analyzing the graph of connections and taking into account the geometry of parts and the properties of contacts between parts [2] [3].

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 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 [4].

The criterion followed in this work is the minimization of the total assembly time (*makespan*) in the execution of the plan in a multirobot system, supposed the estimations of the durations of each possible task as well as of the necessary resources to carry out them [5]. This approach allows using the results 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.

The rest of the paper is organized as follows: Section 2 describes the problem of assembly sequence planning and the model used. Section 3 shows the proposed A* algorithm, as well as the two basic heuristics taken from relaxed models of the problem. The pomset-based model for estimating more accurately the minimum number of setups in the assembly machines is exposed in Section 5. Section 6 presents some of the results obtained, and 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 partially ordered assembly tasks. 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 having the property of being able to 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.

And/Or graphs have been used for a representation of the set of all feasible assembly plans for a product [6]. The *Or* nodes correspond to sub-assemblies, the top node coinciding with the whole assembly, and the leaf nodes with 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.

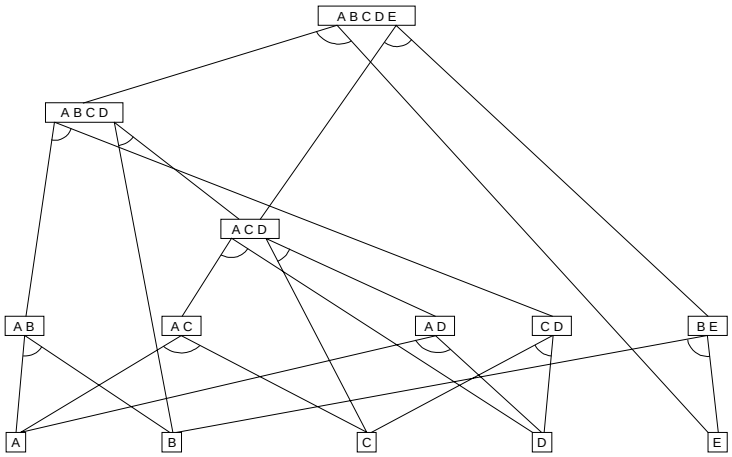

Fig. 1. *And/Or* graph of product ABCDE.

The problem is focused on searching an optimal assembly sequence, i.e. 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 of the durations and resources (robots, tools, fixtures…) needed for each assembly task in the *And/Or* graph. Another factor taken into account here, is the time necessary for changing 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. $\Delta_{cht}(R,T,T')$ will denote 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.

3 Algorithm description

An algorithm based on the A* search [7] has been developed, which has two well-differentiated parts: one of them studies the sequential execution of assembly tasks, and 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 heuristics showed below refer to this last part.

Because of the set-up of the *And/Or* graph, the assembly problem can be studied, starting from the final situation and going towards the initial one. This way, we will solve the opposite problem, that of disassembly, but composed by assembly tasks, so that the solution of the original problem can be made reversing the solution obtained by the algorithm.

Heuristic functions based on the execution of tasks taking only from the part of the tree below the node, and the time remaining for the use of tools and robots (supposing the minimum number of tool changes, in order to maintain the algorithm as A*) has been used in order to expand the minimum number of nodes and avoid redundant nodes.

3.1 Sequential Execution of Tasks

An A* algorithm to search for the global assembly plan can be implemented in the following way. Beginning with an initial node whose state represents the complete assembly realization, and therefore corresponds to the root node of the *And/Or* graph (complete assembly), all its possible successors are generated, whose states will represent the execution at the end of the assembly process of the tasks corresponding to the *And* nodes coming from the root node of the *And/Or* graph.

Two types of nodes may be generated, depending on the destination *Or* nodes of each chosen *And* node. If at least one of these *Or* nodes corresponds to an individual part, the assembly process will continue to be sequential, and the node resulting from the expansion may be treated as the initial node, where the node corresponding to the non-trivial sub-assembly will take the place of the root node.

If, on the other hand, the application of the task starts from two sub-assemblies each with various parts, in the resulting plan (or plans in general) the task arrangement is not

totally specified (various possible sequences exist for each assembly plan), or tasks may be carried out in parallel. There is also an interdependence amongst the sub-assemblies, because they potentially use the same set of resources. The treatment of this type of nodes has therefore to be undertaken in a different way from those corresponding to sequential task execution, and this will connect with the second part of this algorithm.

The evaluation function used for the nodes generated in this part is

$$f(n) = g(n) + h(n) \quad (\mathbf{1})$$

$g(n)$ being the time accumulated in the execution of tasks corresponding to the state of node n, including the delays in the necessary tool changes and in the transportation of intermediate subassemblies, and $h(n)$ being an optimistic estimation of the remaining time in which to complete the global process. ($h(n)$ should be a lower bound of the remaining time for the algorithm to be A*.) Due to the fact that various different plans (and therefore different task sets which would complete the assembly process) may be reached from node n, a detailed study would be computationally costly, and therefore

$$h(n) = durMin \cdot \lfloor \log_2 (a(n)+1) \rfloor \quad (\mathbf{2})$$

has been chosen, $a(n)$ being the number of tasks necessary to complete the assembly plan, and *durMin* the minimum duration of tasks. As can be seen, it is also impossible to determine the minimum number of tool changes without a detailed study, and therefore when estimating $h(n)$ it is assumed to be zero.

All the assembly trees (task precedence trees) are obtained for the "parallel" nodes, and are studied separately. The function $h(n)$ corresponding to each tree is defined in the following subsection.

3.2 Parallel Execution of Tasks

The objective of this part of the algorithm is to determine the total minimum time for the execution of the precedence trees obtained in the previous section. In order to do this, an A* algorithm is again used. The nodes of the expansion tree now present partial information about the execution of the assembly process. Concretely, at each expansion step only one assembly task is introduced, and its processing time will affect only one of the workstations, the same state being retained by the other workstations.

The state corresponding to a node of the expansion tree is represented by using the tasks available for introduction in the state of the next step, termed "candidates", and their earliest starting times, denoted $est(J_i)$. At the same time, the last tool used is included for each robot, denoted $lastTool(R_j)$, as well as the final time of use, denoted $lastTime(R_j)$.

The evaluation function for the nodes obtained by this algorithm is similar to (1), being now $g(n)$ the largest of the earliest starting times of $cand(n)$, the set of candidates, and the final times of the already finished in n without successors. The function $h(n)$ must be an optimistic estimation of the time remaining, taking into account the slacks between $g(n)$ and the different times describing the state of n. Two different heuristic functions have been defined in this work, taken from relaxed models of the problem in which some constraints have been disregarded.

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 as follows:

$$h_1(n) = \max\left(0, \max_{J_i \in cand(n)} \left(h_1(J_i) - e(n, J_i)\right)\right) \quad (3)$$

where

$$e(n, J_i) = g(n) - est(n, J_i) \quad (4)$$

$$h_1(J) = dur(J) + \max_{J_i \in suc(J)} \left(h_1(J_i) + \tau\left(J_i, R(J), T(J)\right)\right) \quad (5)$$

$$\tau(J, R, T) = \max\left(0, \max_{J_i \in suc(J)} \left(h_1(J_i) + \tau\left(J_i, R(J), T(J)\right) - h_1(J)\right)\right) \quad (6)$$

In the above expressions, n is an expansion node, J is an assembly task, and $e(n, J)$ is the existing time slack. $R(J)$ and $T(J)$ are the robot and tool necessary for the execution of task J, and $dur(J)$ is its duration. $\tau(J, R, T)$ is the added delay, due to the fact that the tool T is being used by robot R in task J and successors, because of the necessary tool changes. The equation (6) defines $\tau(J, R, T)$ when $R \neq R(J)$. In the case $R = R(J)$, $\tau(J, R, T)$ is defined as $\Delta_{cht}(R, T(J), T)$ (that could be zero if $T = T(J)$).

Notice that $h_1(J)$ does not depend on the expansion nodes, and thus allows one to calculate a lower bound prior to using the A* algorithm.

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:

$$h_2(n) = \max_{robots} \left(h_2(n, R_i) - e(n, R_i)\right) \quad (7)$$

where

$$e(n, R) = g(n) - lastTime(n, R) \quad (8)$$

and $h_2(n, R_i)$ 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:

$$h_2(n, R) = \left(\sum_{H_j \in R} \sum_{J_i \in cand(n)} h_2(J_i, T_j)\right) + \sum\left(n, \Delta_{cht}(R)\right) \quad (9)$$

with $h_2(J, T)$ the remaining time of usage of tool T by task J and its successors. The term $\sum\left(n, \Delta_{cht}(R)\right)$ 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:

$$\sum\left(n, \Delta_{cht}(R)\right) = N_{cht}(n, R) \cdot \Delta_{cht}(R) \quad (10)$$

where $N_{cht}(n, R)$ is the number of tool changes needed in R for the remaining tasks for completing the assembly form n, and $\Delta_{cht}(R)$ is the duration of a simple tool change in R. Without considering any precedence information, an in order to maintain the ad-

missibility of the heuristic, it must be supposed that the remaining tools will be installed only once in the estimation of $N_{cht}(n, R)$.

4 The Pomset-Based Model

The heuristic h_2 does not consider any information about precedence of tasks, and, in order to meet an admissible heuristic, it must be supposed that every tool will be installed once, so that the number of tool changes is minimized. The heuristic h_3 is based on h_2, but it estimates more accurately the number of tool changes. For that, a pomset-based model is used that reflects some of the precedence constraints of the problem.

Given a precedence tree of tasks, the tasks using the same machine must be done sequentially. For each sequence of tasks using the same machine, it can be defined a sequence of tools, in the same order that that of the tasks which use them. Because we are only interested in the tool changes, we must transform that sequence of tools in another one substituting consecutive repetitions of the same element for an instance of it. These sequences will be denoted as *sequences of tools*.

The objective is to find, for each robot, the minimum number of times that each tool must be installed, but estimated in the more realistic way possible. Figure 2 shows a simple example: for the precedence tree of tasks (a), the precedence tree of usages of tools in the robot R1 is shown (b); in order to minimize the number of tool changes, T1 must be installed once, so that (c) represents the constraint precedence graph of tools, each node corresponding to possibly more than one consecutive usage of the tool. Finally, (d) represents the *pomset*[1] corresponding to a simplified model of (c), maintaining only the precedence constraints referring to the tool in the top node.

The data structure pomset was introduced by Pratt [8] in the specification of concurrent programs. An alignment is a sequence of the elements of the pomset satisfying the ordering constraints. In our pomset-based model, not all the possible alignments from the pomset will correspond to valid sequences of tools, because of the constraints that have been disregarded. For example, the pomset in Figure 2 (d) has an alignment {T2,T3,T2,T1}, which does not correspond to any possible sequences of tool formed from (c). However, this in not the main purpose, but the number of tool changes, which is related to the cardinality of the pomset.

According to the previous comments, the pomset can be defined as a tuple $<B,F>$, B being a multiset containing the same elements (and multiplicity) as the optimal sequence of tools taken from the precedence tree. On the other hand, F denotes the set of tools which can appear the first in any optimal sequence.

Two basic operations are needed in the recursive definition of the pomsets (one for each robot) for a precedence tree. The first operation, denoted push, allows adding one element to the pomset, proceeding from the root node. The other operation, denoted merge, is about merging the pomsets from the subtrees.

The pomset $<B,F>$ corresponding to a precedence tree whose root node is the task J and robot R, denoted $P(J, R)$, is given by

[1] *pomset* = partially ordered multiset. A multiset is a set in which the elements can be repeated.

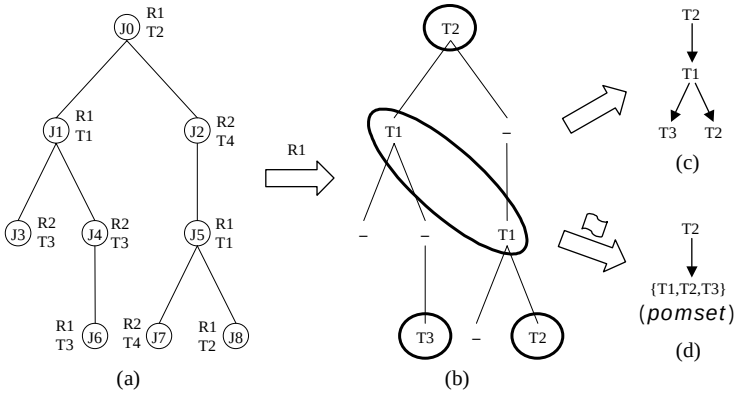

Fig. 2. An example of a pomset from the model.

$$P(J,R) \equiv \begin{cases} <\{T(J)\},\{T(J)\}> & \text{if } suc(J)=\varnothing \\ push(T(J),P(J_1,R)) & \text{if } suc(J)=\{J_1\} \\ push(T(J),merge(P(J_1,R),P(J_2,R))) & \text{if } suc(J)=\{J_1,J_2\} \end{cases} \quad (R=R(J)) \tag{11}$$

if R is the robot used by J. If robot R is not the used by J, then the pomset is defined by

$$P(J,R) \equiv \begin{cases} \varnothing & \text{if } suc(J)=\varnothing \\ P(J_1,R) & \text{if } suc(J)=\{J_1\} \\ merge(P(J_1,R),P(J_2,R)) & \text{if } suc(J)=\{J_1,J_2\} \end{cases} \quad (R\neq R(J)) \tag{12}$$

The operation *push* is defined by

$$push(x,<B,F>) \equiv \begin{cases} <B,\{x\}> & \text{if } x\in F \\ <B \uplus \{x\},\{x\}> & \text{if } x\notin F \end{cases} \tag{13}$$

that shows that the set of first elements must contain the element pushed only, and that and instance of this element must be added (operator $\uplus$, multiset additive union) when the element does no belong to the set of first elements of the original pomset, reflecting the corresponding additional change tool.

The operation *merge* is defined by

$$merge(P_1,P_2) \equiv \begin{cases} <B_1 \cup B_2, D_F(P_1,P_2)> & \text{if } D_F(P_1,P_2)\neq\varnothing \\ <B_1 \cup B_2, F_1 \cup F_2> & \text{if } D_F(P_1,P_2)=\varnothing \end{cases} \tag{14}$$

where
$$D_F(P_1,P_2) \equiv (F_1 \cap F_2) \cup D'_F(P_1,P_2) \cup D'_F(P_2,P_1) \tag{15}$$

and
$$D'_F(P_i,P_j) \equiv \{x\in F_i \mid \#(x,B_i) > \#(x,B_j)\} \tag{16}$$

The multiset union operator is defined taking the maximum multiplicity of each element, so that is the adequate operator for the resultant multiset from the merge of the pomsets. The resultant set of first elements is formed according to the concept of *dominance*. Figure 3 shows two cases. The symbol $\otimes$ represents the merge operator,

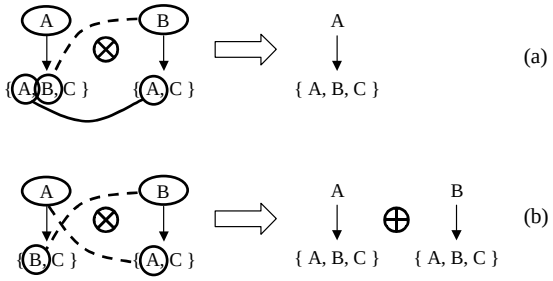

Fig. 3. An example of dominance.

and $\oplus$ is used for expressing the different possibilities we can obtain. In (a), we can obtain a pomset without increasing the number of instances of each element, because the element A in F_1 has more instances in B_1 than in B_2, so we say that A is dominant. Moreover, B is not dominant because it cannot appear as first element in any optimal sequence of tools, i.e. without increasing the number of instances. In this situation, $D_F(P_1,P_2)$ is not empty, so we can determine the set of first elements without any problems. In case (b), A and B are not dominants, but the set of first elements cannot be empty. Notice that the optimal sequences can be taken from two different pomsets, in which the number of instances for A and B are different, expressing that the first element in an optimal sequence can be A or B, and depending of which is choosed, it has an additional instance. We say that there is an indetermination in the number of instance of A and B. The model proposed in this work makes a simplification about this aspect (see eq. (14)), so that the number of instances of A and B are not increased, and both appear as possible first element in an optimal sequence. This way, the resultant heuristic is admissible, but losing accuracy. On the other hand, if an element is present in the set of first elements in both pomsets, it must be present in the set of first elements in the resultant pomset.

A last operation must be defined for the pomset structure, for obtaining the number of tool changes. This operation is denoted as *ntrans*, and for the model proposed in this work is defined as the number of elements of the multiset less one:

$$ntrans(P) = \sum_{x \in B} \#(x,B) - 1 \tag{17}$$

Finally, the new heuristic h_3 is defined in the same way as h_2 (eq. (9)), but in the calculation of $\sum(n, \Delta_{cht}(R))$ (eq. (10)), the term $N_{cht}(n, R)$ is now estimated more accurately. If the candidate tasks in n are J_1, ..., J_m, and T is the last tool used in R,

$$N_{cht}(n,R) = ntrans\left(push\left(T, merge\left(P(J_1,R),...,P(J_m,R)\right)\right)\right) \tag{18}$$

5 Results

The algorithm has been tested in a variety of situations, considering different product structures (number of parts, number of connections between parts), different types of

And/Or graphs (number of sub-assemblies, number of assembly tasks for each sub-assembly), and different assembly resources (number of robots, number of tools).

The results in Tables 1-4 correspond to a hypothetical product of 30 parts, with 396 *Or* nodes and 764 *And* nodes in the *And/Or* graph. The number of linear sequences is about 10^{21}. The tables show the effect of having more or less resources for assembling the product in the performance of the algorithm. The results refer to 10 different combinations of durations and resources for assembly tasks.

The heuristic functions defined present two different effects in calculating $h(n)$. The estimation made from h_1 is due to the most unfavorable candidate task. In the other hand, h_2 and h_3 shows an additive effect, because of the uses of robots by all candidate tasks. Therefore, two new heuristic functions can be defined from the com-

Table 1. Results for 2 machines and 2 tools/machine

Heuristic	Nodes visited			Time (ms)			N-Pr	N-F	%
	Ave	Max	Min	Ave	Max	Min			Error
h_1	41492	89189	2520	19429	30590	180	4	6	0,990
h_2	9316	42775	32	1422	6420	0	5	0	0,000
h_3	617	3019	32	176	870	0	2	0	0,000
$max(h_1, h_2)$	16385	71585	32	4291	30050	0	4	1	0,248
$max(h_1, h_3)$	3422	30506	32	3257	31470	0	1	1	0,248

Table 2. Results for 2 machines and 4 tools/machine

Heuristic	Nodes visited			Time (ms)			N-Pr	N-F	%
	Ave	Max	Min	Ave	Max	Min			Error
h_1	40093	56108	3020	25025	30930	710	2	8	2,648
h_2	5311	19398	267	790	4230	50	1	0	0,000
h_3	1458	6120	32	198	930	0	1	0	0,000
$max(h_1, h_2)$	5078	19009	387	1143	4450	60	1	0	0,000
$max(h_1, h_3)$	1303	4805	32	418	1870	0	1	0	0,000

Table 3. Results for 4 machines and 2 tools/machine

Heuristic	Nodes visited			Time (ms)			N-Pr	N-F	%
	Ave	Max	Min	Ave	Max	Min			Error
h_1	16905	85540	32	2357	11700	0	1	0	0,000
h_2	2751	7195	32	263	710	0	2	0	0,000
h_3	1781	7011	32	533	2030	0	3	0	0,000
$max(h_1, h_2)$	804	2098	32	99	220	0	2	0	0,000
$max(h_1, h_3)$	794	2098	32	148	330	0	2	0	0,000

Table 4. Results for 4 machines and 4 tools/machine.

Heuristic	Nodes visited			Time (ms)			N-Pr	N-F	%
	Ave	Max	Min	Ave	Max	Min			Error
h_1	22697	93376	32	6084	30530	0	4	1	0,302
h_2	1808	5907	128	197	660	0	1	0	0,000
h_3	1072	5014	119	396	2260	0	1	0	0,000
$max(h_1, h_2)$	1765	5907	32	278	980	0	2	0	0,000
$max(h_1, h_3)$	1062	5014	119	396	2260	0	1	0	0,000

bination of both effects, taking the most realistic estimation, $\max(h_1(n), h_2(n))$ and $\max(h_1(n), h_3(n))$.

The use of A* algorithms presents some problems. The most important is the storage space that could be occupied. 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 in order to detect symmetries, so that redundant nodes are avoided.

Apart from the number of nodes visited and execution times, the tables show 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.

6 Conclusions

A model for the selection of optimal assembly sequences for a product in a generic multirobot system has been presented. The objective of the plan is the minimization of the total assembly time. To meet it, 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.

A pomset-based model has been proposed for estimating the number of workcells' setups needed in Assembly Planning. The model is used in the definition of a heuristic function for an A* algorithm. The result is a more informed heuristic than the one which was based on, due to the combination of two kinds of constraints, the precedence constraints and the use of the resources.

References

1. R.H. Wilson, L. Kavraki, T. Lozano-Pérez and J.C. Latombe. Two-Handed Assembly Sequencing. *Int. Jour. of Robotic Research*. Vol. 14, pp. 335-350, 1995.
2. 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.
3. T. L. Calton. Advancing design-for-assembly. The next generation in assembly planning. *Proc. 1999 IEEE International Symposium on Assembly and Task Planning*, pp. 57-62, Porto, Portugal, July 1999.
4. M. H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *Int. Journal of Computational Geometry and Applications*, 9:371-418, 1999.
5. C. Del Valle and E.F. Camacho. Automatic Assembly Task Assignment for a Multirobot Environment. *Control Eng. Practice*, Vol. 4, No. 7, 1996, pp. 915-921.
6. L.S. Homem de Mello and A.C. Sanderson. *And/Or* Graph Representation of Assembly Plans. *IEEE Trans. Rob. Automation*. Vol. 6, No. 2, pp. 188-199, 1990.
7. J. Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Reading, MA, Addison-Wesley, 1984.
8. V.R. Pratt. Modeling Concurrency with Partial Orders. *Int. Journal of Parallel Programming*. Vol. 15, No. 1, pp. 33-71, Feb 1986.