Combining Heuristics in Assembly Sequence Planning

Carmelo DEL VALLE[1], Miguel TORO[1], Eduardo F. CAMACHO[2], Rafael M. GASCA[1]
[1]*Dept. Lenguajes y Sistemas Informáticos, Univ. Sevilla, Spain*
[2]*Dept. Ingeniería de Sistemas y Automática, Univ. Sevilla, Spain*

Abstract. Assembly Sequence Planning is tackled by modelling and solving a planning problem that considers the execution of the plan in a system with multiple assembly machines. The objective of the plan is the minimization of the total assembly time (makespan). To meet this objective, the model takes into account the durations and resources for the assembly tasks, the change of configuration in the machines, and the transportation of intermediate subassemblies between different workstations. In order to solve the problem, different heuristics has been defined from two relaxed model of it, one considering only the precedence constraints among tasks, and the other one considering only the use of shared resources. From these basic heuristics, other ones have been defined, combining both types of information from the problem, so that the refinement produces substantial improvements over the initial heuristics.

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 issue 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 system with multiple machines [11]. To meet this

objective, a scheduling-based model is used [12], which takes into account all factors having an effect on the makespan: an estimation of the duration of tasks; the resources used for them (machines and configurations); the times needed for changing tools in the robots; and the delays due to the transportation of intermediate subassemblies between different workstations.

The rest of the paper is organized as follows: Section 2 describes the assembly sequence planning problem and the model proposed. The details of the A* algorithm are described in Section 3. Section 4 presents the basic heuristics taken from two relaxed models of the problem, and Section 5 shows how these heuristics are combined in order to improve their estimations. Some comparative results when using the different heuristics are shown in Section 6, 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 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. This works supposes that in an assembly task there are two initial sub-assemblies to form a final one. 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 [7] is a representation of the set of all assembly plans for a product. The *Or* nodes correspond to sub-assemblies, the top node corresponding to the whole assembly, and the leaf nodes to the individual parts. Each *And* node corresponds to the assembly task joining the sub-assemblies of its two Or nodes below it producing the sub-assembly of the Or node above it. An And/Or graph consists on several trees whose top nodes are the top node of the And/Or graph and whose leaf nodes are the leaf nodes of the And/Or graph. Each tree 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, where Or nodes are represented as rectangles, and And nodes are represented as hyperarcs.

This work is about the selection of the best assembly plan, that is, one of the And/Or

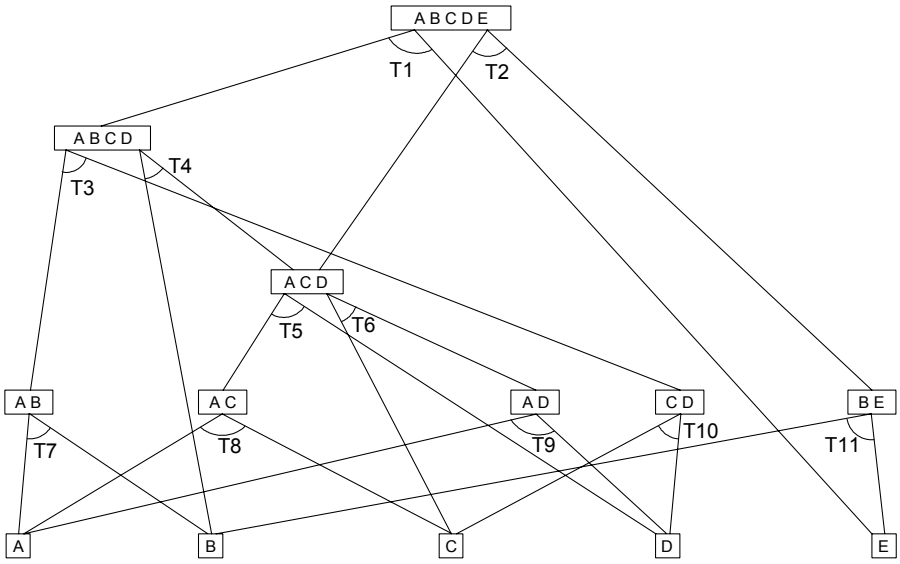

Figure 1. And/Or graph of product ABCDE

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.

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 order to estimate 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 are eliminated from the And/Or graph, as the sub-assemblies which cannot be part of a solution.

An assembly plan can be defined by means of the assembly tasks that form the successive sub-assemblies until the final product is made. An assembly task is defined from the initial and final sub-assemblies involved. An assembly task is defined to be performed in an assembly machine with a determined configuration, and has an estimated duration. If there are different ways (machine-configuration-duration) of joining two sub-assemblies to form another larger sub-assembly, we refer to those as different assembly tasks, which would correspond to additional And nodes in the And/Or graph.

The selection of the assembly plan is made through evaluating the optimal sequencing of their tasks, so that we would be solving at the same time a planning and scheduling problem. In order to evaluate more precisely the cost of the solutions, i.e. the total assembly time, other factors have been taken into account: the change of configurations in the machines and the transportation of subassemblies between different machines. The corresponding tasks (actions) are easily generated and sequenced from the set of assembly tasks defined by the sub-assemblies involved in them: for each two successive assembly tasks executed in a machine using different configurations there will be an adequate *change of configuration task* on the machine between them; if a sub-assembly is generated by an assembly task in a machine and it is required to be used as an initial sub-assembly by another assembly task in another machine, there will be a *transportation task* of this sub-assembly from one machine to the other one between the execution of the two assembly tasks.

Since assembly plans and assembly sequences are defined using only the assembly tasks, some delays are used for modelling both auxiliary tasks: $\Delta_{cht}(M, C, C')$ denotes the time needed for changing the configuration of the machine M from C to C'; and, on the other hand, $\Delta_{mov}(SA, M, M')$ denotes the time needed for transporting the subassembly SA from machine M to machine M'.

The model proposed supposes a well-dimensioned system, with a perfect planning when executing the assembly plan, so that, when a part would be required in a machine for executing an assembly task, it will be present there. The same cannot be guaranteed for an intermediate sub-assembly, because it could be built in a machine and required immediately in another one to form another subassembly.

In order to an easier reasoning, we will suppose in the rest of the paper 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 algorithm.

As mentioned above, with this model, the choice is not limited to the assembly plan, but also it can be specified when each assembly task is to be carried out in order to minimize the makespan (some assembly 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.

3. The A* Algorithm

An algorithm, based on the A* search [13], has been developed to solve the problem stated in the previous section. 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 algorithm starts from the root of the And/Or graph. The sequential part of the algorithm is used while the tasks considered involves only one non-trivial sub-assembly below the corresponding And node. It is the case of tasks T1 and T4 in Figure 1. When an assembly task takes two non-trivial sub-assemblies (for example, task T2 in Figure 1), 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 below 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 previously 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 assembly tasks that have been included into the solution in the previous steps (with the corresponding auxiliary tasks –see Section 2). The order of including assembly tasks specifies the order of execution of them. So, an assembly task will not be included until all its predecessor assembly tasks in the And/Or graph have been included. This strategy allows verifying the precedence constraints of the problem. The state of a node n can be obtained also through the set of assembly tasks $cand(n)$ that can be included in the next expansion step, denoted *candidates*. For each candidate assembly task T we take the earliest start time, $est(T)$, 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 machine M due to the tasks that have been included, $lastTime(n, M)$, and the last configuration used in each machine, $lastConf(n, M)$. An expansion step of the algorithm corresponds to selecting a candidate assembly task T, and including it in the partial solution as it is executed starting at $est(T)$, recalculating the state for the successor node and including in the set of candidate tasks the successor assembly tasks of T.

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:

$$g(n) = \max\left(\max_{T_i \in cand(n)} \left(est(n, T_i) \right), \max_{M_i \in machines} \left(lastTime(n, M_i) \right) \right) \quad (1)$$

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, $e(n, T) = g(n) - est(n, T)$, and another one for the machines, $e(n, M) = g(n) - lastTime(n, M)$.

4. Basic Heuristic Functions

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

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

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

$$hs(T) = dur(T) + \max\left(\min_{T_i \in Or_1(T)} \left(hs(T_i) \right), \min_{T_i \in Or_2(T)} \left(hs(T_i) \right) \right) \tag{3}$$

where $dur(T)$ is the duration of task T and $Or_1(T)$ and $Or_2(T)$ are the Or nodes corresponding to the initial sub-assemblies involved in task T. In these expressions, $T \in Or$ represents the tasks T immediately below the Or node in the And/Or graph.

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, for each sub-problem all tasks are defined, i.e. there are not alternative tasks, and then the amount of usages 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.

4.1 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:

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

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

$$\tau_{mov}\left(T_i, T \right) = \max\left(\tau\left(T_i, M(T), C(T) \right), \Delta_{mov}\left(sa(T_i), M(T_i), M(T) \right) \right) \tag{6}$$

$$\tau(T, M, C) = \max\left(0, \max_{T_i \in suc(T)} \left(h_1(T_i) + \tau\left(T_i, M(T), C(T) \right) - h_1(T) \right) \right) \tag{7}$$

In the above expressions, $M(T)$ and $C(T)$ are the machine and configuration necessary for the execution of the assembly task T. $\tau(T, M, C)$ is the added delay, due to the fact that the configuration C is being used by machine M in task T and successors, because of the necessary changes in configuration. The equation (7) defines $\tau(T, M, C)$ when $M \neq M(T)$. In the case $M=M(T)$, $\tau(T, M, C)$ is defined as $\Delta_{cht}(M, C(T), C)$ (that could be zero if $C=C(T)$). Finally, $\tau_{mov}(T, T')$ is the delay considering the possible transportation of the intermediate subassembly generated between the execution of T and T', and that of the possible change of configurations.

Notice that $h_1(T)$ 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.

4.2 The heuristic function h_2: use of resources

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

$$h_2(n) = \max_{M_i \in machines} \left(h_2(n, M_i) - e(n, M_i) \right) \quad (8)$$

where $h_2(n, M_i)$ is the minimum time of use of machine M_i without considering the task precedence constraints. If each configuration is associated with only one robot, the calculation of $h_2(n, M)$ is equivalent to the traveling salesman problem, when considering the configurations used by the tasks that still have not been included in n as the cities and an origin corresponding to the last-used configuration in the machine M:

$$h_2(n, M) = \left(\sum_{C_j \in M} \sum_{T_i \in cand(n)} h_2(T_i, C_j) \right) + \sum \left(n, \Delta_{cht}(M) \right) \quad (9)$$

with $h_2(T, C)$ the remaining time of usage of configuration C by task T and its successors. The term $\sum \left(n, \Delta_{cht}(M) \right)$ refers to the time needed for the changes of configuration. In the usual case that times for change of configurations do not depend on the types of configuration, it can be calculated easily. Without any precedence information, an in order to maintain the admissibility of the heuristic, it must be supposed that each remaining configuration will be established only once.

5. Combination of heuristics

The heuristics defined in the last section take into account different elements of the problem: h_1 uses the precedence constraints taken from the *And/Or* graph in order to estimate the most unfavourable path from an assembly task to a leaf assembly task, supposing that tasks from different branches, i.e. not related by precedence constraints, can be executed independently, that is, ignoring if they use the same machine. In the other way, h_2 ignores the precedence constraints and calculates the total usage time for each machine. The two heuristics have different effects and are incomparable. Depending on the machines used and the structure of the *And/Or* graph, one of them can obtain a better estimation than the other one. For example, if there is only one machine, there is no parallel execution of tasks, and h_2 will obtain a more accurate estimation. In the other way, if each assembly ask is executed in a different machine, h_1 collects all the information about the problem and its estimation is accurate.

In the previous section h_1 and h_2 were defined related to n, the expansion node. But the nature of the two heuristics are different: $h_1(n)$ is the maximum of the estimations of the candidate tasks, and $h_2(n)$ adds the time of usage of machines for all the candidate tasks. In order to combine properly the information from the two heuristics, we will use their estimations referred to the candidate tasks. For h_1 we have $h_1(T)$ defined in equations (5) to (7). For h_2 we can define $h_2(T)$ in a similar way than in (9):

$$h_2(T) = \max_{machines} \left(h_2(T, M_i) \right) = \max_{machines} \left(\left(\sum_{C_j \in M_i} h_2(T, C_j) \right) + \sum \left(T, \Delta_{cht}(M_i) \right) \right) \quad (10)$$

where $\sum \left(T, \Delta_{cht}(M) \right)$ refers to the time needed for the changes of configuration in M for the execution of T and its successors.

5.1 Heuristic h_3: refining h_1 using h_2

As $h_2(T)$ could be a better estimation than $h_1(T)$, both being optimistic, the first could be taken into account into the calculation of the second, since in the recursive expression for $h_1(T)$ it is required an estimation of the time needed for the execution of all the tasks in the subtree below T. The new heuristic that it is obtained, h_3, is defined based on equation (5),

substituting h_1 for h_3 and considering h_2, as indicated. This way, a better estimation due to h_2 is propagated towards the predecessor tasks in the precedence tree. So, $h_3(T)$ is defined as:

$$h_3(T) = \max\left(dur(T) + \max_{T_i \in suc(T)} \left(h_3(T_i) + \tau_{mov}\left(T_i, T\right)\right), h_2(T)\right) \tag{11}$$

The definition of $h_3(n)$ is similar to that of $h_1(n)$, (equation (4)), resulting a more informed heuristic than $h_1(n)$. However, $h_3(n)$ does not invalidate $h_2(n)$, because the last considers all candidate tasks in the expansion node n.

5.2 Heuristic h_4: estimating the intervals of machine usages

In the definition of $h_3(T)$, the time of usage of the most unfavourable machine, given by $h_2(T)$, is supposed that can take place during the execution of all the tasks of the subtree whose root is T. In some cases, it could be determined, by an analysis of the precedence tree, that the interval of use is smaller, so that the estimation of the total time for the execution of all tasks in the tree could be improved even more. Two new variables are defined, δ_b and δ_e, indicating the intervals, at the start and the end respectively, of the execution of all the tasks in the tree, in which the machine is not used.

The calculation of δ_b is done using its recursive definition:

$$\delta_b(T, M) = \begin{cases} 0 & \text{if } M = M(T) \\ dur(T) + \min_{T_i \in suc(T)} \left(\delta_b(T_i, M) + \max\left(\Delta_{mov}(\cdot), \Delta'_{cht}(T_i, T)\right)\right) & \text{if } M \neq M(T) \end{cases} \tag{12}$$

where $\Delta_{mov}(\cdot) \equiv \Delta_{mov}\left(sa(T_i), M(T_i), M(T)\right)$ and $\Delta'_{cht}(T_i, T)$ represents the delay due to the possible change of configuration between the execution of T_i and T, defined by

$$\Delta'_{cht}(T_i, T) = \begin{cases} \Delta_{cht}\left(M(T), C(T_i), C(T)\right) & \text{if } M(T) = M(T_i) \\ 0 & \text{if } M(T) \neq M(T_i) \end{cases} \tag{13}$$

In order to obtain a consistent definition, we take $\delta_b(T, M) = \infty$ when $M \neq M(T)$ and $suc(T) = \varnothing$. This way, when a machine is not used by any task belonging to a precedence subtree, the value of δ_b will be infinite.

In the other way, δ_e is calculated by means of its recursive definition:

$$\delta_e(T, M) = \begin{cases} \min_{T_i \in suc(T)} \left(\delta_e(T_i, M), h_4(T) - dur(T)\right) & \text{si } M = M(T) \\ \min_{T_i \in suc(T)} \left(\delta_e(T_i, M)\right) & \text{si } M \neq M(T) \end{cases} \tag{14}$$

Again, for a consistent definition, we take $\delta_e(T, M) = \infty$ when $M \neq M(T)$ and $suc(T) = \varnothing$.

The definition of the new heuristic $h_4(T)$ has the same structure than $h_3(T)$, except that instead of using $h_2(T)$ alone, we take into account δ_b and δ_e. Moreover, it is necessary to separate the estimations of $h_2(T)$ for each machine, using $h_2(T, M)$:

$$\begin{aligned} h_4(T) = \max\Bigg(& dur(T) + \max_{T_i \in suc(T)} \left(h_4\left(T_i\right) + \tau_{mov}\left(T_i, T\right)\right), \\ & \max_{h_2(T, M_i) \neq 0} \left(h_2\left(T, M_i\right) + \delta_b\left(T, M_i\right) + \delta_e\left(T, M_i\right)\right)\Bigg) \end{aligned} \tag{15}$$

The definition of $h_4(n)$ is similar to that of $h_1(n)$, (equation (4)), resulting again a more informed heuristic than $h_1(n)$ and $h_3(n)$, but not necessarily better than $h_2(n)$, because the last considers all candidate tasks in the expansion node n.

5.3 Heuristic h_5: considering all machines in h_1, h_3 and h_4

In the definition of the heuristics h_1, h_3 and h_4 referring to a task T we have not taken into account the use of all the different machines, but only that of the machine used for the task T. In order to obtain this estimation we must define a new function related to the maximum delay that can be done in the use of each machine. Based on h_1, the new function, $\tau'(T, M, C)$, is defined as

$$\tau'(T,M,C) = \begin{cases} \Delta_{cht}\left(M, C(T), C\right) & \text{if } M = M(T) \\ \max\limits_{T_i \in suc(T)} \left(h_1(T_i) + \tau'\left(T_i, M, C\right) - h_1(T)\right) & \text{if } M \neq M(T) \end{cases} \tag{16}$$

and corresponds to the time before the execution of T at which the machine M must have configuration C in order to the execution of T and its successors does not finish after $h(T)$. Notice that this definition is similar to that of $\tau(T, M, C)$ (equation (7)), but now we can have negative values, indicating in that case a spare time for an eventual change of configuration that could be necessary, or simply that it is possible to have the configuration C in M after the execution of T has started, without altering the total execution time of T and its successors.

The same ideas used in defining the heuristics h_3 and h_4 can be employed in order to have a better estimation of τ' when $M \neq M(T)$, preserving the same expression when $M=M(T)$. This way, for h_3, $\tau'(T, M, C)$ can be defined, when $M \neq M(T)$, as

$$\begin{aligned} \tau'\left(T,M,C\right) = \max\Bigg(& \max_{T_i \in suc(T)} \left(h_3(T_i) + \tau'\left(T_i, M, C\right) - h_3(T)\right), \\ & h_2(T) + \Delta''_{cht}(T,M,C) - h_3(T) \Bigg) \end{aligned} \tag{17}$$

where

$$\Delta''_{cht}(T,M,C) = \begin{cases} 0 & \text{if } h_2(T,C) > 0 \\ \min\limits_{h_2(T,C_j)>0} \left(\Delta_{cht}(M,C,C_j)\right) & \text{if } h_2(T,C) = 0 \end{cases} \tag{18}$$

that shows that it will be needed a change of configuration if C is not used below T.

For h_4, $\tau'(T, M, C)$ can be defined, when $M \neq M(T)$, as

$$\begin{aligned} \tau'\left(T,M,C\right) = \max\Bigg(& \max_{T_i \in suc(T)} \left(h_4(T_i) + \tau'\left(T_i, M, C\right) - h_4(T)\right), \\ & h_2(T) + \delta_e(T,M) + \Delta''_{cht}(T,M,C) - h_4(T) \Bigg) \end{aligned} \tag{19}$$

The new heuristic $h_5(n)$ is defined as

$$h_5(n) = \max_{T_i \in cand(n)} \left(\max_{\text{machines}} \left(h\left(T_i\right) + \tau'\left(T_i, M_j, lastConf\left(n, M_j\right)\right) - e\left(n, M_j\right)\right)\right) \tag{20}$$

where h refers to the heuristic used in the definition of τ', giving three different heuristic functions, named h_{51}, h_{53} and h_{54}.

6. Comparative results

There are different factors that affect the complexity of the problem proposed. Some of the more important are the number of parts, the size and structure of the *And/Or* graph, and the distribution of values for the durations and resources associated to the tasks.

As it is known, one of the most important problems in applying A* algorithms is the amount of memory wasted. In order to limit this consumption, the algorithm was adapted so that it used a depth-first search periodically for finding a new solution whose value could be

Table 1. Comparative results of the heuristics.

Heuristic	Nodes visited			Time (ms)			N-Df	N-F	% Error
	Ave	Max	Min	Ave	Max	Min			
h_1	30297	93376	32	13224	30930	0	11	15	0,985
h_2	4797	42775	32	668	6420	0	9	0	0,000
h_3	26414	88102	32	12408	30810	0	11	14	1,114
h_4	27046	87456	32	13321	30920	0	11	15	1,279
h_{51}	28745	92326	32	12892	31420	0	11	14	0,909
h_{53}	25641	81332	32	12598	30820	0	16	15	1,233
h_{54}	21698	54860	32	11462	30420	0	17	13	1,279
$\max(h_1, h_2)$	6008	71585	32	1453	30050	0	9	1	0,062
$\max(h_3, h_2)$	3267	23167	32	535	4230	0	8	0	0,000
$\max(h_4, h_2)$	4450	63996	32	1266	30050	0	6	1	0,041
$\max(h_{51}, h_2)$	5925	70432	32	1408	30150	0	8	1	0,062
$\max(h_{53}, h_2)$	2724	18254	32	465	4280	0	9	0	0,000
$\max(h_{54}, h_2)$	2459	18165	32	418	4280	0	8	0	0,000

used for pruning the search tree. Another improvement was done about detecting symmetries, so that redundant nodes are avoided in the expansion.

The results shown in Table 1, rather significant about the behaviour of the heuristics that have been defined, are based on a hypothetical product with 30 parts, with 396 Or nodes and 764 And nodes in the And/Or graph, so that the number of legal linear sequences is about 10^{21}. Each row of the table shows the results of 40 different problems solved using the corresponding heuristic, considering 10 different combinations on durations and resources among tasks, for each of four conditions in the resources used: 2 and 4 machines and 2 and 4 configurations/machine.

Table 1 shows, apart from the number of nodes and the time spent by the algorithm, how many times the optimal solution was found by a depth-first movement (N-Df), 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. The results show the successive improvements that the heuristics have obtained. When analysing the results, it must be taken into account that the optimal solution can have been found through a depth-first movement, that is carried out from the best node just when this strategy is used, that explains the apparent contradiction in the comparative results between different heuristics, since the nodes used for the depth-first movements are (surely) different in each case. Another improvement came from the combined use of the heuristic h_2, of additive nature, with the others, which consider the most unfavourable candidate task for the calculation of the corresponding estimation for each expansion node.

7. Conclusions

Different heuristics have been defined for selecting optimal assembly sequences. The first two ones are based on both relaxed models of the problem, each one considering only a type of constraints, one related to the precedence constraints, and the other one to the use of shared resources. From them, other heuristics have been defined, combining both types of information, obtaining successive improvements, as it is shown in the results obtained.

Acknowledgements

This work has been partially funded by the Spanish Ministerio de Ciencia y Tecnología under grant DPI2003-07146-C02-01, and the European Regional Development Fund (ERDF/FEDER).

References

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

[2] T.L. De Fazio and D.E. Whitney. Simplified Generation of All Mechanical Assembly Sequences. *IEEE J. Robotics and Automation*, Vol. 3, No. 6, pp. 640-658, 1987. Also, Corrections, Vol. 4, No. 6, pp. 705-708, 1988.

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

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

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

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

[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, pp. 188-199, 1990.

[8] J. Wolter. A Combinatorial Analysis of 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, pp. 915-921, 1996.

[12] C. Del Valle, M. Toro, E.F. Camacho and R.M. Gasca. A Scheduling Approach to Assembly Sequence Planning. *Proceedings of the 2003 IEEE Intl. Symp. on Assembly and Task Planning*, pp. 103-108, Besançon, France, 2003.

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