Università di Pisa

Dipartimento di Informatica

Technical Report : TR-98-10

Minimum Makespan Assembly Plans

Giorgio Gallo, Maria Grazia Scutella'

September 22, 1998

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

Minimum Makespan Assembly Plans

Giorgio Gallo and Maria Grazia Scutellà
Dipartimento di Informatica, Università di Pisa,
40, Corso Italia, 56125 Pisa, Italy
gallo@di.unipi.it, scut@di.unipi.it

Abstract

An important class of problems in manufacturing system are those in which there is an item to be produced, a set of components which can be assembled together in order to obtain the wanted item, and a set of machines which are capable to perform the assembly operations. Here we consider the case in which one wants to find an *optimal* list of assembly operations. After discussing several optimality criteria, we address the problem in which the assembly operations are to be chosen in such a way that they can be scheduled on the available machines at minimum *makespan*.

We show that these problems can be quite naturally modelled making use of Directed Hypergraphs, and describe a general heuristics, showing that in a particular but still relevant case, it yields solutions whose relative error is bounded. We present also a special class of assembly problems which can be solved in polynomial time by means of hyperpath computations.

Subject classifications:
Production/scheduling: planning and scheduling of assembly operations.
Analysis of algorithms: optimal hypergraph algorithms.

In robotic assembly systems, one or more types of items are produced by assembling together several parts given in input according to a properly defined assembly plan, where an *assembly plan* is a sequence of feasible assembly operations which allow one to derive the desired product from the given parts.

Systems of this type are required to have a high degree of flexibility in order to allow for quick switches from one type of product to another, or to recover easily from errors or unexpected events which may require changes in the assembly plan. Crucial to this point is the capability of generating and updating assembly plans in an efficient way.

Several alternative ways to represent relations among different feasible assembly operations and several approaches to find in the space of the feasible assembly operations an optimal sequence yielding the desired product have been proposed in the literature and used in practice (De Fazio and Whitney, 1987; Baldwin, Abell, Lui, De Fazio and Whitney, 1991).

Recently, the use of hypergraph models has been proposed to represent and derive assembly plans (Homem De Mello and Sanderson, 1990; Gallo and Pallottino, 1992). In particular, in [Gallo and Pallottino, 1992] the problem of finding optimal assembly plans with unbounded parallelism has been formulated and solved in terms of Shortest Hyperpath computations. Moreover, a heuristics has been described for the problem of assigning the operations of a given assembly plan to the machines in the case of bounded degree of parallelism.

In this paper, we shall investigate the problem of finding an optimal assembly plan under a bounded degree of parallelism, i.e. when k parallel identical machines are available for the execution of the operations. In particular, the Minimum Makespan Assembly Plan Problem will be addressed. Firstly, we shall extend the heuristics described in [Gallo and Pallottino, 1992], analyzing its worst case relative error. Then, we shall describe a particular case for which a minimum makespan assembly plan can be determined in polynomial time by means of hypergraph computations.

1 Assembly Plans

Let Q be a set of distinct parts. An *assembly* is an item obtained by properly connecting some of the parts in Q. Its cardinality is the number of parts it contains (a part is considered as an assembly of cardinality 1).

For the sake of the notational simplicity, we shall consider only binary operations, i.e. the assumption will be made that an assembly having a cardinality greater than one is obtained by connecting two *subassemblies*. The generalization to the case in which connections involving three or more subassemblies are allowed is straightforward. Similarly, it is immediate the generalization to the case in which unary operations are admitted, which may be useful to represent situations in which a given subassembly is modified or completed in some way, so creating a different subassembly. No assumption will be made on how the parts are assembled together (screwed, soldered,...). We shall just assume that, for any two assemblies which can be connected in order to yield a larger assembly, an assembly operation is defined. More precisely, given two assemblies, say π_1 and π_2, if π is the assembly resulting from their connection, we shall denote by $\omega = (\{\pi_1, \pi_2\}, \pi)$ the corresponding assembly operation, which will be called a *feasible assembly operation*.

Let Π be the set of all the assemblies which can be obtained from the parts in Q by means of sequences of feasible assembly operations, and Ω be the corresponding set of feasible assembly operations. Given a distinguished assembly $\pi^* \in \Pi$, by *assembly plan* relative to π^* we mean a minimal set of feasible assembly operations which, starting from the parts in Q, leads to the production of π^*, where by minimal we mean that none of its proper subsets is an assembly plan. More formally:

Definition 1.1 $(\bar{\Pi}, \bar{\Omega}, \pi^*)$ is a *feasible assembly plan* relative to π^* if and only if:

1. $\bar{\Pi} \subseteq \Pi; \bar{\Omega} \subseteq \Omega; \pi^* \in \bar{\Pi}$;
2. $(\{\pi_1, \pi_2\}, \pi) \in \bar{\Omega} \rightarrow \pi_1, \pi_2, \pi \in \bar{\Pi}$;
3. $\pi \in (\bar{\Pi} \setminus Q) \rightarrow \exists (\{\pi_1, \pi_2\}, \pi) \in \bar{\Omega}$;
4. no feasible assembly plan $(\tilde{\Pi}, \tilde{\Omega}, \pi^*) \neq (\bar{\Pi}, \bar{\Omega}, \pi^*)$ exists with $\tilde{\Pi} \subseteq \bar{\Pi}$ and $\tilde{\Omega} \subseteq \bar{\Omega}$.

Observe that we have not assumed $Q \subseteq \bar{\Pi}$ for any feasible assembly plan. In fact, it is possible that several assembly plans, making use of different sets of input parts, lead to the same item π^*. In such cases, a feasible assembly plan not only defines a set of feasible assembly operations needed to produce π^*, but also provides a design for π^*.

Assembly plans can quite naturally be formulated in terms of hypergraph concepts, as shown in the next sections.

1.1 Hypergraphs and related concepts

A directed *hypergraph* is a pair $H = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and E is the set of the hyperarcs, of cardinality m. A hyperarc is a pair, $e = (T(e), h_e)$, with $T(e) \subseteq V$ and $h_e \in V \setminus T(e)$; $T(e)$ is the *tail* of e, while h_e is its *head.*

While the size of a standard graph is uniquely defined by n and m, the size of a hypergraph depends also on the cardinality of its hyperarcs; so, the *size* of H is defined as the sum of the cardinalities of its hyperarcs:

$size(H) = \sum_{e \in E} |T(e)|$.

A *path* P_{st} from s to t in H is a sequence $P_{st} = (v_1 = s, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = t)$, where $s \in T(e_1)$, $h_{eq} = t$ and $h_{ei} \in T(e_{i+1})$, $i = 1, 2, \ldots, q-1$.

If $s = t$, then P_{st} is a *directed cycle.* When no directed cycle exists, then H is said to be cycle free or *acyclic.* In the acyclic case, it is possible to re-number the nodes in such a way that for any hyperarc $e = (T(e), h_e)$ and for any $v \in T(e)$, it is $v < h_e$ (this order is often called a topological order).

A *hyperpath* P_{Rt} from the root set R $(\subseteq V)$ to the sink t $(\in V)$ in H is a minimal acyclic sub-hypergraph of H containing both the nodes in R and node t, such that each node, with the exception of the nodes in R, has exactly one entering hyperarc. This definition of hyperpath can be shown to be equivalent to the following:

Definition 1.2 $P_{Rt} = (\bar{V}, \bar{E})$ is a *hyperpath* from the root set R to the sink t if and only if:

1. $\bar{V} \subseteq V; \bar{E} \subseteq E; R \subseteq \bar{V}; t \in \bar{V}$;
2. $e = (T(e), h_e) \in \bar{E} \rightarrow T(e) \subseteq \bar{V}, h_e \in \bar{V}$;
3. $v \in (\bar{V} \setminus R) \rightarrow \exists e = (T(e), h_e) \in \bar{E}$ with $v = h_e$;
4. no hyperpath from R to t, $(\tilde{V}, \tilde{E}) \neq (\bar{V}, \bar{E})$, exists with $\tilde{V} \subseteq \bar{V}$ and $\tilde{E} \subseteq \bar{E}$.

Given a hyperpath P_{Rt}, we call *characteristic flow* of P_{Rt} the hyperarc function ν recursively defined as follows: $\nu(e) = 1$ if e is the (unique) hyperarc of P_{Rt} entering t, while $\nu(e')$ is the sum of the characteristic flows on the hyperarcs outgoing from node v, if e' is the (unique) hyperarc of P_{Rt}

entering a node v other than t and not belonging to R. The characteristic flow can be computed in $O(size(P_{Rt}))$ time by performing a top down visit of the hyperpath, i.e. a visit in a reverse topological order from t towards the nodes in R (the characteristic flow is a particular hyperflow on H, which is a generalization of standard flows; for more details we refer to [Cambini, Gallo and Scutellà, 1997]).

A hyperpath from $R = \{a, b, c\}$ to $t = g$, with its characteristic flow, is provided in figure 1.

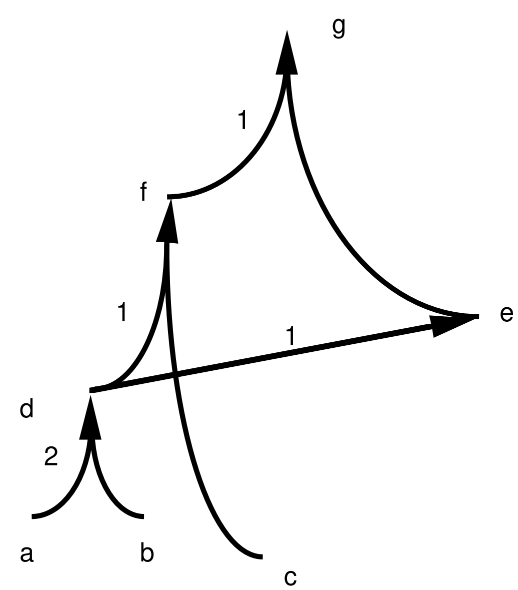

Figure 1: A hyperpath

Now, assume that each hyperarc e is assigned a real cost $c(e)$. Given a hyperpath P_{Rt} from R to t, by *cost function* we mean a node function C which assigns costs to the nodes of P_{Rt} depending on the costs of its hyperarcs. $C(t)$ is the cost of P_{Rt} under the chosen cost function.

Additive cost functions are those for which the cost of a node v not belonging to R can be written as the cost of the unique hyperarc e entering v plus a function $F(T(e))$ of the costs of its tail nodes: $C(v) = c(e) + F(T(e))$, $v \in \Pi_{Rt}, v \notin R$.

Two particular additive cost functions, which will be used in the following sections, are the distance and the value.

The *distance*, $Dist(v)$, is defined by the following recursive equations:

$$\begin{cases} Dist(v) = 0, & \text{if } v \in R; \\ Dist(v) = c(e) + \max\{Dist(w) : w \in T(e)\}, & \text{otherwise,} \end{cases} \tag{1}$$

where $c(e)$ can be interpreted as the length or duration of e.

The *value*, $Val(v)$, is the solution of the following recursive equations:

$$\begin{cases} Val(v) = 0, & \text{if } v \in R; \\ Val(v) = c(e) + \sum_{w \in T(e)} a(w,e)Val(w), & \text{otherwise,} \end{cases} \tag{2}$$

where, for each e and for each $w \in T(e)$, $a(w, e)$ is a nonnegative real coefficient.

The problem of finding minimum cost hyperpaths from a given root set R to all the nodes t for which a hyperpath from R to t exists is a natural generalization of the shortest path tree problem for standard graphs. This problem is equivalent to the problem of finding a solution to the following

Generalized Bellman's Equations, where by $BS(v)=\{e \in E : v = h_e\}$ we denote the *Backward Star* of v:

$$\begin{cases} C(v) = 0, & \text{if } v \in R; \\ C(v) = \min\{c(e) + F(T(e)) : e \in BS(v)\}, & \text{otherwise.} \end{cases} \tag{3}$$

Efficient algorithms for the solution of the Generalized Bellman's Equations in arbitrary hypergraphs have been proposed by Gallo, Longo, Nguyen and Pallottino (1993). Here, we report the simple case in which the hypergraph is acyclic, which is the case considered in this paper. In this particular case, minimum cost hyperpaths rooted at the nodes in R can be computed by the following algorithm, which visits the nodes according to a topological order.

```
Procedure HypMinCost(H, R, F, C)
begin
  C(v) = 0 if v ∈ R;
  C(v) = ∞ otherwise;
  for each v ∈ V according to a topological order do
    for each e ∈ BS(v) do
      begin
        f = F(T(e));
        if C(v) > c(e) + f then begin C(v) = c(e) + f
      end
    end
end
```

If the function F is the max function and the arc costs are the durations, then minimum distance hyperpaths are obtained; otherwise, if the weighted sum for F is chosen, then minimum value hyperpaths are computed. The corresponding minimum cost labels are returned by the algorithm via vector C.

Note that, in both cases, the evaluation of $F(T(e))$ runs in $O(|T(e)|)$ time; then, procedure $HypMinCost$ runs in $O(size(H))$ time.

1.2 Hypergraph formulation of assembly plans

Given a set Π of assemblies and the corresponding set Ω of feasible assembly operations, let us associate one node with each assembly in Π. Then, a feasible assembly operation, $\omega = (\{\pi_1, \pi_2\}, \pi)$, can be naturally represented as a hyperarc having $\{\pi_1, \pi_2\}$ as its tail and π as its head.

Consider the *assembly hypergraph* $H = (V, E)$, where $V = \Pi$ and $E = \Omega$ (if in the operation $(\{\pi_1, \pi_2\}, \pi)$ it is $\pi_1 = \pi_2$, in order to represent the operation as a hyperarc we can consider the assemblies π_1 and π_2 as distinct assemblies, so duplicating the corresponding node).

It is straightforward to see that, by construction, H is acyclic. Moreover, the following correspondence between feasible assembly plans and hyperpaths of H can be easily derived from Definitions 1.1 and 1.2:

Theorem 1.1 There is a one-to-one correspondence between the set of the feasible assembly plans relative to π^* and the set of the hyperpaths of H from Q to π^*.

Observe that each hyperarc $(\{\pi_1, \pi_2\}, \pi)$, belonging to a hyperpath of H from Q to π^*, corresponds to the activity of producing item π, and its corresponding characteristic flow gives the number of times this operation is performed, and hence the number of copies of π produced in assembling item π^* (for example, by interpreting the hyperpath in figure 1 as an assembly plan, two copies of item d

are produced). In particular, the sum of the characteristic flows on the hyperarcs outgoing from q, for $q \in Q$, gives the number of copies of part q needed in the production of π^*.

Observe also that the definition of hyperpath assures that all the assemblies produced are used in subsequent operations, with the exception of the last one, π^*, and that no operation can be performed unless all the assemblies it needs as input have been produced.

Due to this hypergraph interpretation, with a little abuse of notation we shall speak indifferently of assemblies and of nodes of H, and of feasible assembly operations and of hyperarcs of H.

To illustrate the use of hypergraph concepts to model assembly problems, let us take the simple example of a flashlight (figure 2) described by Homem de Mello and Lee (1991). Figure 3 shows the corresponding *assembly liaison graph*, which describes the set of the liaisons among the eight parts composing the flashlight. This graph, although useful to specify the relationships among the input parts, does not allow to obtain a complete information about the set of the feasible assembly operations leading to the production of the flashlight. In fact, some liaisons represent physical liaisons between parts and not feasible ways to link the parts (e.g. liaison L, dashed in figure 3); moreover, while each feasible subassembly does correspond to a connected subgraph of the assembly liaison graph, the opposite is not true, i.e. some connected subgraphs do not represent feasible subassemblies. In other words, liaison graphs represent only a partial instrument to describe the set of the feasible assembly operations. In order to get a complete specification, more sophisticated models are required, like the ones provided by the use of hypergraphs. This point has been clearly illustrated by Homen de Mello and Sanderson (1990) (there is a one-to-one correspondence between their And/Or graphs and the directed hypergraphs we are using here).

In figure 4 a portion of the assembly hypergraph for the flashlight example, which represents one feasible assembly plan, is provided. In the figure, each assembly is represented as the sequence of the parts it contains (the integer labels on the hyperarcs give the operations' costs).

Observe that the generation of the assembly hypergraph requires finding all the assemblies and all the feasible assembly operations associated with a given product. This generally involves the computation of suitable logical predicates, specifying the feasibility of assemblies and of feasible assembly operations (Homem de Mello and Lee, 1991). The hypergraph generation issue will be not addressed in this work. Hereafter, the assumption will be made that the assembly hypergraph is given as an input data. We just observe that all the algorithms presented can be easily adjusted to include the hypergraph generation; in this case the hypergraph is represented by: (i) the set of the initial components, (ii) the specification of the final product, and (iii) a rule which for any pair of subassemblies tells whether they can be connected together to form a new feasible subassembly, and provides the cost of the relative assembly operation.

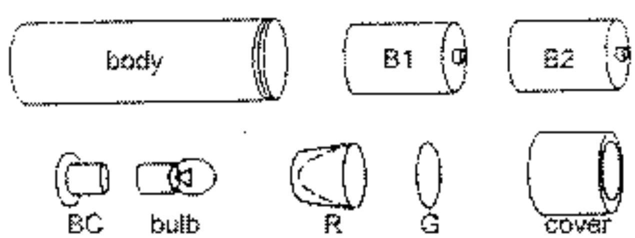

Figure 2: The flashlight assembly

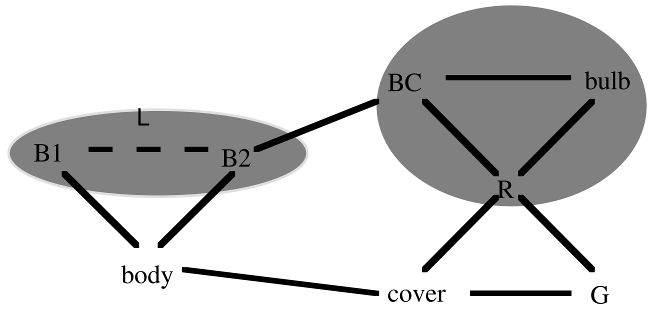

Figure 3: The flashlight liaison graph

1.3 Optimal Assembly Plans

An *optimal assembly plan problem* $(Q, \Pi, \Omega, \pi^*, \phi)$ is the problem of finding an assembly plan relative to π^* which satisfies a given criterion ϕ, usually written as a function to be minimized.

Several criteria (objective functions) can be considered.

As a first case, assume that each feasible assembly operation is assigned a real cost, and that the *total production cost* of an assembly plan is the sum of the costs of all the operations performed (if an operation is performed $p > 1$ times, then its contribution to the total cost is p times its cost). From the hypergraph interpretation just introduced, it follows easily that an assembly plan which minimizes the production cost can be computed by finding a minimum value hyperpath in H from the node set Q to π^*, when the coefficients $a(w, e)$ are all set equal to 1 (equations (2)). The problem can thus be solved in linear time by procedure *HypMinCost*, according to the results reviewed in section 1.1.

As a second interesting case, assume that each feasible assembly operation ω has an associated real value, $p(\omega)$, representing the probability that the execution of ω produces a defective item. Then, if we denote by $P(\pi)$ the probability that there is at least one fault in the assembly π due to the feasible assembly operation which has led to the production of π, it is easy to show that the operation $\omega = (\{\pi_1, \pi_2\}, \pi)$ may introduce a fault in π with probability:

$P(\pi) = P(\pi_1) + P(\pi_2) - P(\pi_1)P(\pi_2) + p(\omega)(1 - P(\pi_1))(1 - P(\pi_2))$.

By interpreting $p(\omega)$ as a cost associated with ω, an assembly plan which minimizes the fault probability can be computed by finding a minimum cost hyperpath in H, from the nodes in Q to node π^*, with respect to the following additive cost function:

$$F(\pi_1, \pi_2) = [P(\pi_1) + P(\pi_2) - P(\pi_1)P(\pi_2)](1 - p(\omega)).$$

This problem, too, can thus be solved in linear time by procedure *HypMinCost*.

A more interesting (and difficult indeed) case is the minimum makespan problem, which will be addressed next. Assume that each feasible assembly operation $(\{\pi_1, \pi_2\}, \pi) \in \Omega$ is assigned an integer duration, $d(\{\pi_1, \pi_2\}, \pi)$, and that k identical machines (robots) are available to perform the operations. The *Minimum Makespan Assembly Plan Problem* (MMAP) consists in finding an assembly plan relative to π^* which minimizes the time (*makespan*) needed to produce one unit of π^* on the k available machines.

Two particular easy cases are the one in which $k = 1$ (i.e. all the operations are performed in sequence) and the one in which $k = \infty$ (i.e. any number of parallel operations can be performed). Consider first the case $k = 1$.

For any assembly plan yielding the item π, its *total duration* is the sum of the durations of all

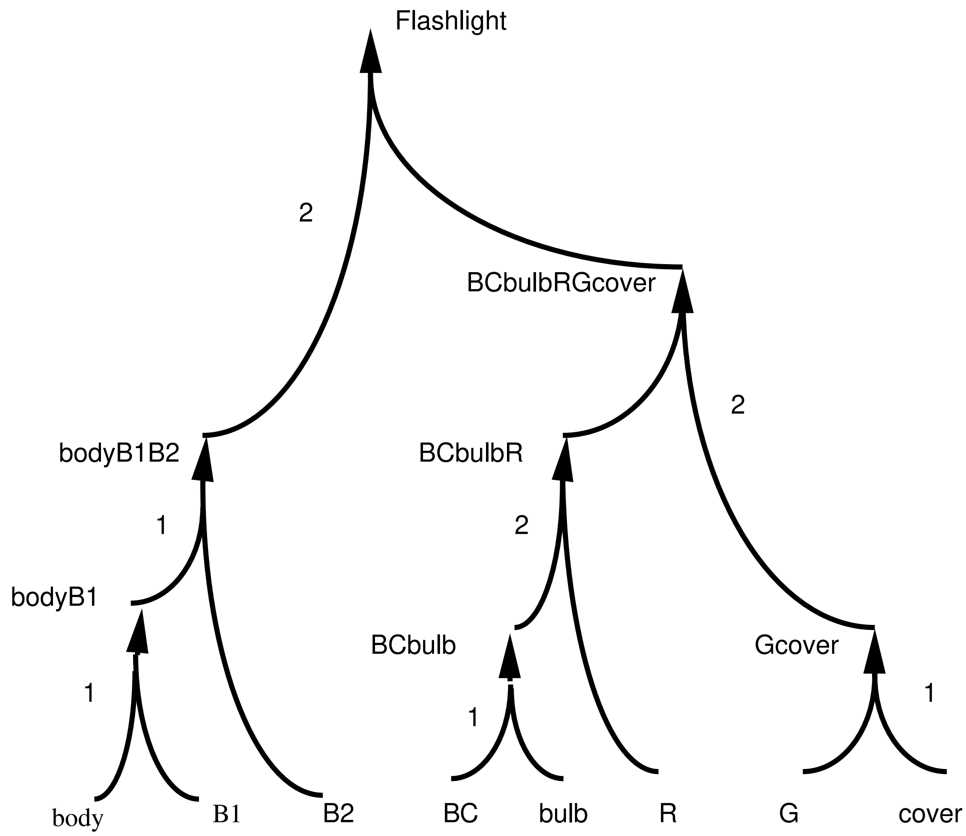

Figure 4: A portion of the flashlight assembly hypergraph

the operations (with the required multiplicities), i.e. it is the assembly plan makespan in the case of $k = 1$.

Being a special case of the minimum production cost problem introduced above, an assembly plan which minimizes the total duration can be computed in linear time by finding a minimum value hyperpath in H from the node set Q to π^*. Observe that, once an optimal assembly plan has been found, the corresponding assembly operations can be scheduled in any order on the machine, provided that the precedence constraints imposed by the plan are satisfied.

Consider now the other extreme case, that is $k = \infty$. As shown in [Gallo and Pallottino, 1992], computing a minimum makespan assembly plan with unbounded parallelism reduces to the problem of finding a minimum distance hyperpath in H from the nodes in Q to π^* (equations (1)). Also this problem can thus be solved in linear time by procedure *HypMinCost*. Observe that, in this case, scheduling the operations in the optimal assembly plan can be obtained by the following simple rule: assign to the first available machine the first available operation, where the operations are ordered by going from the lower levels of the optimal hyperpath towards π^*.

As an example, consider the assembly plan in figure 4, where the integers are the operation durations. There, the total duration associated with the hyperpath, i.e. the makespan in the case $k = 1$, is 10, whereas its distance, i.e. the makespan in the case of unbounded parallelism, is 7.

Let us go now to the general case ($1 < k < \infty$). In this case, the problem involves the solution of two linked subproblems: at the same time we want to find an assembly plan which minimizes the makespan, and we want to schedule the corresponding operations on the k available machines in order to compute the minimum makespan.

MMAP is NP-hard. In fact it has as a particular case the problem of scheduling n jobs on k identical machines, by satisfying a tree precedence relation, with the minimum makespan objective function. This is the case of instances with only one feasible assembly plan. In this case (i.e. one single assembly plan), a heuristics has been described in [Gallo and Pallottino, 1992], which approximates the optimum makespan with a relative error bounded from above by $\frac{k-1}{k}$. In the literature, a similar heuristics is credited to Graham (1966) (see, among the others, [Lawler et ali., 1993]).

In the next section, we shall extend the heuristics described in [Gallo and Pallottino, 1992] to the case of many feasible assembly plans. It will be proved that, when all the feasible assembly plans have the same total duration (Equal Duration Minimum Makespan Assembly Plan, EDMMAP), then the bound $\frac{k-1}{k}$ still holds. It will be shown that MMAP with this additional constraint includes several relevant assembly plan problems as its special cases, which can thus be approximated within $\frac{k-1}{k}$.

In section 3, we shall address a special but realistic case, which can be solved in polynomial time by means of hypergraph computations (Partitioned Minimum Makespan Assembly Plan, PMMAP). The proposed algorithm can also be used as a heuristics for the general MMAP, even if no worst case ratio is provided in such a case.

2 A list scheduling heuristics for MMAP

Here we present a heuristic two-phase algorithm to solve MMAP.

In the first phase, the bound on the number of machines is relaxed, and the minimum makespan assembly plan is found for the relaxed problem (i.e. $k = \infty$). That is done in linear time by computing a minimum distance hyperpath on the hypergraph which models the problem. Let $P^* = (V_{P^*}, E_{P^*})$ be the computed minimum distance hyperpath (or, equivalently, the computed minimum makespan assembly plan with unbounded parallelism).

In the second phase, the feasible assembly operations in P^* are assigned to the k available machines according to the following list scheduling heuristic algorithm.

2.1 The list scheduling algorithm

Let $P = (V_P, E_P)$ be an assembly plan from Q to π^*, and ν be its characteristic flow. The algorithm makes use of the following data structures:

a set of *candidate operations*, *Cand*, which contains the operations currently ready to be performed and not yet assigned to any machine;

for each $\pi \in V_P$, the number of copies of π, say $I(\pi)$, currently available and not assigned to any candidate operation, and the number of copies of π, say $J(\pi)$, which are needed by the assembly plan, but are neither the output of operations already assigned to the machines nor the output of candidate operations.

A subroutine $Init(Cand, I, J)$ initializes the set $Cand$ and the vectors I and J (in the procedure, by e_π we shall denote the unique hyperarc of P entering node π, for each π not belonging to Q). More formally:

Procedure Init$(Cand, I, J)$
begin
  $Cand := \emptyset$;
  for each $\pi \in V_P$ **do**
    if $\pi \in Q$ **then**
      begin $I(\pi) = \sum_{e \in E_P : \pi \in T(e)} \nu(e)$; $J(\pi) = 0$ **end**
      else begin $J(\pi) = \nu(e_\pi)$; $I(\pi) = 0$ **end**;
  for each $(\{\pi_1, \pi_2\}, \pi) \in E_P$ **such that** $I(\pi_1) > 0$ **and** $I(\pi_2) > 0$ **and** $J(\pi) > 0$ **do**
    begin
      $Cand = Cand \cup \{(\{\pi_1, \pi_2\}, \pi)\}$; $I(\pi_1) := I(\pi_1) - 1$; $I(\pi_2) := I(\pi_2) - 1$; $J(\pi) := J(\pi) - 1$
    end
end

For $j = 1, 2, \ldots, k$, let L_j be the list of the operations currently assigned to the machine j, while $\pi(j)$ and $t(j)$ denote the assembly produced by the last operation assigned to j and its completion time, respectively. Initially, by the procedure $Assign(Cand, L_1, \ldots, L_k)$, k candidate operations are arbitrarily assigned to the machines and $Cand$ is updated accordingly. Clearly, when $|Cand| \leq k$, then the problem is trivial and $|Cand|$ machines are sufficient to optimally assign the operations in P.

At each iteration, the algorithm selects a machine j with the current minimum value of $t(j)$, i.e. a machine which first completes the currently assigned operation. Then, if a feasible assembly operation can be performed which needs $\pi(j)$, then this operation is assigned to machine j. Otherwise, any candidate operation from $Cand$, if any, is assigned to machine j.

At termination, L_j provides the schedule of machine j and $t_P = \max\{t(j) : j = 1, 2, \ldots, k\}$ is the returned completion time of the assembly plan P.

Procedure Bounded Assembly$(V_P, E_P, Q, \nu, k, L_1, \ldots, L_k, t_P)$
begin
  Init$(Cand, I, J)$; *Assign*$(Cand, L_1, \ldots, L_k)$; *Make(Pr)*;
  repeat
  j:= *Selectmin(Pr)*;
  if there exists $\omega = (\{\pi_1, \pi(j)\}, \pi) \in E_P$ **such that** $I(\pi_1) > 0$ **and** $J(\pi) > 0$
    then begin *Append*(L_j, ω); $I(\pi_1) := I(\pi_1) - 1$; $J(\pi) := J(\pi) - 1$ **end**
    else
    begin
      $I(\pi(j)) := I(\pi(j)) + 1$;
      if $Cand = \emptyset$ **then** *Remove*(Pr, j)
      else begin *Extract*$(Cand, \omega)$; *Append*(L_j, ω) **end**
    end
  Update(Pr)
  until $Pr = \emptyset$;
  $t_P := t(j)$
end

In the algorithm, a priority queue Pr contains the indices of the machines which are currently working, ordered according to non decreasing values of $t(j)$. Procedure *Make*(Pr) initializes Pr with the indices of the machines which have been assigned an operation by procedure *Assign*. Procedures *Selectmin*(Pr), *Remove*(Pr, j) and *Update*(Pr) perform typical operations on priority queues such as the selection of a minimum key element, the removal of one of its elements and the updating of its internal structure, if needed.

Extract$(Cand, \omega)$ extracts a candidate operation ω from *Cand*, while *Append*(L_j, ω) appends the operation ω to the list L_j, updating accordingly $\pi(j)$ and $t(j)$.

If we denote by $|P|$ the number of the operations of the assembly plan P, it is easy to see that, if Pr is implemented as a heap, then *Bounded Assembly* runs in $O(|P| \lg k)$. In fact, at each iteration the most expensive operation is the heap updating which is performed in $O(\lg k)$ time.

In the following subsections, firstly we shall show that, if only one feasible assembly plan is admitted, then the relative error of the solution returned by *Bounded Assembly* is bounded from above by $\frac{k-1}{k}$. Then, we shall extend this result to a larger sub-family of MMAP.

2.2 The relative error in the case of single assembly plan

Consider the special case in which the set of the feasible assembly plans has cardinality one. Let P be the only feasible assembly plan, and t_P be the completion time as computed by *Bounded Assembly*.

Let t_i denote the time in which the operation ω_i is started in the schedule provided by *Bounded Assembly*; assume, without loss of generality, that the operations in P are ordered in such a way that if $t_i < t_j$ then $i < j$, i.e. the operations are ordered according to a non decreasing value of their starting time in the solution provided by *Bounded Assembly*.

Denote by $depth(\omega_i)$ the sum of the durations in the longest sequence of operations starting from ω_i and such that the output of each operation but the last one is used as one of the inputs in the next operation; that corresponds to the longest path in P from any of the nodes in the tail of ω_i to π^*. Moreover, let $depth = \max\{depth(\omega_i) : i = 1, 2, \ldots, |P|\}$ be the longest path in P. If t_{opt} denotes the minimum makespan of P on the k available machines, then the following lower bound clearly holds:

Lemma 2.1 $t_{opt} \geq \max\{depth, \frac{\sum_i d(\omega_i)}{k}\}$.

Now, let us provide an upper bound on the solution t_P provided by *Bounded Assembly*.

Let ω_l be the last operation removed from *Cand* by *Bounded Assembly*. Note that, after the initialization, only removal operations are performed on *Cand* and hence, after the removal of ω_l, it remains empty. Observe that the time interval $[0, t_P]$ can be partitioned into two intervals $[0, t_l]$ and $[t_l, t_P]$ with the following property: in the first interval, the number of available operations (operations which are either currently been executed or belonging to the current set *Cand*) is greater than k; in the second interval the number of available operations is less than or equal to k.

Define $C_l = \{i \leq l : t_i + d(\omega_i) \leq t_l\}$. The set C_l is the set of the feasible assembly operations which have been completed in the interval $[0, t_l]$. The following theorem holds:

Theorem 2.1(Gallo and Pallottino, 1992) The relative error of the solution provided by *Bounded Assembly* is bounded from above by $\frac{k-1}{k}$.

Proof: Let us consider the schedule provided by *Bounded Assembly*. By construction, there is a machine, say j, which works without any interruption during the whole interval $[0, t_P]$. Such a machine is the one which performs the last operation.

Let ω_h be the first operation assigned to machine j, which does not belong to the set C_l, and t_h be its starting time. If C_h is the set of the operations not belonging to C_l which are assigned to machine j, then we have:

$\sum_{i \in C_h} d(\omega_i) = t_P - t_h$.

Furthermore:

$t_h \leq \frac{1}{k} \sum_{t_i < t_h} d(\omega_i)$
$\leq \frac{1}{k} \sum_{i \notin C_h} d(\omega_i)$.

The inequalities derive from the fact that $t_h \in [0, t_l]$ and so from time 0 to time t_h all the machines work continuously performing operations which start before time t_h and hence which do not belong to C_h. So we have:

$t_P = t_h + \sum_{i \in C_h} d(\omega_i)$
$\leq \frac{1}{k} \sum_{i \notin C_h} d(\omega_i) + \sum_{i \in C_h} d(\omega_i)$,

and so, by Lemma 2.1:

$\epsilon = \frac{t_P - t_{opt}}{t_{opt}} \leq \frac{\sum_{i \in C_h} d(\omega_i) + \frac{1}{k} \sum_{i \notin C_h} d(\omega_i) - \frac{1}{k} \sum_i d(\omega_i)}{depth} =$
$\frac{\sum_{i \in C_h} d(\omega_i) - \frac{1}{k} \sum_{i \in C_h} d(\omega_i)}{depth} =$
$\frac{k-1}{k} \frac{\sum_{i \in C_h} d(\omega_i)}{depth}$.

Since $t_h + d(\omega_h) > t_l$, we have that the only operation available after ω_h to be assigned to machine j is an operation which uses as input the output of ω_h; that remains true up to time t_P. Then C_h defines a path in the hyperpath P and hence it is:

$\sum_{i \in C_h} d(\omega_i) \leq depth.$

It follows that:

$\epsilon \leq \frac{k-1}{k}$,

and the proof is completed. $\diamond$

2.3 The Equal Duration Minimum Makespan Assembly Plan Problem

Consider the problem of computing a minimum makespan assembly plan in the case in which all the assembly plans relative to the required product π^* have the same total duration. Hereafter this special case of MMAP, which generalizes the one discussed in the previous subsection, will be referred to as the *Equal Duration Minimum Makespan Assembly Plan Problem*, or EDMMAP for short.

It is straightforward to show that, under the hypothesis of equal total duration, Lemma 2.1 still holds when applied to the minimum makespan assembly plan in the case of unbounded parallelism, which we called P^* in section 2. In fact, if t_{opt} denotes the minimum makespan associated with EDMMAP, then $t_{opt} \geq depth$ still holds since $depth$, being the longest path in P^*, is the minimum makespan in the case of unbounded parallelism. Moreover, $t_{opt} \geq \frac{\sum_i d(\omega_i)}{k}$ still holds true due to the assumption that all the feasible plans have the same total duration.

As a consequence, Theorem 2.1 is still true, that is the relative error of the solution returned by *Bounded Assembly* when applied to P^* is bounded from above by $\frac{k-1}{k}$:

Corollary 2.1 The relative error of the solution provided by the list scheduling heuristics when applied to EDMMAP is bounded from above by $\frac{k-1}{k}$.

An interesting particular case of EDMMAP is the one in which the problem data can be completely represented by means of an undirected weighted and connected graph $G = (Q, A)$. These problems, which we call *Graph structured assembly plan problems*, are characterized by the following properties:

- the set of the nodes is the set of the initial parts;
- an arc (i, j) exists in A if and only if part i and part j can be connected together to form a subassembly, and the weight $d(i, j)$ of the arc is the duration of the operation connecting i and j;

- there is a correspondence between connected subgraphs $(\tilde{Q}, \tilde{A})$ of G, with $\tilde{Q} \subseteq Q$ and $\tilde{A} = \{(i,j) : i, j \in \tilde{Q}\}$, and feasible subassemblies;
- for any two feasible subassemblies, π_1 and π_2, corresponding to the subgraphs (Q_1, A_1) and (Q_2, A_2) respectively, the operation of connecting them, if feasible, has as duration the sum of the weights of all the arcs (i,j) such that $i \in Q_1$ and $j \in Q_2$;
- the required item π^* corresponds to (Q, A).

Particular cases of Graph structured assembly plan problems are the path and the grid case, as shown in figure 5.

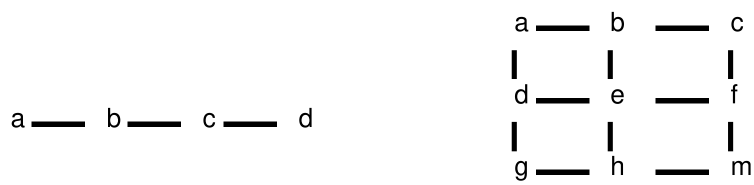

Figure 5: The path and the grid case

By definition, Graph structured assembly plan problems are such that all the feasible assembly plans relative to π^* have the same total duration, i.e. $\sum_{(i,j)\in A} d(i,j)$.

As a consequence, the bound $\frac{k-1}{k}$ on the relative error provided by the list scheduling heuristics holds true for this family of problems.

Graph structured assembly plan problems can be used as relaxations of more general MMAPs. Consider for instance the flashlight assembly plan problem illustrated in figure 2, and its corresponding assembly liaison graph in figure 3. As previously observed, some liaisons represent physical links between parts and not feasible ways to link the parts; in addition, precedence relationships among the feasible assembly operations are not explicitated. However, we can relax the constraints concerning precedence relationships and assume that all the operations represented by the liaison graph (with the exception of the liaisons representing only physical links) are feasible assembly operations. The problem thus reduces to a Graph structured assembly plan problem defined on the liaison graph.

As an example, if we would solve the Graph structured assembly plan problem defined on the liaison graph in figure 3 (by ignoring the dashed liaison L) in the case $k = \infty$ by assuming that each liaison has duration 1, then the optimum assembly plan depicted in figure 6 would be obtained, having makespan 6. On the other hand, if the assumption is made that the set of the feasible assemblies is {bodyB1, bodyB1B2, BCbulb, RG, BCbulbRG, BCbulbRGcover, flashlight}, in addition to the input parts, then an optimum assembly plan would be the one in figure 7, having makespan 7.

3 The Partitioned Minimum Makespan Assembly Plan Problem

Here we consider a special case of MMAP which will be called *Partitioned Minimum Makespan Assembly Plan Problem* (PMMAP). In this problem, the production of the item π^* is (at least from a logical point of view) split into two phases.

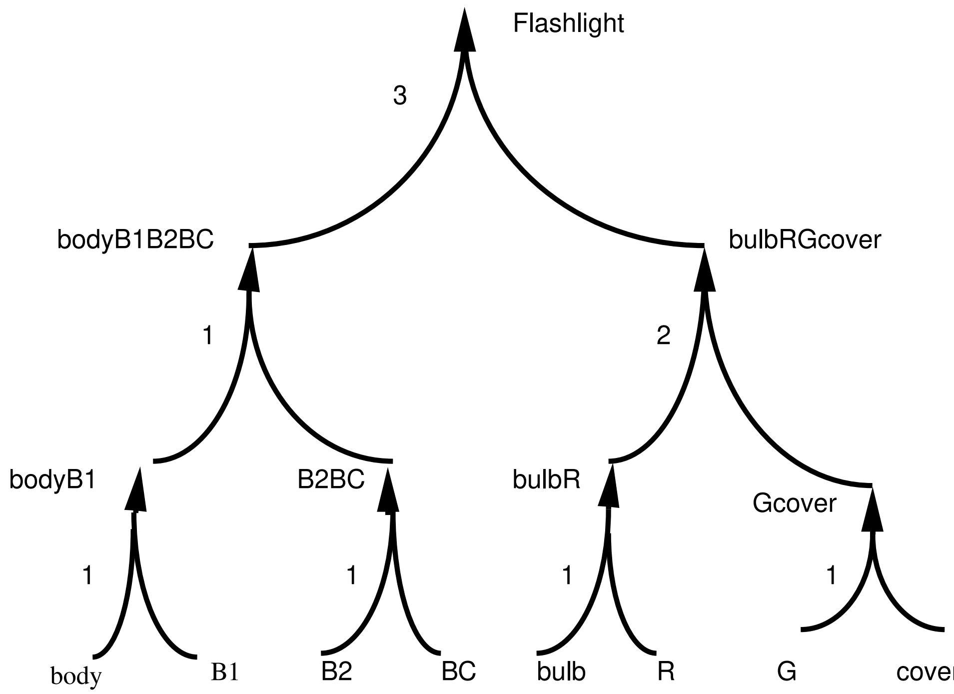

Figure 6: The relaxed assembly plan

In a first phase h ($h \leq k$) subassemblies π_{i1}, π_{i2}, ..., π_{ih}, $h \leq k$, are produced, each by one single dedicated machine (*partitioned scheduling constraint*).

In the final phase the h subassemblies π_{i1}, π_{i2}, ..., π_{ih} are connected together, by means of feasible assembly operations, until the item π^* is produced. In this second phase, being k machines available, we are in the case of unlimited parallelism and, as shown earlier, an optimal assembly plan can be easily determined.

Observe that the two phases might overlap, that is two subassemblies π_{iv} and π_{iw} can be connected by one of the idle machines, while another machine, say q, is still busy in the production of π_{iq}. Here, the difficult problem is to select the h subassemblies π_{i1}, π_{i2}, ..., π_{ih} in such a way that π^* is produced with minimum makespan. PMMAP includes the case in which π^* is a complex item involving a large number of components, and its production is split among a certain number of different units within the factory and/or some external contractors (outsourcing).

As an example, consider the Graph structured assembly plan problem represented in Figure 8, where the label on each arc is the duration of the operation needed to realize the corresponding liaison. By assuming $k = 4$, a possible solution satisfying the partitioned scheduling constraint is one which assigns the subassemblies A, B, C and D to the 4 dedicated machines. The assembly of the 4 subassemblies, together with a possible scheduling of the operations on the 4 machines, is illustrated in figure 9, where each arc is labelled with the duration of the corresponding operation and each node is labelled with the item produced, the machine used and the time at which the item is available.

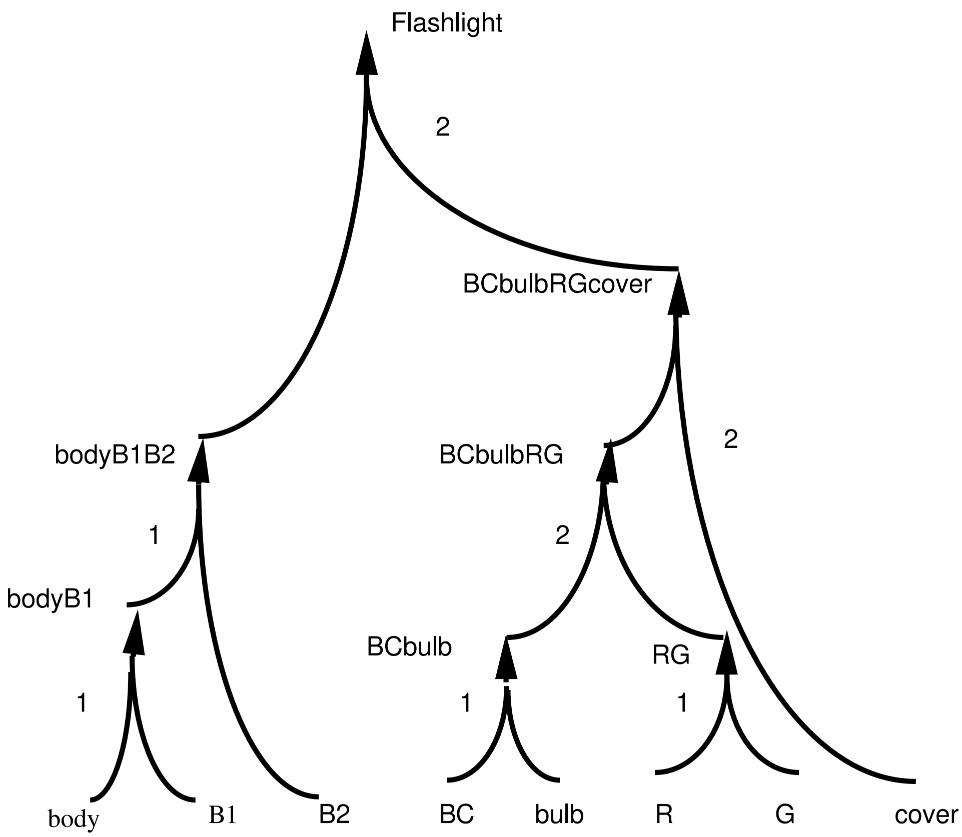

Figure 7: The optimum assembly plan

In the following we shall show that MMAP can be formulated and solved polynomially in terms of hyperpath computations.

Consider the assembly hypergraph $H = (V, E)$ introduced in 1.2.

Let $T^h(\pi)$, $h \in \{1, 2, \ldots, k\}$, be the minimum makespan associated with item π in PMMAP when π is obtained by assembling together exactly h subassemblies, each of them produced by a single dedicated machine. Moreover, $D(\pi)$ denote the minimum makespan needed to produce π in the case $k = 1$.

It is easy to see that the minimum makespan $T(\pi), \forall\pi \in \Pi$, is given by:

$$T(\pi) = \min\{T^1(\pi), T^2(\pi), \ldots, T^k(\pi)\}, \tag{4}$$

where $T^h(\pi), h \in \{1, 2, \ldots, k\}$, is the solution to the following recursive equations:

$$\begin{cases} T^1(\pi) = D(\pi), \\ T^h(\pi) = \min\{d(\{\pi_1, \pi_2\}, \pi) + \max\{T^p(\pi_1), T^q(\pi_2)\} : \\ (\{\pi_1, \pi_2\}, \pi) \in BS(\pi), p + q = h\}, h > 1. \end{cases} \tag{5}$$

Informally, for a given assembly π we have to decide whether to produce π entirely by means of a unique machine (and, in this case, $D(\pi)$ is the minimum makespan), or to obtain it by means of

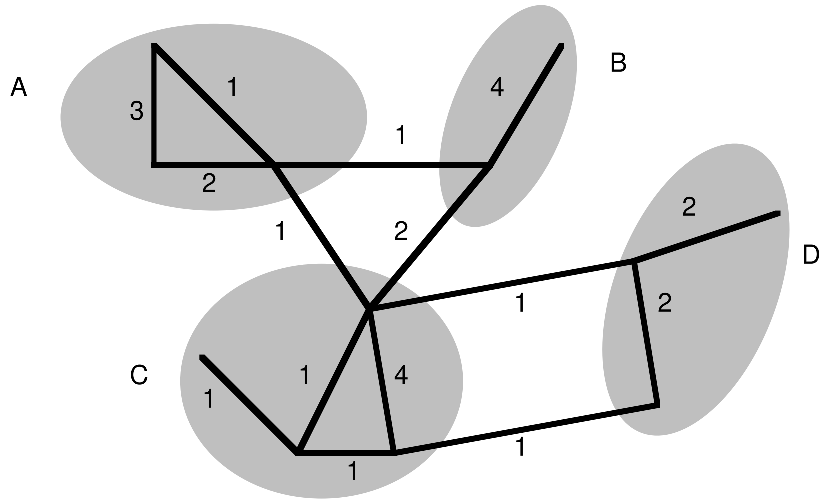

Figure 8: First phase: a partitioned scheduling

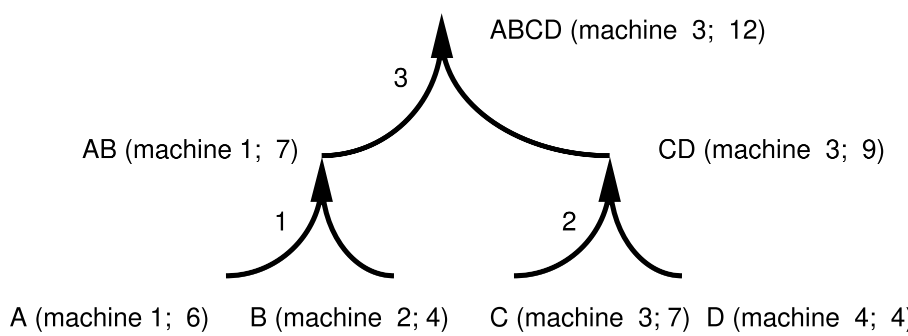

Figure 9: Second phase: assembling the subassemblies

a feasible assembly operation $(\{\pi_1, \pi_2\}, \pi)$ satisfying the partitioned scheduling constraint, i.e. by assembling together two subassemblies π_1 and π_2 such that the number of subassemblies in π_1 which are assigned to dedicated machines, i.e. p, and the number of subassemblies in π_2 which are assigned to dedicated machines, i.e. q, are such that $(p+q) \leq k$. What we need to solve MMAP is to compute $T(\pi^*)$.

Observe that, in practical applications, $D(\pi)$ is not necessarily the minimum makespan to produce item π on a single machine, but more generally it can represent the time required to produce π by units within the factory, which may have more than one machine; in this case, a nested scheduling problem has thus to be solved.

In order to compute $T(\pi^*)$, consider the hypergraph H with the addition of a dummy root node ρ, and of a dummy hyperarc (ρ, π) for each node π, representing the production of the assembly π by means of a unique dedicated machine. Let $D(\pi)$ be the duration associated with (ρ, π), for each node π.

Definition 3.1 A *h-constrained hyperpath* is a hyperpath having $R = \{\rho\}$ as the root set and such that the sum of the characteristic flows on its hyperarcs outgoing from ρ is h.

In figure 10, the 4-constrained hyperpath corresponding to the solution in figure 9 is represented (in fact, the sum of the characteristic flows on the hyperarcs outgoing from ρ is 4); the number associated with an hyperarc of type (ρ, π) is the (minimum) total duration associated with π.

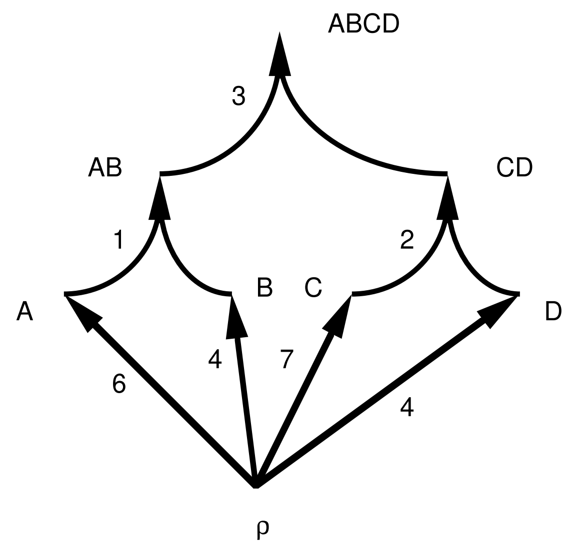

Figure 10: A 4-constrained hyperpath

Let P^k be the set of the h-constrained hyperpaths entering π^*, $h = 1, 2, \ldots, k$. By exploiting the above definition, it is easy to prove that PMMAP reduces to the problem of finding a minimum distance hyperpath in P^k.

Theorem 3.1 There is a one-to-one correspondence between the hyperpaths in P^k and the feasible assembly plans entering π^* when scheduled according to the partioned scheduling policy. Moreover, minimum distance hyperpaths in P^k provide the optimum solution to PMMAP.

The proof follows directly from the definition of h-constrained hyperpaths and from Theorem 1.1.

Now, define the k-expanded hypergraph $H^k = (V^k, E^k)$ with:

- $V^k = \{\pi^h : \pi \in V, h = 1, 2, \ldots, k\} \cup \{\rho, \tau\}$,
- $E^k = \{(\{\pi_1^p, \pi_2^q\}, \pi^h) : (\{\pi_1, \pi_2\}, \pi) \in E, p + q = h, p, q \geq 1, h = 2, 3, \ldots, k\}$
  $\cup \{(\rho, \pi^1) : \pi \in V\} \cup \{(\pi^{*h}, \tau) : h = 1, 2, \ldots, k\}$,

where τ is a dummy supersink.

Clearly any hyperpath from ρ to τ is a h-constrained hyperpath, $h = 1, 2, \ldots, k$, and hence corresponds to a feasible solution to PMMAP. Now, let us assign each hyperarc $e \in E^k$ a cost $c(e)$ defined as follows:

$$\begin{cases} c(e) = d((\{\pi_1, \pi_2\}, \pi)), & \text{if } e = (\{\pi_1^p, \pi_2^q\}, \pi^h), \text{for some } p, q, h; \\ c(e) = D(\pi), & \text{if } e = (\rho, \pi^1); \\ c(e) = 0, & \text{if } e = (\pi^{*h}, \tau), \text{for some } h \end{cases} \tag{6}$$

Then, an optimal solution to PMMAP can be obtained by solving a minimum distance hyperpath problem on H^k. Since $|V^k| = O(k|V|)$ and $|E^k| = O(k^2|E|)$, one has that $size(H^k) = O(k^2 size(H))$ and hence, being the meaningful values for k bounded by the cardinality of V, the k-expanded hypergraph H^k has a size which is polynomial in the size of the original hypergraph H.

Once the cost function $c : E^k \to R$ has been computed, the solution to PMMAP can thus be found in $O(k^2 size(H))$ time by a minimum distance hyperpath algorithm. To compute the cost function the only costly operation is the computation of $D(\pi)$ for all $\pi \in V$. If, as we have assumed, $D(\pi)$ is the minimum time needed to produce π on a single machine, one single run of a minimum value hyperpath algorithm on H yields $D(\pi)$, for all $\pi \in V$, with a $size(H)$ time complexity. We can conclude that:

Theorem 3.2 PMMAP can be solved in $O(k^2 size(H))$ time.

Observe that, during the hyperpath computations, it is possible to avoid the explicit generation of the entire hypergraph H^k. In fact, in a more efficient way, it is possible to generate and visit only the portion of H^k which is necessary in order to compute the minimum makespan assembly hyperpath entering the dummy destination node τ.

Moreover, in the case in which no assumptions are made on the way in which each unit performs the operations it has been given, what we can say is that PMMAP can be solved within a polynomial number of calls to the scheduling algorithm used by the single units.

4 Conclusions

Based on hypergraph models, we have presented and analyzed some special classes of the optimal assembly plan problem. We have shown that, depending on the objective function chosen, an optimal assembly plan problem can be *easy* (in fact solvable by means of rather simple optimal hyperpath computations) or *NP-hard.* A general computationally fast heuristic algorithm for finding a *minimum makespan assembly plan* has been proved to give solutions with bounded approximation when the operations' durations have a particular structure. Moreover, in the last part of the paper a new special class of problems has been presented, for which an elegant and fast polynomial time algorithm exists.

REFERENCES

BALDWIN, D. F., ABELL,T. E., LUI, M. C., DE FAZIO, T. L., AND D. E. WHITNEY. 1991. An integrated computer aid for generating and evaluating assembly sequences for mechanical products. *IEEE Trans. on Robotics and Automation* **7**, 78-94.

CAMBINI, R. , GALLO, G. , AND M. G. SCUTELLÀ. 1997. Flows on hypergraphs. *Mathematical Programming* **78**, 195-217.

DE FAZIO, T. L., AND D. E. WHITNEY. 1987. Simplified generation of all mechanical assembly sequences. *IEEE Journal of Trans. on Robotics and Automation* **RA-3**, 640-658.

GALLO, G. , LONGO, G. , NGUYEN, S. , AND S. PALLOTTINO. 1992. Directed hypergraphs and applications. *Discrete Applied Mathematics* **40**, 177-201.

GALLO, G. , AND S. PALLOTTINO. 1992. Hypergraph models and Algorithms for the Assembly Problem. TR-6/92, Dipartimento di Informatica, Università di Pisa.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computers and Intractability (A Guide to the Theory of NP-Completeness)*. Freeman and Company, San Francisco.

GRAHAM, R. L. 1966. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* **45**, 1563-1581.

LAWLER, E. L., LENSTRA, J. K., RINNOY KAN, A. H.G. AND D. B. SHMOYS. 1993. Sequencing and Scheduling: Algorithms and Complexity. *Handbooks in OR and MS.* **4**, Elsevier Science Publishers.

HOMEM DE MELLO, L. S., AND A. C. SANDERSON. 1990. And/Or Graph Representation of Assembly Plans. *IEEE Trans. on Robotics and Automation* **6**, 188-199.

HOMEM DE MELLO, L. S., AND S. LEE. 1991. *Computer-Aided Mechanical Assembly Planning.* Kluwer Academic Publishers.