Dedicated to
Professor Jakub Gutenbaum
on his 70th birthday

Control and Cybernetics

vol. **29** (2000) No. 1

Assignment and sequencing of parts to autonomous workstations

by

M. Lucertini[1], **F. Nicolò**[2] and **S. Smriglio**[3]

[1] Dipartimento di Informatica, Sistemi e Produzione,
Università di Roma "Tor Vergata", lucertini@disp.uniroma2.it
[2] Dipartimento di Informatica ed Automazione,
Università di Roma Tre, nicolo@uniroma3.it
[3] Dipartimento di Matematica Pura ed Applicata
Università di L'Aquila, smriglio@univaq.it

Abstract: We present an optimization-based coordination protocol among autonomous workstations in a multiprocessor stage devoted to painting of the shutters in a furniture production process. The coordination aims to maximize the number of parallel operations executable at each machine cycle, while fulfilling constraints on the unique-copy tools. The mechanism is derived by a distributed implementation of a bipartite matching algorithm. The resulting procedure is shown to be compatible with the several autonomous decisions characterizing the process.

Keywords: bipartite matching, polynomial-time algorithm, distributed algorithm, coordination mechanism.

1. Introduction

Production systems often suffer from problems caused by unplanned events and complexity. This motivates the investigation of control systems and architectures able to guarantee high degree of modularity, simplicity, flexibility, and distribution. The traditional models, based on scheduling and control theory, may suffer from lack of applicability, since they do not account satisfactorily for several aspects of real world environments (Lin and Solberg, 1992).

A recent approach providing a decomposed and modular framework is based on the paradigm of autonomous agents (AA), in which the plan is not established

by a global controller, but is the result of the decisions of several entities each pursuing its individual goal. Autonomous agents can be defined as entities of an organized system able to decide on the value of a set of control variables, with given objectives and partially defined constraints, on the ground of the available information. It is important to notice that, in practice, autonomous decisions represent a major part of the decision process, whereas they are not primarily considered in system design, especially at the shop floor level. Besides other advantages, the AA paradigm fully accounts for this aspect.

As observed by Lin and Solberg (1992) the AA concept generalizes and integrates other alternative control and scheduling architectures, such as cooperative systems, heterarchical structures, object oriented programming, real time negotiations of resource assignment and opportunistic scheduling, designed to meet the complexity as well as the distributed nature of the process.

The investigation of a complex system by the AA paradigm consists of

(*i*) analysis of the system *architecture*;
(*ii*) identification of the *agents*;
(*iii*) identification of a *coordination module*;
(*iv*) characterization of the agents and of the coordination module in terms of: memory, data elaboration capability and specific features;
(*v*) characterization of the *communication* and *negotiation* protocols (agent to agent, and agent to coordination module).

The coordination module can play different roles, ranging between two extreme situations. The first corresponds to a strong control action, i.e., the agents are not autonomous and just implement the decision of the global controller. The opposite situation arises when the agents are completely autonomous without coordination. The relationship among the aforementioned five basic features of a multi-agent system is the object of several research contributions.

Lin and Solberg (1992) proposed a general framework for part flow management based on a negotiation protocol among parts and other resources (both jobs and resources have their own objectives). The purpose of their experiments, performed with object oriented simulation, is to verify the flexibility of the framework with respect to frequent changes in the environment. A recent proposal is forwarded in Adacher et al. (1999) where different implementations of the AA concept in flexible manufacturing and several control architectures are investigated by an extensive simulation experience. Decker and Lesser (1994) designed a modular family of coordination mechanisms. These support the activity scheduling for teams of cooperative computational agents.

A different approach has been obtained by looking at distributed implementations of known algorithmic paradigms for optimization problems. For instance, in Graves (1982), Della Croce et al. (1993), Gou et al. (1994), Gou and Luh (1997), the authors observe that the natural functional decomposition of the system can be exploited in the study of Lagrangian relaxation of the planning problem. In this framework, a subsystem decides on the value of its variables on the ground of the current values of the Lagrangian multipliers

imposed by the coordination module.

A similar approach is presented in Arbib and Rossi (1999), in which a distributed primal-dual heuristic is proposed for the solution of a *generalized covering* problem arising in a manufacturing environment.

In this paper we investigate the problem of assigning and sequencing a batch of parts to a set of parallel autonomous workstations with some specific operational rules. The analysis is based on a formulation of the problem as a *bipartite matching* with GUB (*Generalized Upper Bound*) constraints. We exploit the combinatorial characterization of the problem to implement autocoordination mechanisms among agents which satisfy simplicity and flexibility requirements, while mantaining a nice behaviour in terms of solution quality. Moreover, the resulting model is sufficiently general to be applied in different production contexts.

The structure of the paper is the following. In Section 2 we describe the system under investigation. Section 3 is devoted to the autocoordination procedure: in Subsection 3.1 we introduce the mathematical background and investigate the unrestricted case; in Subsection 3.2 we discuss the implementation details and the extension of the approach to the general case. Finally, in Section 4 some conclusions are drawn.

2. System description

In this section we describe a working center for the spray painting of components in a flow line devoted the production of kitchen furniture. The system is characterized by the presence of several manual operations and autonomous decisions, and represents a suitable environment for the introduction of optimization-based coordination patterns among autonomous agents.

The working center consists of: (*i*) an input storage area; (*ii*) a processing (painting) stage (including a tool magazine); (*iii*) an output storage area in which the refined components wait to be transported to the assembly working center. The process layout is depicted in Fig. 1. The input storage area is fed by batch arrivals from the cutting center. A batch P contains n raw shutters to be painted and refined. Separately from the batches, some more isolated shutters can be dropped into the input magazine. These are *urgent* parts, which must be processed with priority so as to satisfy an unexpected need having arisen in the assembly center.

The painting operation is performed, within an airtight room, by a set C of m parallel (not identical) workstations. In practice, m ranges from 8 to 12 and n ranges from 200 to 1000, while the number of urgent shutters in the input buffers rarely reaches 50. Each part i is labelled with its identification number and must be processed by one among the workstations in a given subset $C_i \subseteq C$. The compatibility among shutters and workstations depends primarily on the shutter dimensions, which, on the other hand, do not affect significantly the duration of the operation. The latter is therefore assumed to be the same

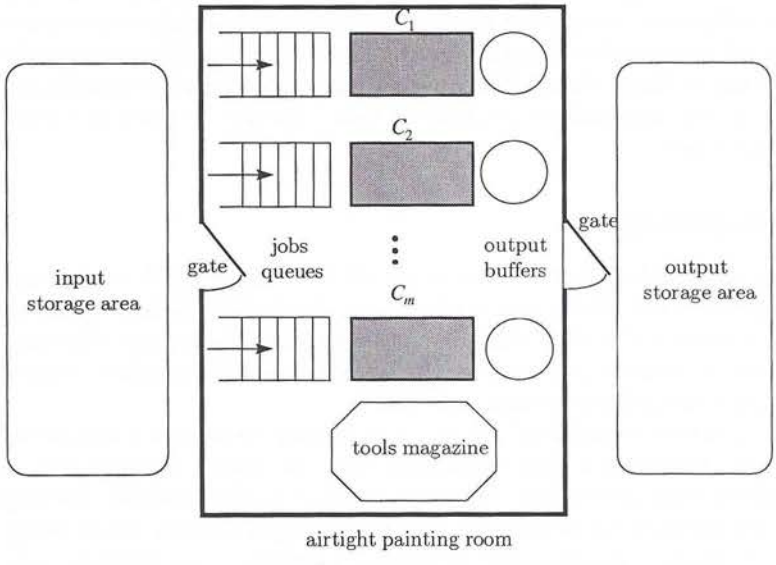

Figure 1. The working center layout

for all the operations. As a consequence, the parallel workstations operate in a synchronous fashion, so as to help the loading/unloading operation. In the latter, a batch of finished parts is brought to the output storage area.

The tool selection and placement is carried out by autonomous operators in charge of tool management. Their activity includes also tool breakage and monitoring. A peculiarity of the problem comes from the fact that some tools, namely, some specific sprinklers, are available in a unique copy. As a consequence, any two operations requiring the same unique copy tool cannot be executed in parallel within the same machine cycle. This kind of sprinklers is devoted to a particular refinishing of urgent shutter, needed when the (previous) cutting and polishing phases have been shortened due to the shutter's urgency. On the contrary, the other tools do not represent a scarce resource. Moreover, tool replacement is not allowed to avoid the introduction of impurities in the painting room: when a new batch arrives, the set of available tools loaded in the magazine is fixed. A key feature of the actual management is that the tools selection is performed on the ground of the workstation status with a weak link (i.e., some aggregated information) with the specific composition of the current batch.

The loading phase consists of assignment and sequencing of shutters to the parallel workstations. Each workstation has its own buffer, containing up to 10 shutters, managed by a FIFO discipline (*jobs queue*). The machine loading is carried out by human operators pursuing a workload balancing policy, in a fully autonomous fashion, without any automatic support. Their main task is to verify the compatibility of a part with the workstation before loading the part in the workstation's queue. In practice, some incompatibilities may occur after the introduction of the shutters into the painting room. In this case, they are resolved by local adjustments. In order to minimize the number of openings of the airtight gate, the loading (as well as the unloading) phase is executed once every 5 to 10 machine cycles.

A major interest of the producer is to minimize the completion time of the n parts, since the painting phase represent a bottleneck for the whole process.

In the above scenario, the minimization of the number of cycle times necessary to refinish all the shutters in a batch (*makespan*) can be carried out by maximizing, at each machine cycle, the number of parts processed. This observation gives rise to an iterative procedure for workload assignment and input sequencing (i.e., construction of the jobs queues). At each iteration (i.e., machine cycle), the maximum number of parallel operations is computed and the corresponding components are assigned to the first currently free slot of each queue. Whenever the input magazine contains at least two urgent shutters (which must be processed with priority), incompatibilities among operations may occur. The problem of maximizing the number of parallel operations within a machine cycle can be considered as a special version of the *Batch Selection Problem* (BSP) (see Van de Klundert, 1999), in which the limited resource is not the magazine capacity, but some specific tools. From now on, the restricted

(unrestricted) problem will be referred to as BSP (UBSP). In our application, the maximum number of urgent shutters in the input magazine is significantly smaller than m and the number of iterations with restrictions, in which we have to solve BSP, is about 10% of the total. In the remaining iterations we have to solve the unrestricted problem UBSP.

In the remainder of the paper we describe an auto-coordination mechanism among the workstations, which allows to: (*i*) find an optimal (i.e., maximum cardinality) solution for UBSP; (*ii*) find a locally optimal solution for BSP. The resulting procedure is shown to be flexible with respect to variation of shutters arrival pattern, loading/unloading timing and possible machine downtimes. Moreover, it does not require heavy information exchange. The procedure is based on a formulation of (BSP) as a bipartite *matching* problem with side constraints.

3. Auto-coordination protocol among autonomous workstations

Let us introduce some definitions. Given an undirected graph $G = (V, E)$, a *matching* M is a set of egdes such that no two edges in M share the same node. Let us define the following

PROBLEM 1 *Let $B = (U \cup V, E)$ be a bipartite graph of vertices $U \cup V$ and edges E. Let $E_1, \ldots, E_k$ be subsets of E and $r_1, \ldots, r_k$ positive integers. Given an integer p, the* Restricted Maximum Matching *(RMM) is the problem of determining whether there exists a matching M such that: (i) $|M| \geq p$, (ii) $|M \cap E_i| \leq r_i$, for $i = 1, \ldots, k$.*

The problem is known to be NP-complete also for $r_1 = \ldots = r_k = 1$, $i = 1, \ldots, k$ (see Itai et al., 1978).

We observe that BSP is equivalent to finding a restricted matching of maximum size on the part-workstation compatibility graph G, where a set E_i corresponds to a group of operations sharing a unique-copy tool.

Whenever restrictions do not arise, i.e., all operations are compatible with each other, the problem boils down to finding a matching of maximum cardinality in the bipartite graph G. Let us introduce the decentralized procedure for this case (i.e., UBSP), which, in our application, occurs in more than 90% of machine cycles. The extension to the general BSP is discussed in Subsection 3.2.

3.1. Optimal solution of UBSP

The purpose of this section is to show that an asynchronous coordination protocol, requiring a limited information exchange among the autonomous workstations, is sufficient to reach the global optimal solution to UBSP.

It is worthwile to recall the following basic results of matching theory. An exhaustive treatement can be found in Lovasz and Plummer (1986).

Given a matching M, we define as *matched* (*exposed*) the nodes which are (are not) endpoints of one edge in M. The set of exposed nodes w.r.t. M is denoted by $Exp_G(M)$. If $v \in V$ is matched, we denote by vv_M the matching edge. A path $p = [v_0, v_2, \ldots, v_k]$ is called *M-alternating* if $v_{i-i}v_i \in M$ iff $v_iv_{i+1} \notin M$, for $i = 1, \ldots, k-1$. An M-alternating path is called *M-augmenting* if both v_0 and v_k are exposed.

Augmenting paths yield larger matchings:

REMARK 3.1 *If P is an augmenting path w.r.t. a matching M, then the symmetric difference $M' = P\Delta M$ is a matching with $|M'| = |M| + 1$.*

The following characterization is the basis for the combinatorial algorithms for computing a matching of maximum size in a graph.

THEOREM 3.1 *A matching M in a graph G is maximum if and only if there is no augmenting path in G with respect to M.*

This implies that the computation of maximum matchings can be carried out by searching for augmenting paths. A general implementation leads, in the case of bipartite graphs, to a time complexity $O(|E| \min(|U|, |V|))$ (see also Gerards, 1995).

In order to describe the coordination protocol among workstations we consider the following implementation of the bipartite matching algorithm:

DEFINITION 3.1 *A tree T in G is called M-alternating if the following holds:*

(i) T contains exactly one exposed node, denoted by r_T;

(ii) for each node $u \in V(T)$, the path from r_T to u in T is alternating;

(iii) for each node u of degree one, other than r_T, the matching edge uu_M is in T.

The basic operation of the algorithm deals with searching for augmenting paths by growing an M-alternating tree T, rooted at some node v. Given an exposed node v we refer to this operation as SCAN(v). This is based on a breadth first search procedure. During its execution, the status of a node can be either `unreached` (set U), `candidate` (set L, managed by a FIFO discipline) or `visited` (set Z). SCAN(v) is detailed in Table 1, where the set of nodes adjacent to a node v in G is denoted by $N_G(v)$ and the set of the edges in the alternating tree is denoted by T.

Searching for augmenting paths can be carried out by scanning operations. In fact, an M-augmenting path p exists if and only if there exists a node $v \in Exp_G(M)$ such that SCAN(v) detects p. Two different implementations arise according to whether the scanning operations are executed sequentially or in parallel. Let us first describe the former. The generic iteration starts with a matching M. The algorithm keeps a set $W(M) \subseteq Exp_G(M)$ of nodes arbitrarily

```
Procedure SCAN(v);

Input:  matching M in G, v ∈ Exp_G(M);

Output: an M-augmenting path p or failure;

{
        1.  Initialization:

                T = ∅, STOP = FALSE, L = {v}, U = V − {v}, Z = ∅;

        2.  Main loop:

        while (not STOP) {

                pick the first node w ∈ L;
                L = L − {w}, Z = Z ∪ {w};

                if (w ∈ Exp_G(M)) /* w = v */

                   foreach u ∈ U, u ∈ N_G(w) {

                   if (u ∈ Exp_G(M))

                      return p = alternating path from v to u;
                      STOP = TRUE;

                   else
                      T = T ∪ {wu}, L = L ∪ {u}, U = U − {u};
                      }

                else /* w is matched */

                   foreach u ∈ U, u ∈ N_G(w), wu ∉ M {

                   if (u ∈ Exp_G(M))

                      return p = alternating path from v to u;
                      STOP = TRUE;

                   else
                      T = T ∪ {wu}, L = L ∪ {u}, U = U − {u};
                      }

        }
}
```

Table 1. Scanning an exposed node

ranked. At the beginning of an iteration $W(M) = Exp_G(M)$. Let v be the first node in $W(M)$. The algorithm removes v from $W(M)$ and executes SCAN(v). If SCAN(v) returns an augmenting path p, then M is updated as in Remark 3.1 and the next iteration can start. Otherwise, the algorithm updates v as the (new) first node in $W(M)$ and executes SCAN(v). The algorithm stops when $W(M) = \emptyset$ and no augmenting paths have been found. The time complexity is now $O(|E|(\min(|U|, |V|))^2)$, since each augmentation takes $O(|E| \min(|U|, |V|))$ time and the maximum number of augmentations is $\min(|U|, |V|)$. Although we worsen the complexity of the algorithm, in the new implementation the search for augmenting paths is executed by handling one exposed node at a time. From an efficiency point of view, the increase of complexity comes from the fact that the exposed nodes are scanned sequentially. On the contrary, if all the scan operations can be executed in parallel, then the overall complexity falls back to $O(|E| \min(|U|, |V|))$.

3.1.1. Finding a maximum matching by autonomous workstations

We now show that the described bipartite matching algorithm can be implemented at the lowest complexity (i.e., all the scanning operations are executed in parallel) as the result of local auto-coordination mechanisms among autonomous workstations.

This procedure requires a simple coordination protocol between two workstations and a few basic control facilities. In particular, we assume that each workstation u can send to any other workstation v four different messages:

1. `request`: u asks v to free the part it is currently assigned to;
2. `release`: u informs v that it is ready to release the part it is currently assigned to;
3. `failure`: u informs v that the required part cannot be released.
4. `acknowledgement`: u informs v that it is ready to accept the part which v is releasing.

Moreover, each workstation u has a two-entry memory device able to store:

(i) the name of the currently assigned part (if no parts are assigned the entry is NIL);

(ii) the name of *one* workstation which sent a request message to i (if i did not receive any request the entry is NIL).

The entry (ii) acts as a flag representing the status of the workstation: whenever the first request switches the entry from NIL to some workstation name, u blocks any further request, i.e., becomes *inactive*, until a reset signal arrives.

Forwarding a request message includes, as a first step, checking the status of the receiving workstation. If the latter is inactive, the request cannot be acknowledged and the queried workstation immediately returns a failure message.

Finally, each part is labeled with the name of the workstation to which the part is currently assigned.

Let us analyze how to implement in a distributed fashion both the search of augmenting paths and the augmenting phase in which we increase the size of the current matching (i.e., the number of processed parts). The former is based on a sequence of request messages, while the latter requires a sequence of release messages.

The procedure starts from an inclusionwise maximal matching which is the result of the autonomous actions of the workstations (i.e., no coordination among the agents). From now on, we will call matched (exposed) a workstation or a part whose associated node is matched (exposed). Moreover, we denote by $N_v(M)$ the set of workstations matched by M to the parts compatible to workstation v, $\forall v \in C$. By definition, $v \notin N_v(M)$.

Let us begin with detailing how to grow a single alternating tree, rooted at an exposed workstation v, i.e., how to implement SCAN(v). Without loss of generality, we assume that, at the beginning, all the workstations except v are active. They correspond to `unreached` nodes (see Table 1). In the first iteration, workstation v sends a request message to all workstations in $N_v(M)$. Whenever an active workstation acknowledges the first request message it becomes inactive and behaves as a `candidate` node.

Each candidate workstation u tries to satisfy the request looking for a different compatible part. We refer this action to as *request phase*. Four cases are possible. In the first two cases u gives an immediate answer:

1. u finds an exposed compatible part;
2. the current part is the only one compatible with u.

In the second two cases, u acts as an exposed workstation: it looks for a new compatible part with the aim of setting free the current one. Hence, it forwards request messages to all the workstations in $N_u(M)$:

3. all the requests are refused (i.e., all the queried workstations are inactive);
4. at least one query is accepted.

If condition 1 occurs, an augmenting path has been found. On the contrary, both conditions 2 and 3 lead to a failure, and workstation u immediately returns a failure message to the requiring workstation. Finally, if condition 4 holds, the request phase is iterated by each of the queried workstations.

The search for augmenting paths fails if conditions 2 or 3 occur for all candidate workstations. In this case, the current matching cannot be augmented by SCAN(v). When v completes the request phase, it corresponds to a `visited` node in SCAN(v).

An example is depicted in Fig. 2. The initial matching is $M = \{1a, 2b, 4c\}$. The sequence of requests $(3 \rightarrow 2), (2 \rightarrow 1)$ fails since workstation 1 cannot free part a. On the contrary, the sequence $(3 \rightarrow 4)$ succeeds by detecting the augmenting path $p = [3, c, 4, d]$.

Let us now discuss the execution of parallel scan operations. We show that

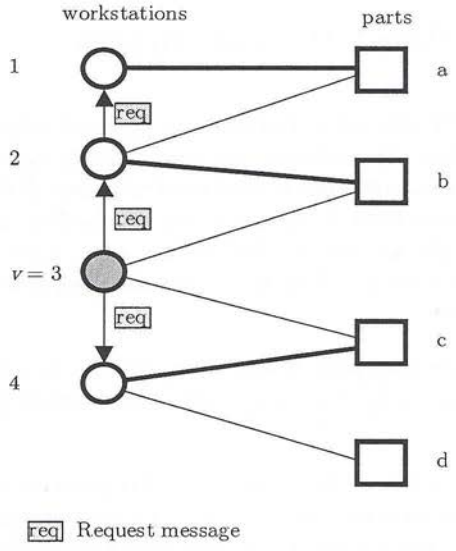

Figure 2. Scan operation by request messages

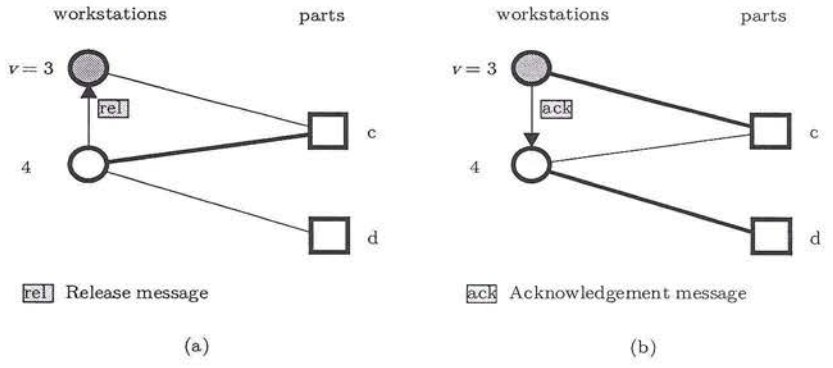

Figure 3. The augmenting phase

it can be carried out with the same coordination protocol supporting the single scan operation. To explain this, suppose that two different exposed workstations u and v start two different sequences s_1 and s_2 of requests. If the two sequences converge to the same (matched) workstation, say w, the first sequence reaching w will continue, while the second will be blocked since after the first request workstation w becomes inactive. This fact does not compromise any potential augmenting path. In fact,

REMARK 3.2 *An augmenting path exists from node u to an exposed node z containing node w if and only if an augmenting path exists from node v to z containing node w.*

Hence, a simple FIFO rule in the acceptance of request messages leads to an exhaustive search for augmenting paths by parallel scan operations.

If the search phase is successful, we are left with the problem of performing augmentation in a decentralized fashion. Consider a queried workstation w, currently assigned to part w_M able to detect an exposed compatible part $u \neq w_M$ (in Fig. 2, $w = 4$, $w_M = c$ and $u = d$).

Workstation w stores the name of the workstation v which first required w_M (in Fig. 2, $v = 3$) and it is now ready to answer with a release message indicating that part w_M is going to be free. This process is repeated by each workstation which received a request message, whenever a release message allows it to switch its current assignment. Notice that each release message determines univocally the new assignment (due to the fact that only the first request message is stored). The matching is updated by a backward sequence of acknowledgement messages.

The augmenting phase is shown in Fig. 3: workstation 4 informs workstation 3 that part c can be released (3a); the acknowledgement of this message leads to the augmenting (3b). The new matching is $M = \{1a, 2b, 3c, 4d\}$.

Notice that a rule is needed to handle possibly multiple release messages

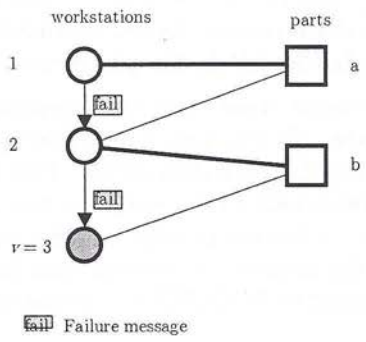

Figure 4. Backward sequence of failure messages

arriving at a workstation. Also in this case, using an argument similar to the one expressed in Remark 3.2, we can observe that the FIFO discipline supports the correct auto-coordination. Moreover, whenever a workstation acknowledges a release, it becomes matched and replies with a failure to other releases. A failure message is propagated backward so as to reset the workstations which are ready to free their current part.

An important aspect deals with the system reset. In Fig. 2, workstation 2 has been blocked by the request of 3 and enters the request phase. It consists of a further accepted request from 2 to 1, which immediately returns a failure. This resumes the active status of 2. This mechanism is general: an inactive workstation v becomes active whenever *all* its requests have been answered with a failure message. At the same time v forwards a failure message to the (unique) requiring workstation. The situation is depicted in Fig. 4.

The total amount of information stored (i.e., space complexity) is $O(n+m)$. From the analysis of time complexity (see Subsection 3.1) we conclude that the total number of messages (at each machine cycle) is $O(nm^2)$.

3.2. Heuristic solution of BSP

In this section we discuss the implementation details of the described procedure and extend it so as to obtain a heuristic algorithm for BSP.

The painting workstations are actually equipped with a simple device able to control temperature, composition of the paint, and operating mode. The implementation of coordination protocols requires a higher data processing capability. A suitable equipment for the workstation is the *Programmable Logic Control* (PLC), commonly operating in FMSs. In fact, PLC can perform fast peripherals polling and sequential queries of large archives (see Studebacker, 1996), which are executed many times in the proposed procedure. Moreover,

the computational complexity analysis shows that the size of the instances at hand (see Section 2) lead to a slight computational workload, and data processing time does not represent a bottleneck for the procedure.

The first implementation issue deals with the construction of the part-workstation compatibility graph. To this aim, a new field <part_dim> is needed in the part label, containing the shutter's dimensions. This information has to be made available to the workstations PLCs outside of the airtight room by a memory device PART_MEM (i.e., a blackboard-like system) shared among workstations. This must contain the number of shutters, and, for each shutter, its dimensions. The complete record in PART_MEM is summarized in the following table.

Record: <part_id, part_status, part_dim, curr_assignment, opn_type>
<integer><part_id>: shutter's name;
<boolean><part_status>: 1 if the shutter has been assigned in a previous iteration and 0 otherwise;
<integers pair><part_dim>: pair (w, h), w (h) is the shutters width (height);
<integer><curr_assignment>: workstation the shutter is currently assigned to;
<integer><opn_type>: type of painting;

The field <part_status> has been introduced to eliminate from the current instance of BSP the parts processed at previous steps.

In order to solve BSP, each workstation must check the availability of the required set of tools before taking the part. To this aim, a field <opn_type> in each element of PART_MEM stores the type of the painting operation required. Each workstation, depending on its characteristics, reads <opn_type> and builds the set of necessary tools, whose availability is checked by a query to a second shared memory device storing the tool status. We denote such a device TOOL_MEM. Its size equals the number of tools and each record i contains the number of copies of tool i currently available.

This *tool checking procedure* may be executed (i) in the construction of a starting solution and (ii) in the search of augmenting paths. In case (i) a possible strategy is to rank the workstations and build the assignment according to this ranking. Another possibility could be to accept a starting solution carried out by human operators, as in the current management.

As far as case (ii) is concerned, during the request phase, the current candidate workstation executes a tool checking procedure before forwarding the request message to verify if the operation is executable w.r.t. the current workstations configuration. In case of failure, the request is not forwarded. In this case, the search for augmenting paths by (sequential or parallel) scanning operations can fail even if an augmenting exists and the algorithm returns a locally optimal solution.

4. Conclusions

We proposed a distributed procedure able to: (*i*) find the maximum number of parallel operations in an "unrestricted" machine cycle (UBSP) and (*ii*) return a locally optimal solution whenever restrictions on tools availability occur (BSP). The validity of the approach resides upon two facts. First of all, the number of restricted machine cycles represents less than 10% of the total. Second, the heuristic algorithm can be applied to any starting solution, such as the one built by human operators. In other words, it can be used for tentatively improving the starting solution.

The procedure does not need any centralized controller and has been shown to require manageable information flows. Another nice feature is its robustness with respect to different parts arrival patterns and to machine downtimes.

A further research direction deals with the investigation of system architectures able to guarantee good performances to distributed control procedures. In our case this can be carried out by a sensitivity analysis with respect to different configurations of the part-workstation compatibility graph.

References

ARBIB, C. and ROSSI, F. (1999) Optimal Resource Assignment through Negotiation in a Multi-agent Manufacturing Environment. Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, Report 14.

ADACHER, L., AGNETIS, A. and MELONI, C. (1999) A Simulation Study of Autonomous Agents in FMSs. Dipartimento di Informatica e Automazione, Università di Roma Tre, Technical Report 43-99, to appear in *IIE Transactions on Design and Manufacturing.*

DECKER, K.S. and LESSER, V. (1994) Communication in the Service Coordination. Department of Computer Science, University of Massachusetts.

DELLA CROCE, F., MENGA, G., TADEI, R., CAVALOTTO, M. and PETRI, L. (1993) Cellular control of manufacturing systems. *EJOR*, **69**, 498-509.

GERARDS, B. (1995) Matching. *Network Models.* Volume in: *Handbook in Operations Research and Management Science.*

GOU, L., HASEGAWA, T., LUH, P.B., TAMURA, S. and OBLAK, J.M. (1994) Holonic planning and scheduling for a robotic assembly testbed. *Proceedings of 4th Rensselaer International Conference on Computer Integrated Manufactoring and Automation Technology.* Rensselaer, NY, October.

GOU, L. and LUH, P.B. (1997) Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Tokyo, Japan, June 16-20.

GRAVES, S.C. (1982) Using Lagrangean techniques to solve hierarchical production planning problems. *Management Science*, **28**, 3.

ITAI, A., RODEH, M. and TANIMOTO, S. (1978) Some Matching Problems for

Bipartite Graphs. *Journal of the Association for Computing Machinery*, **25**, 4, 517–525.

LIN, G.Y. and SOLBERG, J.J. (1992) Integrated shop floor control using autonomous agents. *IIE Transactions*, **24**, 3, 57–71.

LOVÀSZ, L. and PLUMMER, M.D. (1986) *Matching Theory*. North Holland, Amsterdam.

STUDEBACKER, P. (1996) PLC or PC? *Control Magazine*, **9**, 11, 24 - 30.

VAN DE KLUNDERT, J. (1999) Scheduling Problems in Automated Manufacturing. Faculty of Economics and Business Administration, University of Limburg, Maastricht, The Netherlands, Dissertation no. 96-35.