Università degli Studi di Roma
"Tor Vergata"

Doctoral Thesis

in

Ingegneria Economico - Gestionale

MANAGING DISTRIBUTED FLEXIBLE MANUFACTURING SYSTEMS

Riccardo Onori

Advisor: Prof. Lucio Bianco

December 2004

Author's Address:

Riccardo Onori
Dipartimento di Ingegneria dell'Impresa,
Università degli studi di Roma "Tor Vergata"
Viale del Politecnico 1 - 00133 Rome - Italy
Email: onori@disp.uniroma2.it

Dissertation submitted to the Department of Enterprise Engineering, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Ingegneria Economico - Gestionale at the University of Rome "Tor Vergata".

Contents

List of Figures

List of Tables

Preface

Even in these post-industrial times, manufacturing is one of the cornerstones of our society. For several years, research has focused on several aspects of manufacturing, from the individual processes towards the management of virtual enterprises, but several aspects, like coordination and control, still have relevant problems in industry and remain challenging areas of research.

The application of advanced technologies and informational tools by itself does not guarantee the success of control and integration applications. In order to get a high degree of integration and efficiency, it is necessary to match the technologies and tools with models that describe the existing knowledge and functionality in the system and allow the correct understanding of its operation. In a global and wide market competition, the manufacturing systems present requirements that lead to distributed, self-organised, co-operative and heterogeneous control applications.

A *Distributed Flexible Manufacturing System* (DFMS) is a goal-driven and data-directed dynamic system which is designed to provide an effective operation sequence for the products to fulfil the production goals, to meet real-time requirements and to optimally allocate resources. In contrast to a centralized or hierarchical structure, the distributed control architecture reveals its advantages not only in providing efficient and parallel processing, but also in flexible realizing of system integration and maintenance. It is proved to be more suitable for a complex and heterogeneous system where the enormous control complexities can be distributed to each separate component and thus to gain a high system performance. However, on the other hand, it results in the high autonomy and intelligence requirements for individual components.

This thesis develops two models for managing two relevant issues in DFMS, the *task mapping* problem and the *task scheduling with multiple shared resources* problem.

Both arise at the operational level of the manufacturing system management (see Section 1.2). A DFMS is designed to process a large variety of tasks in a prespecified time. The challenges at operational management level are the real-time coordination of operations between the parts (facilities, department, cells) of the distributed manufacturing system, and the effective scheduling of jobs in the atomic and operative parts (robotic cells).

The former challenge is addressed with the task mapping problem, which con-

cerns how to divide the set of jobs and assign them to the parts of the system. In this thesis we study the case in which the jobs are quite homogeneous, do not have precedence constraints, but need some communications to be coordinated. So, jobs assignment to different parts cause a relevant communication effort between those parts, increasing the managerial complexity. The issue here is to assign the jobs to reduce the communications, and to balance the work-load between the parts.

The latter challenge is the effective sequencing of operations in the atomic parts of the manufacturing systems. Each distributed part of a DFMS is usual composed by some automatic robotic cells, where the jobs are performed. The issue here is that typically the jobs performed need the use of more than a single resource within the cell, and the decisions involve which jobs process in a given unit time and using which resources, in order to achieve a reduction of the completion time of all the jobs in the system.

This thesis is organized as following.

In Chapter 1 DFMS are presented, and a layered approach for the modeling of such production systems is proposed. According to that modeling representation a DFMS may be seen as multi-layer resource-graph such that: vertices on a layer represent interacting resources; a layer at level l is represented by a node in the layer at level $l-1$.

In Chapter 2 the Task Mapping problem in DFMS is addressed. We show that the standard models usually used to formal represent such a problem are wrong and lack of expressibility. Through some graph theoretical results we relate the problem to the well-known *hypergraph partitioning problem* and briefly survey the best techniques to solve the problem. A new formulation of the problem is then presented. Some considerations on the improved formulation permit the computation of a *good* Lower Bound on the optimal solution in the case of the *hypergraph bisection*.

In Chapter 3 the Task Scheduling with multiple shared resources problem is addressed for a robotic cell. We study the general problem of sequencing multiple jobs, where each job consists of multiple ordered tasks and tasks execution requires simultaneous usage of several resources. NP-completeness results are given. A heuristic with a guarantee approximation result is designed and evaluated.

Chapter 1

Distributed and Flexible Manufacturing Systems

1.1 Introduction

World-wide competition among enterprises led to the need for new systems to perform the control and supervision of distributed manufacturing, through the integration of information and automation islands.

The market demands should be fulfilled by manufacturing enterprises to avoid the risk of becoming less competitive.

The adoption of new manufacturing concepts combined with the implementation of emergent technologies, is the answer to the improvement of productivity and quality, and to the decrease of price and delivery time.

Nowadays, the *distributed manufacturing* organisational concept, e.g. the *virtual enterprises*, requires the development of decentralised control architectures, capable to react to disturbances and changes in their environment, and capable to improve its performance.

The mass manufacturing, idealised by Henry Ford was a strap down system, incapable to treat variations in the type of product. This rigidity started to be an obstacle and with the worldwide competitiveness the mass manufacturing became viable only in some products. In this way, the mass manufacturing emerged from the era of manufacturing to the era of mass customisation.

Nowadays, each product has several models, and each model can be highly customised in order to fulfil the requirements of the customers (a good example is the

automobile industry), which requires an automated job shop type of production in a truly coordinated production system.

Coordination and effectiveness can be reached only by an appropriate organization and management of the manufacturing system, therefore by a efficient and effective implementation of manufacturing control.

1.2 Manufacturing Control (MC)

1.2.1 Definition of Manufacturing Control

In a factory, a lot of activities have to be performed to produce the right goods for the right customer in the right quantities at the right time [8]. Some activities are concerned with the physical production and manipulation of the products. Other activities refer to the management and control of the physical activities. These production management and control activities can be classified in strategic, tactical and operational activities, depending on the long term, medium term or short term nature of their task [19, 12, 67].

Strategic production management issues relate to the determination of the products to be designed, manufactured and sold, considering the markets and the customer expectations. It also relates to the design of the appropriate manufacturing system to produce these products, including the generation of a master schedule to check whether the manufacturing system has enough capacity for the estimated market demand.

Tactical production management issues refer to the generation of detailed plans to meet the demands imposed by the master schedule, including the calculation of appropriate release and due dates for assemblies, sub-assemblies and components. These detailed plans may refer to predicted orders, in a make-to-stock environment, or to real customer orders, in a make-to-order environment. Typically, MRP or MRP-II systems (Material Requirements Planning and Manufacturing Resource Planning) carry out this task.

Operational production management issues relate to the quasi-real time management of the manufacturing system on the shop floor. It involves commanding the appropriate machines, workers and other resources in a co-ordinated way, thereby selecting the appropriate resources for each task and the appropriate sequencing of these tasks on each resource. It includes the detailed planning and optimisation to

fulfil the requirements of the MRP schedule as good as possible.

Manufacturing Control (MC) refers to the management and control decisions on this operational level, and is the decision making activity concerned with the short-term and detailed assignment of operations to production resources [18].

1.2.2 Manufacturing Control Functions

The main function of Manufacturing Control is **resource allocation**. As stated above, resource allocation is also performed at the tactical and strategic level of production management. MC refers to the operational level of resource allocation only. This usually includes a detailed short-term **scheduler**, which plans and optimises the resource allocation on beforehand, considering all production requirements and constraints. Typically, scheduling has a total time span ranging between a few weeks and a day, and a time granularity ranging from days till below minutes. When the schedule is calculated, it is implemented on the shop floor by a **dispatcher**, taking into account the current status of the production system. This function is called **on-line manufacturing control** (OMC). During OMC, the status of the vital components in the system is monitored.

Manufacturing Control also covers process management [62]. Process management includes machine and process monitoring and control, but also the operational level activities for process planning. The **process planning** function defines how products can be made, including the overall process plan defining the possible sequences of operations for to manufacture a product and the detailed operation process planning, prescribing the parameters of the individual manufacturing steps. In traditional manufacturing systems, process planning provides manufacturing control with a single sequence of operations to be executed for each product. Recently, process planning is considering more flexible solutions, based on the use of alternative routings, and alternative machine allocations.

The use of alternative routings, modelled e.g. by precedence graphs [27], dynamic precedence graphs [74] or Petri nets [24] enables manufacturing control to select the best sequence of operations depending on the machine load and unforeseen disturbances.

The use of alternative resources for an operation gives similar opportunities. Process management also includes operational aspects of manufacturing system design, layouting and configuration. Depending on the flexibility of the manufacturing sys-

tem, the manufacturing system may be adaptable on short term. Those short-term modifications and set-ups are managed by manufacturing control.

MC further needs to provide information to people and machines at the right moment. It therefore has to collect, store and retrieve these data and provide them to the people in the right quantity and level of aggregation. This function includes data capturing from the machines and the analysis of these raw data. It also is concerned with the management of the programs for NC-machines, and with quality and labour management.

Manufacturing control relates to several other activities in an enterprise, like marketing, design, process planning, sales, purchasing, medium and long term planning, distribution, and servicing, to which it has to interface. Via the sales function, new orders are introduced in the shop. Orders can be directly assigned to customers, or they may be stock replenishment orders, in a make-to-stock environment. In industry, very often mixtures of a make-to stock and make-to-order environment are found and pragmatically dealt with. Traditionally, manufacturing control considers sales as a black box, generating orders with a stochastic distribution. Medium-range planning provides MC control with due dates and release dates for orders. It defines for a large amount the performance objectives for manufacturing control.

1.2.3 Classification of Manufacturing Systems

The characteristics of a manufacturing system, like production volume, flexibility and layout, have a major influence on its control.

The complexity of the resource allocation optimisation (*scheduling*, see below) and on-line control problems depend strongly on the layout. Therefore, manufacturing control strategies differ considerably according to the corresponding layout.

Manufacturing systems and subsystems are often categorised according to their production volume and their flexibility.

Traditionally, the targeted production volume and flexibility of a manufacturing system are inversely proportional: high volume manufacturing systems have a limited flexibility.

Typical production types range from continuous production, over mass production, large batch manufacturing, discrete manufacturing, to one-of-a-kind and project-wise manufacturing.

This thesis is restricted to discrete manufacturing in a distributive environment.

Within discrete manufacturing, manufacturing systems are classified by their layout (usually related to the way products flow through the manufacturing system).

The rest of this subsection surveys the main production layouts relevant for this thesis.

The **single machine model** [8] has the most simple layout. It consists of a single machine that performs all operations. Therefore, orders usually are not further decomposed into operations.

The **identical processors model** [8] is similar to the single machine model, but consists of multiple, identical machines.

A **flow shop** consists of different machines, processing orders with multiple operations, where:

- each operation is to be executed on a specific machine;
- all operations of an order have to be executed in a fixed sequence;
- all orders have to visit the machines in the same sequence.

The main features of flow shop systems are high *through-put*, low variability in the output product-mix (often, only one type of product is processed by a line), short *flow time*, low *work-in-process* (*WIP*), high machine utilization rate, uniform quality, high automation, high investments, high unreliability (risk of production-stops in case of a machine breakdown), and high time to market for developing a new product/process.

A **job shop** is the same as a flow shop, except that each order can visit the machines in a different sequence. Hence, the workload is characterized by different products concurrently flowing through the system. Each part requires a series of operations which are performed at different work stations according to the related process plan. Some work centers are interchangeable for some operations, even if costs and quality standards are slightly different from machine to machine. This feature greatly increases the flexibility of the system and, at the same time, the cost and quality variability in the resulting output.

The manufacturing control system is responsible to choose the best option based on the status of the system. In a job-shop, the machines are generally grouped together by technological similarities. On one side, this type of process-oriented layout increases transportation costs due to the higher complexity of the material-handling

control. On the other side, manufacturing costs decrease due to the possibility of sharing common resources for different parts.

The main features of a job-shop are high flexibility, high variability in the output product-mix, medium/high flow time, high WIP, medium/low machine utilization rate, non-uniform quality, medium/low automation level, medium/low investments, high system reliability (low risk of production-stop in case of a machine breakdown), and for developing a new product/process.

A **cell manufacturing system** is a job shop in which the machines are grouped in work-cells. In fact, following the criteria of group technology, some homogeneous families of parts may be manufactured by the same group of machines. A group of machines can be gathered to form a manufacturing cell. Thus, the manufacturing system can be split into several different cells, each dedicated to a product family. Material-handling cost decreases, while at the same time flexibility decreases and design cost increases. The main features of cell-manufacturing systems range between the two previously mentioned sets of system characteristics.

This thesis is focused on the cellular job shop layout, as it is the most general case, and as it reveals the most suitable system to reach the flexibility in a distributive environment.

Those layout models refer to more specific manufacturing systems, like factories with a bottleneck, lines, flexible manufacturing systems (FMS), or flexible flow shops (FFS). For some aspects of the control of factories with a bottleneck, the single machine model performs quite adequately [8, 32].

Assembly lines usually are flow shops with limited or no opportunities for orders overtaking each other.

Flexible manufacturing systems are extended job shops [25], where:

- operations of orders follow a sequence allowed by a precedence graph instead of a fixed operation sequence;
- different workstations can perform the same operation;
- transport and set-up have to be modelled.

Recently [55], flexible flow shops are becoming more important: they inherit the layout and properties of flow shops, but have an increased flexibility with respect to machines selection, operation sequencing and sometimes even routing flexibility.

In this respect, they are mathematically more similar to job shops. However, while all routings are feasible, some routings are highly preferable. In this way, they combine the efficiency and transparency of a flow line with the reactivity and flexibility of a job shop. Such manufacturing systems are promising to break the compromise between flexibility and high volume production as it is found in traditional manufacturing systems.

1.2.4 Flow Shop and Job Shop: characteristics and limitations

A comparison of the Flow Shop (FSM) and Job Shop (JSM) manufacturing systems can be summarized as follows. The main difference occurs in the production capability for the former and the technological capability for the latter. This is translated into high throughput for FSM and high flexibility for JSM. A number of scheduling issues become apparent during different phases in these systems. These phases and different issues are:

1. Design phase. In case of FSM, great care is to be taken during this phase. FSM will operate according to its design features, therefore, if the speed of a machine is lower than the others, it will slow down the entire line, causing a bottleneck. Similar problems will occur in the case of machine breakdowns. Availability and maintenance policies for the machines should be taken into account during the design phase. Higher levels of automation generate further concern during the design phase because of the risk stemming from the high investment and specialization level, e.g., risk of obsoleteness. On the other hand, the JSM is characterized by medium/low initial investment, a modular structure that can be easily upgraded and presents less problems in the design phase. The product-mix that will be produced is generally not completely defined at start-up time, therefore, only the gross system capacity may be estimated on the basis of some assumptions about both processing and set-up time required. The use of simulation models highly improves this analysis.

2. Operating phase. The opposite occurs during the operating phase. In an FSM, scheduling problems are solved during the design stage, whereas in a JSM, the complexity of the problem requires the utilization of a production activity control (PAC) system. A PAC manages the transformation of a shop order from

the planned state to the completed state by allocating the available resources to the order. PAC governs the very short-term detailed planning, executing, and monitoring activities needed to control the flow of an order. This flow begins the moment an order is released by the planning system for execution, and terminates when the order is filled and its disposition completed. Additionally, a PAC is responsible for making a detailed and final allocation of labor, machine capacity, tools, and materials for all competing orders. Also, a PAC collects data on activities occurring on the shop floor involving order progress and status of resources and makes this information available to an upper level planning system. Finally, PAC is responsible for ensuring that the shop orders released to the shop floor by an upper level planning system, i.e., manufacturing requirement planning (MRP), are completed in a timely and cost effective manner. In fact, PAC determines and controls operation priorities, not order priorities. PAC is responsible for how the available resources are used, but it is not responsible for determining the available level of each resource.

Scheduling in a job-shop is further complicated by the dynamic behavior of the system [8, 12]. The required output product-mix may change over time. Part types and proportion, deadlines, client requirements, raw material quality and arrival time, system status, breakdowns, bottlenecks, maintenance stops, etc., are all factors to be considered. A dynamic environment represents the typical environment in which a JSM operates. The complexity of the job-shop scheduling problem frequently leads to over-dimensioning of the system capacity and/or high levels of WIP. A machine utilization coefficient may range between 15-20% for nonrepetitive production. Some side effects of WIP are longer waiting time in queue and a manufacturing cycle efficiency (MCE) ranging from 1/25-1/30 for the job-shop, compared to approximately one for line manufacturing. MCE is defined as the ratio between the total processing time necessary for the manufacturing of a part and the flow time for that part, which is equal to the sum of total processing, waiting, setup, transporting, inspection, and control times.

1.3 Scheduling

1.3.1 Definition

Scheduling is defined as the process of *optimising* resource allocation decisions *on beforehand*.

Resource allocation is deciding for all tasks when to happen and with which resources [8, 65]. In literature, the difference between these two terms is not always clear. For this thesis however, the difference is important.

Scheduling is an optimisation process, and thus refers to one or more **goals** (also called **objectives**, or **performance criteria**). Typical performance criteria in a manufacturing context are:

- the **throughput** (Q), defined as the number of orders the manufacturing system finishes per time unit;
- the **work-in-process** inventory (WIP), defined as the number of orders in the system which are not finished yet;
- the mean order **flow time** (or **lead time**, $\bar{F}$), defined as the difference between the order finish time and the order start time);
- the mean order **tardiness** $\bar{T}$ (The tardiness is the difference between the order finish time and the due date, if this difference is positive. Otherwise, the tardiness is 0.).

The resource allocation process usually is subject to constraints. In practical situations, not all possible resource allocation decisions are feasible. Typical constraints express the limited capacity of a resource, or precedence relations between operations.

1.3.2 Mathematical Model for Job Shop Scheduling

For a clear understanding of scheduling and manufacturing control, it is often helpful to define a mathematical model of the scheduling problem. Such a model is strongly dependent on the specific manufacturing system focused at. Therefore, this subsection defines a mathematical model for (extended) job shop scheduling that is directly related to the problem under the study scope of this thesis.

The model can be represented as follows.

To define the symbols, suppose the manufacturing system consists of M workstations m_k, with k ranging from 0 till $M-1$. Let Ω denote the set of workstations. There also are A auxiliary resources r_a in the system, with a ranging from 0 till $A-1$. If necessary, these auxiliary resources can be distributed among the workstations.

Consider the set of orders O, with N orders i, for i ranging from 0 till $N-1$. Each order i has:

- a release date R_i, denoting the time an order enters the system;
- a due date D_i, denoting the time the order should be ready;
- a weight w_i, denoting the importance of an order.

Each order i has a number of operations (i,j), for j ranging from 0 to N_i-1. Each operation (i,j) can be executed on a set of alternative workstations

$$H_{ij} = \{h_{ijl} \in \Omega | h_{ijl} \text{ can execute operation } (i,j)\},$$

If operation (i,j) can be executed on workstation k (i.e. $m_k \in H_{ij}$), it has a duration d_{ijk}, otherwise, d_{ijk} is undefined. If the duration is the same for all alternative workstations, d_{ijk} can be simplified to d_{ij}.

Given these data, the resource allocation process selects which workstation h_{ij} will execute operation (i,j) and at which time b_{ij} it will start. In other words, h_{ij} and b_{ij} are the decision variables. c_{ij} is the resulting completion time of operation (i,j):

$$c_{ij} = b_{ij} + d_{ijh},$$

where d_{ijh} is short for $d_{ijh_{ij}}$. C_i is the order completion time:

$$C_i = c_{i,N_i-1}.$$

The resource allocation problem is subject to capacity constraints and precedence constraints. Capacity constraints express that a machine can only execute one operation at the same time. To introduce a mathematical formulation of the capacity constraints, Hoitomt [41] and Luh [59] use a formulation with discretised time t and Kronecker deltas:

$$\sum_{ij} \delta_{ijtk} \leq M_{tk}, \quad (t = 1, \ldots, T;\ k = 0, 1, \ldots, M-1)\,,$$

where M_{tk} is the capacity of workstation m_k at time t, and $\delta_{ijtk} = 1$ if operation (i, j) occupies workstation m_k at time t, and otherwise $\delta_{ijtk} = 0$. T is the time horizon.

Precedence relations can be modelled with sequences of operations [65], with precedence graphs [42, 65], simple fork-join precedence constraints [59], assembly trees [55], dynamic precedence graphs [74] or non-linear process plans [24].

The experienced reader may notice that this representation of the scheduling problem does not yet cover all situations and does not model all possible sources of flexibility in full detail. For instance, and-or graphs can better model choices among processes that imply different sets of operations. For machines with multiple capacity, the exact modelling of set-ups and loading becomes more complex. However, complete modelling of the scheduling problem becomes too complex to address in a generic way and too problem specific to describe in this thesis.

The model of the scheduling problem described in this thesis is specific for a particular distributed and flexible environment, where there is a huge set of jobs that must be processed by a set of resources. Each job can be processed on each distributed resource, and there are no precedence constraints between jobs.

An **active schedule** is defined as a schedule where no operation can be started earlier without delaying another operation or violating some constraints.

A **non-delay schedule** is a schedule where no workstation remains idle if an executable operation is available. In other words, if in a non-delay schedule, a workstation is not occupied on time t, then all operations that can be executed on that workstation are scheduled before time t, or cannot be scheduled on time t because of precedence constraints. Dispatching rules can only produce non-delay schedules. For regular performance measures, the optimal schedule is always an active schedule. Non-delay schedules are easier to calculate, but make no statement on optimality.

1.3.3 Complexity of Scheduling problems

The best-known difficulty of the scheduling problem is its computationally hard nature. If the size of the problem grows, the time required for the computation of the

optimal solution grows rapidly beyond reasonable bounds. Formally speaking, most scheduling problems are NP-hard or NP-complete. An NP problem – *nondeterministically polynomial* – is one that, in the worst case, requires time polynomial in the length of the input for solution by a non-deterministic algorithm [29].

Non-deterministic algorithms are theoretical, idealised programs that somehow manage to guess the right answer and then show that it is correct. An NP-complete problem is NP and at least as hard as every other NP problem. An NP-hard problem is NP-complete or harder than NP. NP-completeness proofs are available for a number of simple scheduling problems, and realistic problems tend to be even more complex. The practical consequence of NP-hardness is that the required calculation time for finding the optimal solution grows at least exponentially with the problem size.

In other words, for realistic scheduling problems, where finding an optimal solution could require thousands of centuries, it is even useless to consider optimal algorithms. This implies that nearoptimal scheduling algorithms are the best possible ones. A *near optimal scheduling* algorithm spends time on improving the schedule quality, but does not continue until the optimum is found.

1.4 On-Line Manufacturing Control

Dispatching is the implementation of a schedule taking into account the current status of the production system [12].

This definition takes the existence of a scheduler for granted. The dispatcher has to execute the schedule, reacting to disturbances as quickly as possible. Therefore, the dispatcher has no time to optimise its decisions. The task of a dispatcher seems to be straightforward, such that usually, only simple heuristics are used for decision taking.

A more general term is **on-line manufacturing control**. As suggested by its name, on-line manufacturing control refers to that part of manufacturing control that takes immediate decisions. Traditionally, it is a synonym of dispatching. However, some new manufacturing paradigms that do not use scheduling, require a more general term that is not based on scheduling. Moreover, in the manufacturing world, dispatching refers to simple decision making, while on-line manufacturing control also refers to more advanced methods.

However, due to the NP-complete nature of resource allocation, dispatching in manufacturing quickly got the meaning of applying simple priority rules. Using an analogy, on-line manufacturing control is the task of a foreman. Similar to dispatching, on-line manufacturing control has to react quickly to disturbances, such that it cannot optimise the performance with time-intensive computations.

On-line manufacturing control also covers some more practical functions. It includes the management, downloading and starting of NC-programs on due time. While Bauer [12] defines monitoring as a separate module, data acquisition and monitoring is often performed by the on-line (manufacturing) control system.

So, more are the data needed more is the complexity with which the OCM must manage.

Finally, note that On-line manufacturing control is not involved with process control, like adjusting the parameters in the machining process or calculating the trajectory of a robot.

1.5 Distributed Flexible Manufacturing Systems (DFMS)

On today's global and highly competitive market, enterprises must be aware of momentary market opportunities, and quickly and properly react to customers' demands.

A *Distributed Flexible Manufacturing System* (DFMS) is a goal-driven and data-directed dynamic system which is designed to provide an effective operation sequence for the products to fulfil the production goals, to meet real-time requirements and to optimally allocate resources [11].

In contrast to a centralized or hierarchical structure, the distributed control architecture reveals its advantages not only in providing efficient and parallel processing, but also in flexible realizing of system integration and maintenance, thus providing the right solutions to the competitiveness requirements.

In fact:

- the increase of product diversity over time expands associated risks and costs, which are sometimes prohibitive, while distributing responsibilities by multiple entities, risks and costs become acceptable and market opportunity can be achieved;

- the increase of technological complexity enforces enterprise to acquire knowledge in non-fundamental domains, which implies increased time-to-market periods, while distributing competencies by different enterprises, each one maintains its core competency while achieving the market opportunity;
- market globalisation virtually increases both enterprise opportunities and risks. Each enterprise has to operate in the global market with globally based enterprises supplying global products. However, developing relations and partnerships with such enterprises, impairs challenges and risks while benefiting from a wider market.

Different management approaches have been adopted, related to different levels of partnership, trust and dependency between enterprises:

- *Supply Chain* management, characterized by rudimentary relationship between supplied and supplier, tasks and technological competencies distribution, but centralizing strategies and risks;
- *Extended Enterprise*, where entities develop more durable, coupled and mutual intervening relation, sharing technological and strategic efforts. Yet, supplied entity maintains a dominant position over suppliers;
- *Virtual Enterprise*, is a very dynamic and restructuring organization, where supplier and supplied are undifferentiated and no dominant position exists.

Although previous description relates to inter-enterprise context, the same characteristics and behaviours (distribution, decentralization, autonomy and dependency) are also suggested in an intra-enterprise context. Intra-enterprise workgroups emphasize self-competencies while combining efforts for a global response to external requirements.

Distributed Flexible Manufacturing System is an abstract concept (i.e. a class of systems) characterized by a set of common features and behaviours, with several specific characterizations (i.e. instantiations), named *organizational paradigms*.

1.5.1 DFMS Properties

DMS are characterised by several properties and behaviours. Such features relate both to the overall system and to each composing entity. We may classify

those properties in *basic properties* (i.e., those properties that define a DFMS) and *characterizing properties* (i.e., those properties that characterize DFMSs from each other).

Basic properties

Autonomy - An entity is said to be autonomous if it has the ability to operate independently of the rest of the system and if it possesses some kind of control over its actions and internal state [17], i.e. autonomy is the ability of an entity to create and control the execution of its own plans and/or strategies, instead of being commanded by other entity (e.g. a master/slave relationship);

Distribution - A system is said to be distributed if different entities operate in the system;

Decentralization - Decentralisation means that an operation/competency can be carried out by multiple entities;

Dynamism - Refers to changes in the manufacturing system's structure and behaviour during operation. This expresses different competencies, responsibilities and relations between entities;

Reaction - An entity is said to be reactive if it adjusts its plans according to its perceptions;

Characterizing properties

Flexibility - Flexibility is the ability the system exhibits during operation that allows it to change processes easily and rapidly in a predefined set of possibilities each one specified as a routine procedure, defined ahead of time so that the needs to manage it are in place. In manufacturing, Flexibility is related to physical flexible machinery. Flexible is in the sense that machines (or cells) are able to execute several operations. In addition, they can quickly change among different production plans according to the part's type to manufacture at a given point in time. The concept of Flexible Manufacturing System (FMS) is very popular with companies which produce small lots of each product, mixing different lots in the production flow. One of the main problems in achieving flexibility is related to transportation. Since a product will need

pass through several workstations in order to be manufactured and different products will have different routes, the transport links between workstations should be as *free* as possible.

Adaptability - Adaptability is the manufacturing system ability to be maintained easily and rapidly, in order to respond to manufacturing requirements, based on its shop floor constraints. Adaptability refers to production facilities reconfiguration and scalability; workforce that has the incentive and flexibility to respond creatively to customer needs and thus requires flexibility. Ill-specification is a well-known synonym. A system is said to be adaptable if it can continue to operate in the face of disturbances changing its structure, properties and behaviours accordingly to new situations it encounters during its *life-time*. A disturbance is any event not previously and formerly specified (e.g. machine breakdown or a new type of product to manufacture). However, it is very hard to predict every disturbance that may occur.

Agility - Agility is understood as a management paradigm, consisting of different approaches in multiple organisational domains. However, Agility presumes system empowerment, achieved by continuous observation searching for new market opportunities. Agile manufacturing enterprise continually evolves to adapt itself and to pursue strategic partnerships in order to prosper in an extremely dynamic and exigent economy.

Flexibility is the simplest approach and relates directly to the shop floor. It allows the shop floor to react accordingly in a predefined set of possibilities to meet primary disturbances in production. Feedback from the shop floor comes mainly with the identification of the current lot, which will serve as a basis of decision for the download of the correct production plan.

On the contrary, Adaptability is based on sub-specification; i.e. the system is not completely defined, which allows run-time specification and change according to momentary requirements. Adaptability must be able to understand the disturbances in the shop floor, generating new production plans for the current situation if necessary.

Agility is the uppermost concept and relates to strategic options. Perceiving its business environment, the enterprise has to continuously evolve, adapting internally and pursuing external strategic partnerships that complement its own competencies.

Adaptability also plays an important role, by understanding the business strategies generated by the *agility*. Using these strategies, new production plans or modified ones are added to the production plans database to be used at the shop floor. These new plans may be alternative ways of making current products, or plans for the production of new products.

1.5.2 DFMS Behaviours

A Flexible Distributed Manufacturing System may exhibit several behaviours, the most important, for the scope of our study, are co-operation and co-ordination. However, the goal of any system (not only in manufacturing) is to behave coherently. *Coherence* refers to how well a set of entities behaves as a whole [72]. A coherent system will minimise or avoid conflicting and redundant efforts among entities [63].

A coherent state is achieved by engaging in one or more of the following behaviours:

Co-operation - a process whereby a set of entities develops mutually acceptable plans and executes these plans [75]. These entities explicitly agree to try to achieve a goal (or several goals) with the partial contribution of each participant. The goal needs not to be the same for each participant, but every participant expects to benefit from the co-operation process.

Co-ordination - is the process involving an efficient management of the interdependencies between activities [60].

Competition - is a process whereby several entities independently try to achieve the same goal (with or without the knowledge of the other participants - *explicit* or *implicit* competition).

During the system's natural execution, the majority these behaviours are observed. For instance, in order to compare a DFMS with other distributed systems, consider a distributed problem-solver [38]: it will exhibit both co-operation and co-ordination. In fact, the several solvers co-operate by sharing effort and also co-ordinate their activities' dependencies, e.g. on a finite element analysis [46], each solver operates only on a small set of data, exchanging some data with its neighbours. They co-ordinate their activities so that each one's output is valid, and co-operate by dividing the workload.

Cooperation and Coordination in manufacturing are intuitively proven necessary when adopting a distributed solution [70]: the resources must *cooperate* to manufacture a product and must *coordinate* their actions since the dependencies between operations must be observed; looking at the multi-enterprise level, each company involved establishes plans accepted by the other partners for the fulfilment of the global objectives, thus they *cooperate* to manufacture a product (e.g. providing different assembled parts) and/or to provide some kind of necessary service (e.g. accounting, distribution, quality testing), and must *coordinate* their activities to avoid inefficiency.

1.5.3 Organizational Paradigms

Distributed Manufacturing has been adopted for a long time but its organisational structures were only formalised and proposed as an essential solution in recent years.

In order to achieve the above-mentioned characteristics and behaviours, several organisational paradigms were proposed, namely the Fractal Factory [77, 69], Bionic Manufacturing Systems [64] and Holonic Manufacturing Systems [22, 68], which were comprised in a broader term designated *Open Hierarchical Systems*.

The paradigms can be distinguished by their source of origin, i.e. mathematics for the fractal factory, nature for bionic and genetic production systems, philosophical theory on the creation and evolution of complex adaptive systems for holonic manufacturing.

The *Fractal Factory* is an open system, which consists of independent self-similar units, the fractals, and it's a vital organism due to its dynamic organisational structure. The fractal manufacturing uses the ideas of mathematical chaos: the companies could be composed by small components or fractal objects, which have the capacity to react and adapt quickly to the new environment changes. A fractal object has the following features:

- *self-organised*, which means that does not need external intervention to reorganise itself.
- *self-similar*, which means that one object in a fractal company is similar to other object. In other words, self-similar means that each object contains a set of similar components and shares a set of objectives and visions.
- *self-optimised*, which means that continuously increase its performance.

The explosion of fractal objects into other fractal objects, has the particularity of generating objects which possess organisational structure and objectives similar to the original ones. For Warneke [77] the factory of the future will present different dynamic organisational structure, adopting the project orientation organisation in contrast with the traditional function oriented organisation. This approach implies the organisational structure will encapsulate the process and the technology, therefore forming a cybernetic structure.

The *Bionic Manufacturing Systems* (BMS) have developed under the biology ideas and concepts, and assume that the manufacturing companies can be built upon open, autonomous, co-operative and adaptative entities, which can evolve. The BMS translates to manufacturing systems the structure and organisation behaviour of the living beings, defining a parallelism between biologic systems and manufacturing systems. The cell, organ or living being is modelled in BMS by the *modelon* concept, which is composed by others modelons, forming a hierarchical structure. Each modelon has a set of static properties and behaviours, which can be combined with others, forming distinct entities also designated modelons. The notion of DNA inheritance is translated to manufacturing context by the properties and behaviours that are passed intrinsically to developed modelons [73]. The biological concept *enzymes* and its role in the living beings is modelled in manufacturing systems by entities called supervisors, which are responsible for the regulation and control of the system. Furthermore, the supervisors also play an organisational and structural role in the co-operation process within the BMS, influencing the relations between modelons, imposing self-division or aggregation, in order to adapt and react to the requirements imposed by the environment [71].

The *Holonic Manufacturing System* translates the concepts that Koestler [54] developed for living organisms and social organisations into a set of appropriate concepts for manufacturing industries. Koestler used the word *holon* to describe a basic unit of organisation in living organisms and social organisations, based on Herbert Simon theories and on is observations. Simon observed that complex systems are hierarchical systems formed by intermediate stable forms, which do not exist as auto-sufficient and non-interactive elements but, on the contrary, they are simultaneously a part and a whole. The word holon is the representation of this hybrid nature, is a combination of the Greek word holos, which means whole, and the suffix on, which means particle.

A holon is an autonomous and co-operative entity of a manufacturing system, which include operational features, skills and knowledge, and individual goals. It can represent a physical or logical activity, such as a robot, a machine, an order, a Flexible Manufacturing System, or even a human operator. The holon has information about itself and the environment, containing an information processing part and often a physical processing part. An important feature of HMS is that a holon can be part of another holon, e.g., a holon can be broken into several others holons, which in turn can be broken into further holons, which allows the reduction of the problem complexity.

Note that Those paradigms suggest the idea that manufacturing systems will continue to need a hierarchical structure beside the increased autonomy assigned to individual entities. They also advise the hierarchy needed to guarantee the inter-entities conflict resolution, and maintain the overall system coherence and objectivity resulting from the individual and autonomous attitude of the entities.

1.6 A layered approach to DFMS modeling

In this work, in order to unifying the organisational paradigm concepts exposed in Section 1.5.3, and to studying management issues related to a generalized DFMS, we adapt the layered approach presented in [56].

According to that modeling approach, a DFMS can be seen as an aggregation of four layer, such that each element which belong to a layer at level i represents a layer at level $i + 1$, and is organized as follows.

In the first layer, called *Multi-Enterprise layer*, the interaction between distributed enterprises is modelled. The enterprises act together in order to achieve a common objective, and each enterprise interacts with its suppliers and customers.

A similar environment is found within each manufacturing enterprise. In fact, each enterprise (element) of the first layer is a layer itself at the second level, called the *Enterprise layer*. In this layer the cooperation between geographically distributed entities (normally, the sales offices and/or the production sites) takes place.

Zooming into a production site element shows the *Shop Floor layer*, where the distributed manufacturing control (MC) within a production site or shop floor can be found. In this layer, the entities are distributed work areas working together and

in cooperation, in order to fulfil all orders allocated to the shop floor, respecting the due dates.

A single work area contains a *Cell layer*, where the cooperation between equipments, machines and humans takes place.

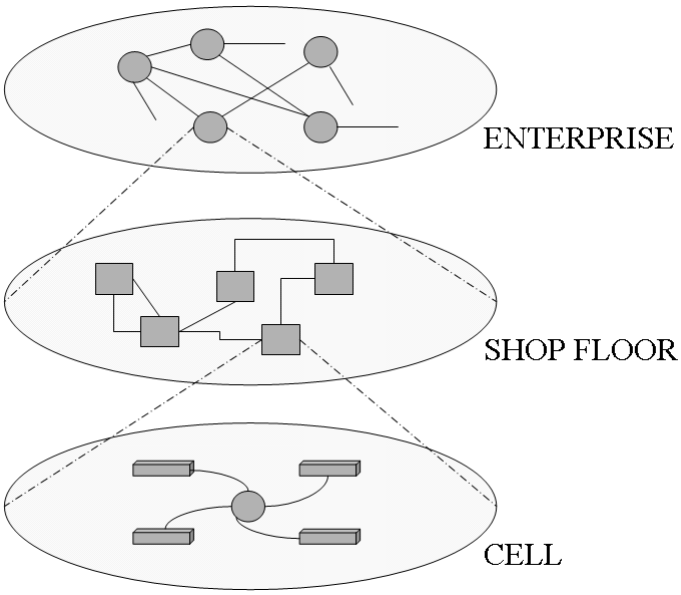

Figure 1.1: Layered DFMS modeling

In Figure 1.1 a 3-layered DFMS is considered, starting from the Enterprise layer.

Collapsing the layers, i.e. exploding the nodes on the top layer in all the elements of the Cell layer, it is easy to see that this approach models a DFMS as a graph where the vertices represent the resources in the system, and links between vertices represent the interaction among the resources. Such graph is called *resource graph*, and its structure will discuss in the next chapter.

In this thesis the work is focused on a generic 3-layered DFMS as depicted in Figure 1.1, in order to study two managerial problems that arise in the distributed manufacturing control function. So, the resource graph we consider contains many vertices as the number of interacting facilities in the system, multiplied to the work areas in each facility, multiplied to the robotic cells in each work area.

1.7 Scheduling issues in DFMS

Considering the high level of automation and cost involved in the development of a DFMS, all development phases of this technology are important in order to achieve the maximum utilization and benefits related to this type of system. However, because the main topic of this thesis is smart scheduling in an DFMS, a vast available scientific literature in related areas, i.e., economical considerations, comparisons between DFMS and standard technologies, design of a DFMS, etc. are only referred to. These main development phases can be identified as (a) strategic analysis and economic justification, (b) facility design to accomplish long term managerial objectives, (c) intermediate range planning, and (d) dynamic operations scheduling.

In multi-stage production systems, scheduling activity takes place after the development of both the master production schedule (MPS) and the material requirements planning (MRP). The goal of these two steps is the translation of product requests (defined as requests for producing a certain amount of products at a specific time) into product orders. A product order is defined as the decision-set that must be accomplished on the shop floor by the different available resources to transform requests into products.

In the scheduling process the following phases can be distinguished as (a) allocation of operations to the available resources, (b) allocation of operations for each resource to the scheduling periods, and (c) job sequencing on each machine, for each scheduling period, considering the job-characteristics, shop floor characteristics, and scheduling goals (due dates, utilization rates, through-put, etc.) In a DFMS, dynamic scheduling, as opposed to advance sequencing of operations, is usually implemented. This approach implies making routing decisions for a part incrementally, i.e., progressively, as the part completes its operations one after another. In other words, the next machine for a part at any stage is chosen only when its current operation is nearly completed. In the same way, a dispatching approach provides a solution for selecting from a queue the job to be transferred to an empty machine.

It has been reported that the sequencing approach is more efficient than the dispatching approach in a static environment; however, a rigorous sequencing approach is not appropriate in a dynamic manufacturing environment, since unanticipated events like small machine breakdowns can at once modify the system status.

The complexity of the scheduling problem arises from a number of factors:

1. Large amount of data, jobs, processes, constraints, etc.
2. Tendency of data to change over time.
3. General uncertainty in such items as process, setup, and transportation times.
4. The system is affected by events difficult to forecast, e.g., breakdowns.
5. The goals of a good production schedule often change and conflict with each other.

Recent trends toward lead time and WIP reduction have increased interest in scheduling methodologies. Such an interest also springs from the need to fully utilize the high productivity and flexibility of expensive DFMSs. Furthermore, the high level of automation supports information intensive solutions for the scheduling problem based both on integrated information systems and on-line monitoring systems. Knowledge of machine status and product advancement is necessary to dynamically elaborate and manage scheduling process.

Chapter 2

Task mapping in distributed manufacturing systems

2.1 Introduction and problem modeling

Process Planning function selects the proper resource types (machine and/or cell) for each operation. If only one resource of this type exists, the routing decision is complete. Otherwise, the problem of deciding which resource will process a given task develops. This problem is known in literature as *Task Mapping Problem* [8].

Task Mapping problem deal with the issue of assigning the planned tasks to the resources of the systems, and is a decision-making activity performed by the Manufacturing Control (MC).

In a generic Distributed and Flexible Manufacturing System each resource can perform each task, so the task mapping problem grows in complexity.

In fact, the resource allocation must take into account the coordination and cooperation needed to guarantee the coherence of the system.

An optimal solution to the task mapping problem must balance the work-load between the resources, in order to reach the cooperation, and must minimize the complexity of communication between the resources, in order to reach the coordination.

2.1.1 Definitions and Graph Partitioning Problem

An undirected graph $G = (V, E)$ is defined as a set of vertices V and a set of edges E. Every edge $e_{ij} \in E$ connects a pair of distinct vertices i and j. The degree d_i of

a vertex i is equal to the number of edges incident to i, and the nodes i' that belong to those edges are the adjacents to i. The neighborhood $N(V')$ of a subset $V' \subseteq V$ of nodes is a set containing all the adjacents to V', and so the set of the adjacents to i is the neighborhood of i (i.e., $|N(i)| = d_i$). Weights and costs can be assigned to the vertices and edges of the graph, respectively. Let w_i and c_{ij} denote the weight of vertex $i \in V$ and the cost of edge $e_{ij} \in E$ respectively.

$\Theta = \{G_1, \ldots, G_K\}$ is a *K-way partition* of G if the following conditions hold:

- each part G_k, $1 \leq k \leq K$, is a subgraph of G induced by a node set V_k, a nonempty subset of V_G;
- parts are pairwise disjoint, i.e. $V_k \cap V_h = \emptyset$ for all $1 \leq k < h \leq K$;
- the union of the vertices in K parts is equal to the node set of G, i.e. $\bigcup_{k=1}^{K} V_k = V_G$.

A K-way partition is also called a *multiway* partition if $K > 2$ and a *bipartition* if $K = 2$. A partition is said to be balanced if each part G_k satisfies the *balance criterion*:

$$W_j \leq W_{avg}(1 + \epsilon), \quad \text{for } j = 1, 2, \ldots, K$$

where the weight W_j of a part G_j is defined as the sum of the weights of the vertices in that part (i.e. $W_j = \sum_{i \in V_j} w_i$), $W_{avg} = (\sum_{i \in V_G} w_i)/K$ denotes the weight of each part under the perfect load balance condition, and ϵ represents the predetermined maximum imbalance ratio allowed.

In a partition Θ of G an edge is said to be *cut* if its pair of vertices belong to two different parts, and *uncut* otherwise. The cut and uncut edges are also referred to here as external and internal edges, respectively. The set of external edges of a partition Θ is denoted as E_Θ. The *cutsize* definition for representing the cost $c(\Theta)$ of a partition Θ is:

$$c(\Theta) = \sum_{e_{ij} \in E_\Theta} c_{ij}$$

where each cut edge e_{ij} contributes its cost c_{ij} to the cutsize. Hence, the graph partitioning problem can be defined as the task of dividing a graph into two or more parts such that the cutsize is minimized, while the balance criterion on part weights

is maintained. The graph partitioning problem is known to be NP-hard even for bipartitioning unweighted graphs [28].

A *mapping* is a function $\pi : V_G \rightarrow V_R$ that, given node sets V_G and V_R and a fixed balance criterion, returns a partition of V_G in $|V_R|$ parts and a bijective assignment of each part $V_k \subseteq V_G$ to the correspondent node $k \in V_R$. So the cost $c(\pi)$ associated to a mapping π is a function of the cost of the partition, i.e. $c(\pi) = f(c(\Theta))$.

2.1.2 Standard task mapping models

The problem of mapping tasks on a given set of resources can be modeled in its generalized form with two graphs, one modeling the tasks and their interactions, and one modeling the resources (facilities) in the system. So the problem takes as input a *task graph* and a *resource graph* and outputs the mapping from tasks to resources.

A *task graph* $G = (V_G, E_G)$ models the information needed to coordinate the processing of tasks through the distributed manufacturing system. A task in the system continuously needs information from adjacent tasks, and sends information about its state to other tasks. A task graph is a weighted graph. A node $i \in V_G$ represents a task and an edge $e_{ii'} \in E_G$ a continuous communication (stream) between a pair of tasks. The weight of a node represents the task's processing time and that of an edge the communication requirement of the connected tasks (the amount of data that must be communicated for these tasks).

In this thesis we consider quasi-homogeneous tasks. So, the weight of the tasks may be scaled, and we assume that each node has a weight equal to one.

The continuous stream of data between tasks can be also be scaled and discretized through an interval time. So, without loss of generality, we can suppose that a each task sends one information in the unit time.

A *resource graph* $R = (V_R, E_R)$ models the resources in the system and the interactions among them. A node $j \in V_R$ of a resource graph represents a facility and an edge $e_{jj'} \in E_R$ a link between a pair of facilities. It is also a weighted graph (both nodes and edges are weighted). The weight of a node represents the facility's production capacity (the amount of work that can be performed) and that of an edge the link's communication capacity (the amount of data that can go through the link in a unit time).

According to the DFMS layered modeling approach discussed in Chapter 1, we

can construct the resource graph by exploding each node of the Enterprise layer.

In order to study a generic DFMS, through this work we assume that the resource graph is *complete*, namely it is a *clique*, which means that all the resources are able to communicate with each other.

Moreover, we assume that the capacity of the edges is equal to infinite.

Note that a task graph is not a traditional dependence graph, in which an edge $i \rightarrow i'$ represents the fact that task i' can start only after i has finished. Rather, our task graph models a data processing system, in which all tasks continuously receive and send pieces of data.

If too many tasks are assigned on a facility or too much communication goes through a link, the facility or the link becomes a bottleneck and determines the overall throughput of the entire system.

The key to achieving a good throughput is *clustering* of a task graph, a process which recognizes highly-connected components in a task graph. A cluster in a task graph represents a set of tasks that are intensively communicating with each other.

Unlike other formulations, our model does not have an explicit notion of *parallelized tasks*. That is, a single node of a task graph can be mapped only on a single node of a resource graph. A parallelized task can be to some extent modeled by many nodes that together represent a single logical task.

Once we have a graph model, *graph partitioning* can be used to determine how to divide up the work and data for an efficient parallel processing. Our objectives, stated loosely, are to evenly distribute the work over K resources by partitioning the vertices into K equally weighted sets while minimizing communication between resources which is represented by edge crossing between partitions.

Until recently only the standard graph partitioning approach has been employed. The standard approach is to model the problem using a graph as described above and partition the vertices of the graph into equally weighted sets in such a way that the weight of the edges crossing between sets is minimized.

Hence, by setting $c_{ii'} = 2$ for each edge $e_{ii'} \in E_G$ the mapping of G on R reduces to the K-way partitioning of G according to the cutsize definition given in Section 2.1.1. Thus, minimizing the cutsize is an effort towards minimizing the total volume of communication between the resources. Maintaining the balance criterion corresponds to maintaining the work load balance during local processing.

Unfortunately, the standard graph partitioning approach has significant short-

comings which are discussed in detail in the next section. The edge cut metric that it tries to minimize is, at best, an imperfect model of communication in a distributed manufacturing system. The model also suffers from a lack of expressibility which limits the applications it can address.

2.2 Our contributions

In this chapter we present standard graph partitioning models of the task mapping techniques, show how the standard methodology for graph partitioning minimizes the wrong metric and lacks expressibility. A key contribution of our work is the development of some theoretical results that brings to a new model, for which we provide some insights on the formulation and bounds on the optimal value of the objective function that allow to provide high quality partitions quite consistently.

2.3 Graph Partitioning deficiencies

In this section we discuss several shortcomings of the standard graph partitioning approach. We begin with flaws associated with using the edge cuts metric, and continue with limitations of the standard graph model.

2.3.1 Flaws of the Edge Cut metric

Minimizing edge cuts has several major flaws. First of all, although it is not acknowledged, edge cuts are not proportional to the total communication volume. The scenario is illustrated in Figure 2.1

The ovals correspond to different resources among which the vertices of the graph are partitioned. Assume that each edge has a weight of two corresponding to one unit of data being communicated in each direction. So the weight of the cut edges is ten. However, observe that the data from node v_2 on resource R_1 need only be communicated once to resource R_2; similarly with nodes v_4 and v_7. Thus, the actual communication volume is only seven. In general, the edge cut metric does not recognize that two or more edges may be representing the same information flow, so it over counts the true volume of communication.

Secondly, the performance of the system is generally limited by the slowest resource. Even if the work is well balanced, the coordination (communication) effort

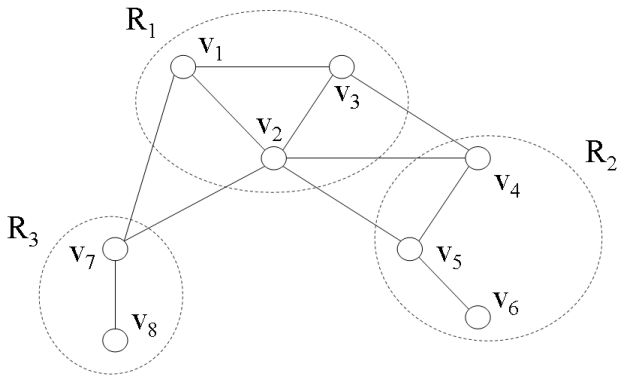

Figure 2.1: Flaws of the edge cut metric

might not be. So, rather than minimizing the total communication volume, we may instead wish to minimize the maximum volume handled by any single resource. The standard edge cuts does not encapsulate this type of objective.

Lastly, many resources may be geographically distributed, and the effectiveness in coordination can be reached only if communication is preferable restricted to facilities which are near each other. So, for the problem illustrated in Figure 2.1, on a one-dimensional row of resources, the layout $R_3 \leftrightarrow R_1 \leftrightarrow R_2$ would be preferable to $R_1 \leftrightarrow R_2 \leftrightarrow R_3$.

2.3.2 Limitations of the Standard graph model

Besides minimizing the wrong objective function the standard graph partitioning approach suffers from limitations due to the lack of expressibility in the model.

Major limitations of the undirected graph model is that it can only express symmetric data dependencies. For example a node v_i might need informations from a node $v_{i'}$, while $v_{i'}$ does not need v_i. This situation can be easily represented in a *directed* graph, but not in the standard model.

In a directed graph, edges are directed from the data producing vertex to the data consuming vertex. There are two work-arounds to take the unsymmetric dependencies with the standard model. The first is to convert the directed edges to undirected edges. The second is a slight extension of the first; an edge that represents only a one-way communication gets a weight of one, and an edge that represents two-way communication gets a weight of two.

2.4 Graph theoretical results

Given a K-partition of the task graph G in $G_1, \ldots, G_K$ subsets and a mapping of the partitions on the resource graph R, we can define for each node $i \in G$, which belong to a generic partition G_j mapped to a node $j \in R$, the *external neighborhood* $N_{ext}(i)$ of i as the vertices adjacent to i not mapped in the same resource where i is mapped:

$$N_{ext}(i) = \{v \in N(i) : i \in G_j, v \notin G_j, j \in V_R\}$$

In the same way, we can define the *internal neighborhood* of i as the set of the adjacents to i which belong to the same partition to which i is mapped:

$$N_{int}(i) = \{v \in N(i) : i, v \in G_j, j \in V_R\} = N(i) \setminus N_{ext}(i)$$

In graph theoretical view, the graph partitioning model of the task mapping problem treat all cut edges of equal cost in an identical manner while computing the cutsize.

However, m cut edges, each of cost 2, stemming from a vertex i in par G_j to m vertices $v_1, \ldots, v_m$ in part $G_{j'}$ incur only $m + 1$ communications instead of $2m$ (see Figure 2.2).

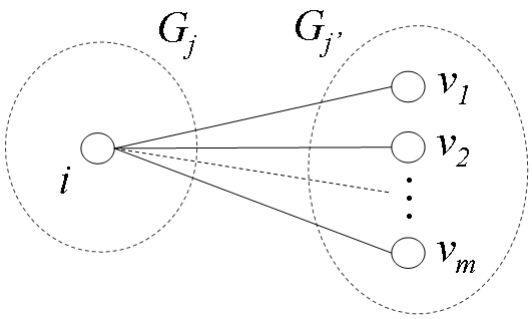

Figure 2.2: Communication stream between a node i in a partition G_j and its external neighborhood in a partition $G_{j'}$

Denote as $N_R(i)$ the set of partitions which contain the external neighborhood of i, i.e. the set of external resources which are adjacents to i:

$$N_R(i) = \{j \in V_R : i \notin G_j, N_{ext}(i) \cap G_j \neq \emptyset\}$$

Denote as $N_G(j)$ the set of external vertices to which a partition G_j is adjacent:

$$N_G(j) = \{i \in V_G : i \notin G_j, N_{ext}(i) \cap G_j \neq \emptyset\}$$

Starting from the above considerations on the effective communication stream the Lemma 1 follows:

Lemma 1 *Given a task graph $G = (V_G, E_G)$, a resource graph $R = (V_R, E_R)$ with K resources (i.e., $|V_R| = K$) and a mapping function π of G on R partitioning G in K subsets $G_1, \ldots, G_K$, each associated with a node of R, then the effective communication volume in which a given node $i \in G$ incurs is:*

$$c(i) = |N_{ext}(i)| + |N_R(i)|$$

Proof: It is easy to see that the resource to which i is mapped sends data about the task i to each of the resources i is adjacent to (i.e., sends $|N_R(i)|$ data), and receive exactly $|N_{ext}(i)|$ data in order to achieve the coordination between task i and all the tasks that are adjacent to i. □

In Figure 2.3 the example illustrated in Figure 2.2 is drawn underlining the effective communication edges that link a node i with its external adjacents in the system.

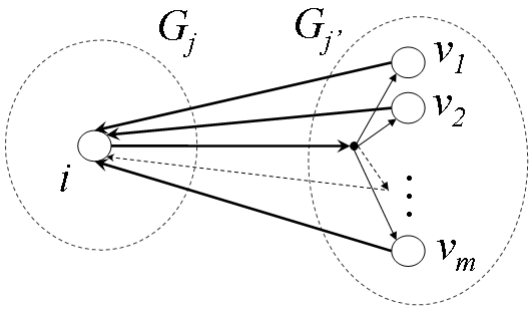

Figure 2.3: Effective communication flow *from* and *to* a node i

So, the evaluation of an effective cut metric can be stated through the following theorem.

Theorem 1 *Given a task graph $G = (V_G, E_G)$, a resource graph $R = (V_R, E_R)$ with K resources (i.e., $|V_R| = K$) and a mapping function π of G on R partitioning G*

in K subsets $G_1, \ldots, G_K$, each associated with a node of R, the following assertions are equivalent:

1. *communication cost associated with π is: $c(\pi) = \sum_{i \in V_G} |N_R(i)|$*
2. *communication cost associated with π is: $c(\pi) = \sum_{j \in V_R} |N_G(j)|$*

Proof: In fact, given a K-partition of G, by Lemma 1 each task $i \in V_G$ sends one information to each of the resources it is adjacent to (each resource j such that $N_{ext}(i) \cap G_j \neq \emptyset$), and needs exactly $N_{ext}(i)$ informations. So the amount of data exchanged is equal either to the sum of informations send from each partition, which is equal to the sum of the partition adjacent to each node in a given partition for all the partitions in the system (*1.*), or to the sum of informations received by each partition, which is equal to the sum of the nodes adjacent to each partition for all the partitions in the system (*2.*). □

Theorem 1 shows which is the right cut metric to evaluate a mapping of a set of tasks on a set of resources. To really understand the relation between a mapping solution with the right metric and that with the edge cut metric, the following results are given.

Remind from Section 2.1.1 that the cost associated to an optimal partition Θ found solving the graph partitioning problem is $c(\Theta) = \sum_{e_{ij} \in E_\Theta} c_{ij}$, and that for our problem $c_{ij} = 2$, so: $c(\Theta) = 2\,|E_\Theta|$.

First, we deal with the bipartition ($K = 2$) case in the following theorems.

Lemma 2 *Given a graph $G = (V_G, E_G)$, and a bipartition Θ of G in $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, let $B_\Theta = (B_1 \subseteq V_1, B_2 \subseteq V_2, E_\Theta)$ be the subgraph of G induced the external edges E_Θ.*

Then:

$$|E_\Theta| \geq \max\{|B_1|, |B_2|\}$$

Proof: B_Θ is a subgraph induced by a set of edges, so it does not contains isolated vertices. Since B_Θ is a bipartite graph without singletons, there are at least many edges as the maximum cardinality of the node partitions B_1 and B_2. □

Theorem 2 *Given a graph $G = (V_G, E_G)$, let Θ' be the bipartition of G in $G'_1 = (V'_1, E'_1)$ and $G'_2 = (V'_2, E'_2)$ obtained by solving at the optimum the graph bipartitioning problem on G, let Θ'' be the bipartition of G in $G''_1 = (V''_1, E''_1)$ and $G''_2 = (V''_2, E''_2)$ obtained by solving at the optimum the bipartitioning problem on G with the right cut metric defined in Theorem 1, and let $B_{\Theta'} = (B'_1, B'_2, E_{\Theta'})$ and $B_{\Theta''} = (B''_1, B''_2, E_{\Theta''})$ the bipartite subgraphs induced by Θ' and Θ'' respectively.*

Then:

$$c(\Theta'') \leq c(\Theta')$$

Proof: Note that:

- $|E_{\Theta'}| \leq |E_{\Theta''}|$, because Θ' is the optimal solution of a graph partitioning problem (see Section 2.1.1);
- $|B''_1| + |B''_2| \leq |B'_1| + |B'_2|$ by Theorem 1;
- $|E_{\Theta'}| \geq \max\{|B'_1|, |B'_2|\}$ by Lemma 2.

Thus, the following inequalities hold:

$$\begin{aligned} c(\Theta'') &= |B''_1| + |B''_2| \leq |B'_1| + |B'_2| \leq \\ &\leq 2\max\{|B'_1|, |B'_2|\} \leq \\ &\leq 2|E_{\Theta'}| = c(\Theta') \end{aligned}$$

□

The results may be generalized to the K-partition case, as follows.

Theorem 3 *Given a graph $G = (V_G, E_G)$, let Θ' be the K-partition of G obtained by solving at the optimum the graph partitioning problem on G, let Θ'' be the K-partition of G obtained by solving at the optimum partitioning problem on G with the right cut metric defined in Theorem 1.*

Then:

$$c(\Theta'') \leq c(\Theta')$$

Proof: Coupling together Theorem 1 and Theorem 2 it is easy to verify that the cost of the solution obtained with a graph partitioning technique corresponds to

consider each node i that belongs to a cut edge (see Section 2.1.1 for the definition of cut edge) as adjacent to exactly $N_{ext}(i)$ partitions (i.e. $|N_{ext}(i)| = |N_R(i)|$), while a solution using the cut metric proposed in Theorem 1 tends to minimize this adjacency. □

Finally, to manage with the possibility of unsymmetric data exchanges, we can generalize the result from Theorem 1 for a directed graph.

Given a directed task graph $G = (V_G, A_G)$, where an arc $a_{ii'} \in A_G$ represents an oriented edge from i to i', and a mapping function of G on a resource graph R, denote as $N_R^+(i)$ the set of resources to which a node i sends data (i.e. $N_R^+(i) = \{j \in V_R : i \notin G_j, N_{ext}(i) \cap G_j \neq \emptyset, \exists i' \in N_{ext}(i)\text{such that } a_{ii'} \in A_G\}$), and as $N_G^-(j)$ the set of nodes from which a resource j receives data (i.e. $N_G^-(j) = \{i \in V_G : i \notin G_j, N_{ext}(i) \cap G_j \neq \emptyset, \exists i' \in G_j\text{such that } a_{ii'} \in A_G\}$).

Theorem 4 *Given a directed task graph $G = (V_G, A_G)$ and a mapping π of G on K resources, the following assertions are equivalent:*

1. *communication cost associated with π is: $c(\pi) = \sum_{i \in V_G} \left|N_R^+(i)\right|$*
2. *communication cost associated with π is: $c(\pi) = \sum_{j \in V_R} \left|N_G^-(j)\right|$*

Proof: Directly by the Theorem 1. □

2.5 Hypergraph Partitioning approach

A *hypergraph* is a generalization of a graph in which edges can include more than two vertices. A hypergraph, $H = (V_H, E_H)$, consists of a set of vertices V_H and a set of hyperedges E_H. Each hyperedge $h_i$comprises a subset of vertices, i.e. $h_i \subseteq V_H$,. The size of a hyperedge is equal to the number of vertices it contains. The set of hyperedges connected to a vertex $i \in V_H$ is denoted as $nets(i)$, and the degree of a vertex is equal to the number of hyperedges it is connected to, i.e. $d_i = |nets(i)|$. Note that graphs are special cases of hypergraphs in which each hyperedge only contains two vertices. Similar to graphs, let w_i and c_j denote the weight of vertex $i \in V_H$ and the cost of hyperedge $h_i \in E_H$, respectively.

Definition of K-way partition of hypergraphs is identical to that of graphs. In a partition Θ of H a hyperedge that has at least one vertex in a part is said to *connect*

that part. *Connectivity set* Λ_i of a hyperedge h_i is defined as the set of parts connected by h_i. *Connectivity* $\lambda_i = |\Lambda_i|$ denotes the number of parts connected by h_i. A hyperedge h_i is said to be *cut* if it connects more than one part (i.e. $\lambda_i > 1$), and *uncut* otherwise (i.e. $\lambda_i = 1$). The *cutsize* definition for representing the cost $c(\Theta)$ of a partition Θ is equal to the sum of the costs of the cut hyperedges:

$$c(\Theta) = \sum_{h_i:\lambda_i>1} c_i$$

Hence, the hypergraph partitioning problem [57] can be defined as the task of dividing a hypergraph into two or more parts such that the cutsize is minimized, while a given balance criterion among the part weights is maintained. Here, part weight definition is identical to that of the graph model. The hypergraph partitioning problem is known to be NP-hard [57].

For our purposes, hyperedges allow an alternative representation of data dependencies. The partitioning problem is now to divide the vertices into equally weighted sets so that few hyperedges cross between partitions.

Starting from the task graph $G = (V_G, E_G)$ we can construct the *equivalent task hypergraph* $H = (V_H, E_H)$ with $|V_H| = |V_G|$ = hyperedges. Each vertex $i \in G$ corresponds to a hyperedge h_i consisting of i and its neighborhood $N(i)$.

By assigning unit costs to the hyperedges (i.e. $c_i = 1$), the cost of a mapping function of vertices V_G on K resources is measured by the following result.

Theorem 5 *Given a task graph* $G = (V_G, E_G)$*, its* equivalent task hypergraph $H = (V_H, E_H)$*, a resource graph* $R = (V_R, E_R)$ *with* K *resources (i.e.,* $|V_R| = K$*) and a mapping function* π *of* G *(*H*) on* R *partitioning* G *(*H*) in* K *subsets* $G_1, \ldots, G_K$*, each associated with a node of* R*, then the system communication cost is:*

$$c(\pi) = \sum_{i \in V_G} |N_R(i)| = \sum_{j \in V_R} |N_G(j)| = \sum_{h_i:\lambda_i>1} c_i \, (\lambda_i - 1)$$

Proof: A hyperedge reflects all of the tasks that either send or receive an information. When the vertices are partitioned and mapped on the resources, that information must be communicated from the resource which process the task to all those that need such a information. Thus, the communication associated with a hyperedge is one less than the number of resources its constituent vertices are partitioned among, which corresponds to the cost of a mapping function as it is described in Theorem 1. □

So by partitioning the hypergraph in such a way that hyperedges are split among as few resources as possible, the model correctly minimizes communication volume.

Note that in addition to resolving the principle problem of the edge cut metric, the hypergraph approach is more expressive than the standard model, since it can encode problems with unsymmetric dependencies. In fact, the guiding principle in the construction of a hypergraph is that each hyperedge contains the set of vertices which generate or need some data. This principle applies equally well to the case when dependencies are uni-directional, and it continues to correctly model the communication volume. However, motivated by Theorem 4, there is a subtle requirement that the data is produced by one of the vertices that depends on it.

Since hypergraph partitioning is a NP-hard problem, task mapping problem is also believed as difficult as hypergraph partitioning, although the computational complexity of the latter still remains an open question.

The hypergraph partitioning version of the task mapping may be solved by heuristics that approximate the optimal solution.

Kernighan-Lin (KL) based heuristics, for example, are widely used for graph/hypergraph partitioning because of their short run-times and good quality results. The KL algorithm is an iterative improvement heuristic originally proposed for graph bipartitioning [50]. The KL algorithm, starting from an initial bipartition, performs a number of passes until it finds a locally minimum partition. Each pass consists of a sequence of vertex swaps. The same swap strategy was applied to the hypergraph bipartitioning problem by Kernighan and Schweikert [51]. Fiduccia-Mattheyses (FM) [26] introduced a faster implementation of the KL algorithm for hypergraph partitioning. They proposed vertex move concept instead of vertex swap. This modification, as well as proper data structures, e.g., bucket lists, reduced the time complexity of a single pass of the KL algorithm to linear in the size of the graph and the hypergraph. Here, *size* refers to the number of edges in graph and to the size of all the hyperedges in hypergraph.

The performance of the FM algorithm deteriorates for large and very sparse graphs/hypergraphs. Here, sparsity of graphs and hypergraphs refer to their average vertex degrees. Furthermore, the solution quality of FM is not *stable* (*predictable*), i.e., average FM solution is significantly worse than the best FM solution, which is a common weakness of the move-based iterative improvement approaches.

Random multi-start approach is used in VLSI layout design to alleviate this

problem by running the FM algorithm many times starting from random initial partitions to return the best solution found [9]. However this approach brings a lost in efficiency.

These considerations have motivated the *two-phase* application of the move-based algorithms in hypergraph partitioning [31]. In this approach, a clustering is performed on the original hypergraph H_0 to induce a coarser hypergraph H_1. Clustering corresponds to coalescing highly interacting vertices to *supernodes* as a preprocessing to FM. Then, FM is run on H_1 to find a bipartition Θ_1, and this bipartition is projected back to a bipartition Θ_0 of H_0. Finally, FM is re-run on H_0 using Θ_0 as an initial solution. Recently, the two-phase approach has been extended to *multilevel* approaches [15, 37, 45] leading to successful graph partitioning tools Chaco [39], MeTiS [47] and hMeTiS [48]. These multilevel heuristics consist of 3 phases: *coarsening*, *initial partitioning* and *uncoarsening*. In the first phase, a multilevel clustering is applied starting from the original graph by adopting various matching heuristics until the number of vertices in the coarsened graph reduces below a predetermined threshold value. In the second phase, the coarsest graph is partitioned using various heuristics including FM. In the third phase, the partition found in the second phase is successively projected back towards the original graph by refining the projected partitions on the intermediate level uncoarsened graphs using various heuristics including FM.

In the rest of the chapter we give some insights on the solution of the problem, using theoretical results on the structure of the problem formulation. In fact, although the multilevel partitioning scheme is the best known solution method for the hypergraph partitioning problem, it has some drawbacks, especially in the graph coarsening phase.

In fact, the coarsening phase is based on the solution of a matching problem, which try to identify nodes that share common data, but there is not a guarantee that such nodes should be placed in the same partition in a optimal mapping solution. So, a refinement on the solution is needed, and it can be very computational-expensive.

Moreover, a matching-based clustering allows the clustering of only pairs of vertices in a level. So, how to properly coarsen a hypergraph is still an open problem [45].

Lastly, the computational effort depends on how the starting feasible solution is close to the optimal solution.

2.6 A Mixed Integer Linear Programming formulation

In this section we present a new formal definition of the task communicating mapping problem in a distributive manufacturing environment, different from the standard one, and ralated to the hypergraph partitioning approach discussed in the previous Section.

Let n be the number of tasks, and K denote the number of resources in the system, as above defined.

Let ϵ represents the maximum imbalance ratio allowed between the cardinality of the partitions, and consider the following decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if } N_{ext}(i) \cap G_j \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if } i \in G_j \\ 0 & \text{otherwise} \end{cases}$$

According to the problem definitions given in Section 2.1 and to the graph theoretical results presented in Section 2.4, the problem can be formulated as a 0-1 mixed integer linear programming (MILP) problem as follows:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{K} x_{ij} \\
\text{s.t.} \quad & |\sum_{i=1}^{n} y_{ij} - \sum_{i=1}^{n} y_{ij'}| \leq \epsilon, \quad \forall j, j' = 1 \ldots K, \quad j' \neq j && (2.1)\\
& \sum_{j=1}^{K} y_{ij} = 1, \quad \forall i = 1 \ldots n && (2.2)\\
& x_{ij} \leq \sum_{i' \in N(i)} y_{i'j}, \quad \forall i = 1 \ldots n, \forall j = 1 \ldots K && (2.3)\\
& x_{ij} \leq 1 - y_{ij}, \quad \forall i = 1 \ldots n, \forall j = 1 \ldots K && (2.4)\\
& x_{ij} \geq y_{i'j} - y_{ij}, \quad \forall i = 1 \ldots n, \forall i' \in N(i), \forall j = 1 \ldots K && (2.5)\\
& x_{ij} \in \{0, 1\}, \quad \forall i = 1 \ldots n, \forall j = 1 \ldots K && (2.6)\\
& y_{ij} \in \{0, 1\}, \quad \forall i = 1 \ldots n, \forall j = 1 \ldots K && (2.7)
\end{aligned}$$

According to Theorem 1 the objective function tends to minimize the sum over each node of the partitions (i.e., resources) to which the nodes are adjacent, or the

sum over each partition of the nodes (i.e., tasks) to which the partitions are adjacent. In fact, it is easy to see that: $\sum_{i=1}^{n}(\sum_{j=1}^{K} x_{ij}) = \sum_{i \in V_G} |N_R(i)|$ and $\sum_{j=1}^{K}(\sum_{i=1}^{n} x_{ij}) = \sum_{j \in V_R} |N_G(j)|$.

The group constraints (2.1) ensure a balance criterion between the cardinality of the partitions, measured by ϵ.

The group constraints (2.2) ensure the assignment of each task to exactly one partition.

The group constraints (2.3), (2.4) and (2.5) relates the assignment of the nodes to the partitions with the cost of such mapping. In particular: (2.3) ensures that, given a node $i \in V_G$ and a partition $j \in V_R$, if i has no adjacents in j, then the communication cost between i and j is null, either i is assigned to j (in which case $y_{ij} = 1$) or it is not ($y_{ij} = 0$); (2.4) ensures that if a node i is assigned to a partition j, the communication cost between i and j is null; (2.5) ensures that, given a node i not assigned to a partition j, if at least one of its adjacents, say i', is assigned to j, then there is a positive communication cost between i and j.

Constraints (2.6) are limitations on the communications cost for our problem.

Constraints (2.7) define binary variables y_{ij}.

2.7 An improved formulation

The problem presented in the previous section is no doubt general (i.e. it is independent of the problem instance), however, it has the following shortcomings:

- it is large in dimension, i.e. it has the largest number of variables and constraints that are possible for the given problem dimension;
- A large number of alternative optima are possible. It is easy to observe that an alternative optima can be obtained by switching the assignments of all tasks of a particular resource with another resource. Since the problem is probably NP-hard, these alternatives make it further difficult to prove the optimality of a solution even when an optimum one is found. It is generally accepted that when using a search method to solve an NP-hard problem, even if a small amount of time is expended in the *search* part of the algorithm (to encounter a good or to-be-proven optimal solution), exponential time is required in the *prove* part of the algorithm.

With an objective to overcome all these shortcomings, in this section we present an improved formulation that follows the same basic structure as the general formulation but capitalizes on problem data specific features.

First, note that for our problem $\epsilon = 1$, since we want a strict balanced partition of the tasks. So, without loss of generality, we can assign to each partition j a fixed cardinality, which is equal to the production capacity of the resource it represents, so equal to the weight C_j of the node j in the resource graph. The first group constraints can be stated as:

$$\sum_{i=1}^{n} y_{ij} = C_j, \quad \forall j = 1 \dots K$$

Moreover, note that the group constraints (2.3) and (2.4) are redundant, since the objective function tends to minimize the sum of x_{ij}, trying to take null the values of these variables, unless there is a group constraints (2.5) that forces some x_{ij} to be equal to 1.

Starting from the above considerations, integer and binary constraints (2.6) can also be relaxed, since every x_{ij} tends to be null, or is at most equal to 1.

Lastly, note that the binary group constraints (2.7) holds even if y_{ij} is a positive integer, since the group (2.2) forces the sum of those variables be equal to 1.

So an improved formulation of the problem is:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j=1}^{K} x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n} y_{ij} = C_j, \quad \forall j = 1 \dots K && (2.8) \\
& \sum_{j=1}^{K} y_{ij} = 1, \quad \forall i = 1 \dots n && (2.9) \\
& x_{ij} \geq y_{i'j} - y_{ij}, \quad \forall i = 1 \dots n, \forall i' \in N(i), \forall j = 1 \dots K && (2.10) \\
& x_{ij} \geq 0, \quad \forall i = 1 \dots n, \forall j = 1 \dots K && (2.11) \\
& y_{ij} \in Z^{+}, \quad \forall i = 1 \dots n, \forall j = 1 \dots K && (2.12)
\end{aligned}$$

It is emphasized that this formulation is not an approximation, instead it capitalizes on the problem data and on the redundancies to reduce the MILP dimension and speed up solution times.

2.8 Towards the solution in the bisection case

The problem formulation given in the previous section is believed to be NP-hard, although its complexity is still not known.

This motivate the research of *good* bounds on the optimal solution even in the case of bisection (i.e. bipartitioning). Note that the hypergraph K-partitioning problem is NP-hard, and remains NP-hard even for $K = 2$ [57].

Writing the problem formulation for $K = 2$, we can use the binary concept associated with the *assignment* variables y_{ij} for further reduce the size of the problem.

In fact, without loss of generality, we can specify an ordering on the partitions of G, say G_1 and G_2, and associate to a node $i \in G$ a binary variable y_i as follows:

$$y_i = \begin{cases} 1 & \text{if } i \in G_1 \\ 0 & \text{if } i \in G_2 \end{cases}$$

So the problem formulation may be stated:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} y_i = C && (2.13) \\
& x_i \geq y_{i'} - y_i, \quad \forall i = 1 \ldots n, \forall i' \in N(i) && (2.14) \\
& x_i \geq y_i - y_{i'}, \quad \forall i = 1 \ldots n, \forall i' \in N(i) && (2.15) \\
& x_i \geq 0, \quad \forall i = 1 \ldots n && (2.16) \\
& y_i \in \{0, 1\}, \quad \forall i = 1 \ldots n && (2.17)
\end{aligned}$$

Note that it suffice to specify a single capacity limit C (constraints (2.13)) for the partition G_1, since automatically the capacity limit for G_2 is equal to $n - C$.

Constraints (2.9) and (2.12) are grouped together in the group constraints (2.17).

Constraints (2.10) must be duplicated in (2.14) and (2.15) to take into account the binary meaning of y_i.

The given problem formulation is relevant for our purpose, since a constraint-relaxation of it allows the determination of *good* bounds on the optimal solution.

In fact, removing the group constraints (2.15), the formulation refers to a problem which solution is a size-constrained minimum-separator set of vertices, i.e. we want to find a fixed-size (C) subgraph of G which can be disconnected from G by deleting the minimum set of vertices.

This problem is easier to solve than the hypergraph bipartitioning one, and it is easy to see that, given an optimal partition Θ^*, called $|n_1^*|$ and $|n_2^*|$ the size of the minimum separator set n_1^* and the size of the adjacents to the latter in G_1 respectively (i.e., n_1^* contains the nodes i such that $x_i = 1$ in the optimal solution, n_2^* contains the nodes i such that $y_i = 1$ and some of the adjacents to i belongs to n_1^*), then the cost of the optimal solution of the hypergraph bipartitioning problem, say Θ, is bounded as follows:

$$2\,|n_1^*| \leq c(\Theta) \leq |n_1^*| + |n_2^*|$$

Moreover, using the separator set n_1^* we can construct, if some conditions hold, an *efficient coarsening/solution scheme*, as follows:

1. Find all the connected components in the bipartite graph $G^* = (n_1^*, n_2^*, E_{\Theta^*})$ and apply what follows to each one of them (for now we can suppose, without loss of generality that G^* is connected);

2. Find in n_1^* a subset n_k^* of vertices such that:

$$|N_{ext}(n_k^*)| - |N_{ext}(n_1^* \setminus n_k^*) \cap N_{ext}(n_k^*)| \geq |n_k^*|\,;$$

3. If $n_k^* \equiv n_1^*$, then we can coarsen n_1^* in a *supernode* and compute a new solution to the problem, because there is not a proper subset of n_1^* that can be moved to another partition in order to improve the solution;

4. If $n_k^* \subset n_1^*$ then:

   - coarsen n_k^* in a super node and $n_1^* \setminus n_k^*$ in another supernode
   - try to improve the solution moving n_k^* with the FM algorithm, because if n_k^* and $n_1^* \setminus n_k^*$ are assigned to different partitions in the optimal solution, then one of them is on the *boundary* of the solution (i.e. nodes i such that $x_i = 1$), since G^* is connected
   - if no improvement is allowed, then coarsen n_k^* with $n_1^* \setminus n_k^*$

This scheme can be enclosed in an algorithm in order to reduce the size of the initial graph.

Note that the scheme is applied on the vertices of the task graph, not on the hyperedges or nodes of the equivalent task hypergraph (as standard coarsening techniques).

Moreover, the proposed method coarsen nodes that should share a partition in the optimal solution, so does not need any refinements in the uncoarsening phase.

Chapter 3

Task scheduling in flexible manufacturing cells

3.1 Introduction

A substantial amount of recent research has been directed towards the development of industrial robots. The bulk of this work has dealt with electromechanical capabilities, sensing devices and computers controls. Relatively little research has investigated the operational problem associated with application of this technology ([5], [6], [14], [34], [49]).

We investigate one operational problem which is encountered in applications, in which a robot is used to tend a number of machines. Such an application would arise, for example, when machines have been organized into a machine cell to implement the concept of group technology ([1], [2], [3], [4], [8], [76]). The cell would be used to produce multiple set of parts at prespecified production rates. The feasibility of assigning one robot to perform the tasks necessary for tending all the machines, so that parts are produced at specified production rates, is an important operational problem. In fact, the resolution of this problem determines the number of robots that might be necessary to tend machines in a manufacturing system and hence the investment required to robotize tending activities.

These kinds of problems were first introduced by Asfahl ([7]). A summary of the related literature can be found in [13] and in [33]. In particular, in [13] the authors find the optimal sequence of moves for a two machine robot-centered cell producing a single part-type, and solve the part sequencing problem for a given

one-unit robot move cycle in a two-machine cell producing multiple part-types. Hall et al. [33] showed that the optimal solution to the multiple part-types problem in a two machine cell is not generally given by a one-unit cycle. They develop an $O(n^4)$, where n is the number of parts, time algorithm that jointly optimizes the robot move cycle and part sequencing problems. The latter algorithm was improved later by Aneja and Kamoun [6] with one of $O(n \log n)$ complexity.

Blazewicz et al. ([13]) provided a summary of the related literature and described a line for machining castings for truck differential assemblies in the form of a three-machine robotic cell where a robot has to transfer heavy mechanical parts between large machines.

Descriptions of interesting applications were provided, e.g. by Hartley ([36]). Complexity issues were discussed by Wilhelm ([78]), and by Crama and Van De Klundert ([21]), that provided a proof that a basic version of the problem is strongly NP-Complete, and described a polynomial time algorithm for minimizing cycle time over all one-unit cycles in an m machine cell producing a single part-type in a robot-centered cell.

Hall et al. in [34] considered a three-machine cell producing multiple part-types, and prove that, in two out of the six potentially optimal robot move cycles for producing one unit, the recognition version of the part sequencing problem is unary NP-complete. Moreover, they shown that the general part sequencing problem not restricted to any robot move cycle in a three machine cell is still intractable.

Levner et al. in [49] addressed a cyclic robot scheduling problem in an automated manufacturing line in which a single robot is used to move parts from one work-station to another, with the goal of optimizing the cycle length. For this problem they proposed an algorithm of complexity $O(m^3 \log m)$ where m is the number of machines.

Many of the results and algorithms in the literature are devoted to robotic flow-shop scheduling problems (e.g., see [7, 13, 20, 33]). However, the robotic cell configuration is very flexible: the robot can easily access the machines in any order, thus producing a large variety of products in form of a job-shop (e.g., see [30, 35, 40, 44]).

In this work we concentrate on the latter class of problems, studying the general problem of sequencing multiple jobs where each job consists of multiple ordered tasks and tasks execution requires simultaneous usage of several resources ([10]).

The remainder of the chapter is organized as follows. In Section 3.2 we formally

describe the problem. The complexity of the problem is analyzed in Section 3.3. A heuristic algorithm is described in Section 3.4, and approximation results on the solution found by that algorithm are addressed in Section 3.5. Finally in Section 3.6 we present computational results.

3.2 Problem definition

Let us consider a cell composed of M machines configured to produce a batch of parts. Part p, $p = 1, ..., P$, requires N_p tasks. Let p_j denote the j^{th} task for part p. Task p_j takes t_{p_j} time units to complete. We allow the general condition that a task requires concurrent usage of more than one cell resource during its execution. Let S_{p_j} be the set of resources required for task p_j. The objective is to determine the schedule for all tasks of all parts so as to minimize the makespan.

Note that the mainstream of research in robotic scheduling is devoted to two classes of production performance measures. The first is the makespan which addresses the case of a finite part set where one is interested in minimizing the maximum completion time of these parts (e.g., see [23, 40, 52, 53]). The other class of models (which is not addressed in this work) assumes that the part set is infinite and attempts to minimize the long run cycle time of the schedule, which is the same as maximizing the throughput rate (e.g., see [13]).

As an example of this model, we examine an assembly cell of $M - 1$ machines plus a material handling robot. The robot will be modelled as machine M. Each part has a processing time associated with each machine. In addition, the robot is needed to load a part on a machine and then to unload the part after the production task has been performed. In our model, each production task is divided into three tasks, namely Load, Make, and Unload. The Load and Unload tasks require both the machine and the robot. The Make operation would normally require only the machine. The formulation also permits the case where the robot is needed to hold the part while certain production tasks are performed. This problem can be easily extended to the case of multiple material handlers, each assigned to a specific set of machines. Note that the solution must take into account precedence restrictions. For instance, we can not Unload until we have Loaded and performed the Make task.

This is an example of operational problems associated with the application of

industrial robots, used to tend a number of machines that have been organized into a machine cell. The cell is used to produce a set of parts at production rates specified by managers. The time required to tend a machine, may be different for each machine according to its location and orientation in the cell, and to the part to be processed. All machines in the cell are dependent on a single robot, so the sequence in which tending tasks are performed may be critical and force certain machines to be in idle.

Let us denote the Load, Make and Unload operations of part p requiring machine i with L_p^i, M_p^i, U_p^i (the machine index i will be omitted when no necessary). Moreover, we shall refer to the problem of scheduling Load, Make and Unload operations for the set of parts in order to minimize makespan as the LMU Problem.

As an example of LMU Problem, in Figure 3.1 is shown a schedule, for three parts and two making machines. In particular, the robot is indicated with R, and the machines with m_1 and m_2 respectively. Moreover, the Make operation for Part 2, namely M_2^2, requires resource R.

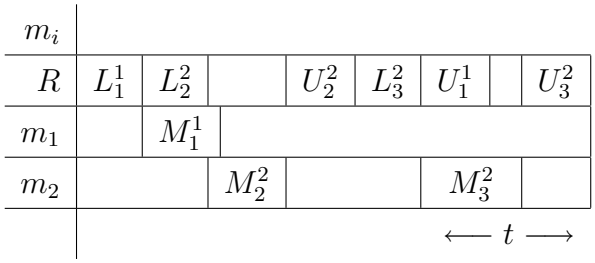

Figure 3.1: An example of LMU Problem.

3.3 NP-completeness result

We show that LMU Problem with general processing times on four machines is NP-hard by a transformation from 3-Partition, which is known to be strongly NP-complete (see Garey and Johnson [29]), to our problem. The 3-Partition problem is defined as follows.

3-Partition problem: Given a set $A = \{a_1, a_2, \ldots, a_{3z}\}$ of $3z$ integers such that $\sum_{t=1}^{3z} a_t = zB$ and $B/4 < a_t < B/2$ for $t = 1, \ldots, 3z$, can A be partitioned into z subsets, $A_1, A_2, \ldots, A_z$, such that $\sum_{a_t \in A_k} a_t = B$ for $k = 1, 2, \ldots, z$?

Theorem 6 *The LMU problem with m=4 is strongly NP-complete*

Proof: For a given instance of the 3-Partition problem let us define a corresponding instance of our problem with $5z$ parts. Let there be $2z$ parts requiring machine 1, z parts requiring machine 2, z parts requiring machine 3, and z parts requiring machine 4. The processing times are:

- $L_p^1 = 1,\ M_p^1 = B,\ U_p^1 = 1, \quad p = 1, \ldots, 2z$
- $L_p^2 = a_{p-2z},\ M_p^2 = B - a_{p-2z} + 2,\ U_p^2 = a_{p-2z}, \quad p = 2z+1, \ldots, 3z$
- $L_p^3 = a_{p-3z},\ M_p^3 = B - a_{p-3z} + 2,\ U_p^3 = a_{p-3z}, \quad p = 3z+1, \ldots, 4z$
- $L_p^4 = a_{p-4z},\ M_p^4 = B - a_{p-4z} + 2,\ U_p^4 = a_{p-4z}, \quad p = 4z+1, \ldots, 5z$

If the 3-Partition problem has a positive answer, then we can construct a schedule of length $Y = 2z(B+2)$ as shown in Figure 3.2.

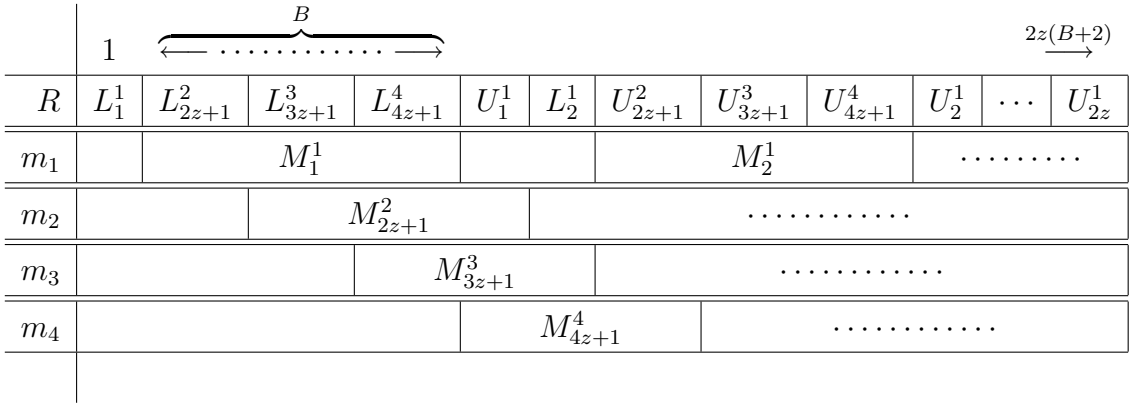

Figure 3.2: An optimal schedule with $Y = 2z(B+2)$.

Now, we are going to show that there is a positive answer to 3-Partition if there exists a feasible schedule with length less than or equal to Y. We observe that the value of the makespan to schedule in any order the $2z$ parts requiring machine 1 is $Y = 2z(B+2)$. This partial schedule has no idle times on machine 1 and has $2z$ idle times on the robot all of length B. The total time required to perform the Load and Unload operations of the remaining $3z$ parts is $2zB$. Note that a feasible schedule for the remaining parts can be obtained scheduling in any order the Load operations of parts on machines 2, 3, 4, on an idle time of the robot. The Make operation can start as soon as the Load is completed. Unload operations can be performed in the successive robot idle time in the same order than the corresponding

Load operations. In a schedule of length Y the robot must have no idle times. This is possible if the sum of the Load (Unload) operations of parts requiring machines $2, 3, 4$ in each robot idle time is equal to B, that is if it exists a 3-Partition of the set A. Thus the problem is strongly NP-complete. □

3.4 The $LMUA$ heuristic

In the following we describe a heuristic algorithm (namely LMUA) which finds a feasible solution to our scheduling problem.

First we observe an active schedule can have idle times either on one of the machines or on the robot which cannot be eliminated trivially. The robot can be idle when all machines are simultaneously processing a Make task. A machine m_i may be idle, waiting for either a Load or an Unload task to be performed, because the robot is busy tending an other machine. Second, for any feasible schedule the maximum completion time is the completion time of the last Unload operation.

The $LMUA$ algorithm proposed is a single pass heuristic in which the loading-unloading sequence and the corresponding schedule are determined once. A list containing the sequence of Load-Make-Unload tasks is built considering any order of the part types. At the beginning the robot R loads all machines. Make operations can start immediately after the preceding Load is performed. Successively the robot unload the machine which ended first the Make task.

In the generic stage of the algorithm the first unselected task in the list is examined. If it is a Make operation it can be scheduled immediately after the loading. Otherwise, the first Load-Unload operation in the list of remaining tasks which tends the machine which has been idle for the longest time is selected. The following is a pseudo-code description of algorithm $LMUA$.

Algorithm $LMUA$

- *Step 1:*

  `Consider an instance with` $M-1$ `machines, one robot (modelled as machine` M`) and` Z `parts.`

1.1 Take any ordering of all the parts (assume the order $1, \ldots, Z$);

1.2 Build the list of tasks:

$$LT = \{L_1^k, M_1^k, U_1^k, L_2^k, M_2^k, \ldots, L_Z^k, M_Z^k, U_Z^k\};$$

1.3 Build the list of Tasks Make that require the resource robot:

$LTR = \{M_i^k |$ part i requires the robot during the Make on $m_k\}$;

1.4 Build the list of processing times:

$$PT = \{p_{L_1^k}, p_{M_1^k}, p_{U_1^k}, \ldots, p_{U_Z^k}\};$$

1.5 Build the list of the instants of time at which the machines are available: $AT = \{At_1, At_2, \ldots, At_M\}$;

1.6 Initialize the current scheduling time at $t = 0$;

1.7 Initialize the list of the tasks that can be processed at the current t with all the Load tasks:

$$LTA = \{L_1^k, L_2^k, \ldots, L_Z^k\};$$

1.8 Build the list reporting the instants of time at which the tasks in list LT could start their execution:

$$FTA = \{Ft_{M_1^k}, Ft_{U_1^k}, \ldots, Ft_{U_Z^k}\};$$

1.9 Set the values of the variables in FTA equal to infinite.

- *Step 2:*

While $LT \neq \emptyset$ or $LTA \neq \emptyset$ do:

2.1 Scan tasks in list LTA and

2.1.1 if there exists a task Make that at time t requires a machine m_k that is available according to list AT then go to Step 3; otherwise

2.1.2 if the robot is available at time t and there exists either a task Load whose corresponding Make operation requires a machine which is available, or a task Unload, then go to Step 4 (tie breaks choosing the Load or Unload task waiting for more time in the list).

2.2 If there does not exist a task obeying Step 2.1.2 or Step 2.1.3, then:

2.2.1 Increase $t = t + 1$;

2.2.2 Update lists LTA and LT, by moving tasks from LT to LTA according to FTA.

- *Step 3:*

3.1 Schedule the Make task selected starting from t on the required machine;

3.2 Set equal to infinite the variable in AT associated with the machine handling the Make task.

3.3 Update in FTA the earlier starting time of the Unload task associated with the processed Make task, setting it to the finishing time of the latter;

3.4 Delete from LTA the processed Make task.

3.5 Set $t = t + 1$;

3.6 Go to Step 2.

- *Step 4:*

If the selected task is a Load (whose Make task does not require the robot) or an Unload, then:

4.1 Process the task;

4.2 Update the instant of time At_M at which the robot will be available again according to the processing time of the task executed; set t to this latter value.

4.3 If a Load task has been selected, then update in FTA the earlier starting time of the Make task associated;

4.4 If an Unload task has been selected, set to $t + 1$ the time at which the machine which have processed its predecessor Make task will be available; update the instant of time At_M at which the robot will be available again according to the processing time of the task executed; update t accordingly.

4.5 Delete from LTA the processed task;

4.6 Go to Step 2.

If the selected task is a Load task such that the following Make task requires the presence of the robot (as shown in list LTR), then:

4.7 Process the task Load and immediately after the following task Make;

4.8 Update the variable in list AT indicating when the robot will be available again (i.e., after an interval of time equal to the sum of the processing times of the Load and the Make operations), while set the availability of the machine which has processed the Make task equal to infinity after this Make operation has been performed;

4.9 Update the variable in list FAT of the earlier starting time of the corresponding Unload task, i.e., t plus the processing times of the Load and the Make operations performed, say $p_{L_i^k}$ and $p_{M_i^k}$ respectively;

4.10 Update $t = t + p_{L_i^k} + p_{M_i^k}$;

4.11 Delete from LTA the Load task;

4.12 Delete from LT the Make operation.

4.13 Go to Step 2.

- *Step 5:* Return the makespan: $C_{\max} := t$

In Figure 3.3 we show an application of $LMUA$ algorithm with 4 parts: Make operations for parts 1 and 3 require machine m_2; Make operations for part 2 and 4 require machine m_1; moreover, Make operation for part 2, namely M_2^1, also requires resource R.

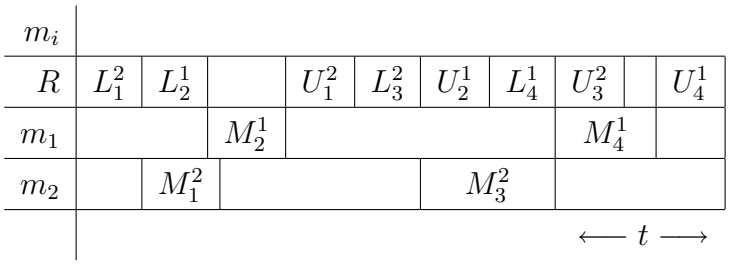

Figure 3.3: An application of $LMUA$ Algorithm.

In order to determine the computational complexity of the $LMUA$ algorithm, note that, for a given instance of Z parts, there are $3Z$ tasks and:

Step 1: the lists can be constructed in $O(Z)$;

Step 2: the cycle **while** is repeated at most $3Z$ times, and the selection of the task requires at most $3Z$ comparisons;

Step 3: runs in $O(c)$, where c is a constant;

Step 4: can be processed in $O(c)$;

Step 5: runs in $O(1)$;

Hence, for a given instance of the LMU problem, denoting with Z the size of the input, $LMUA$ algorithm has a worst case complexity $O(Z^2)$.

3.5 Graph model and approximation result

In this section we prove an approximation result on the $LMUA$ solution, moving from a graph model of the problem.

Let us consider a graph $G = (V, E \cup A)$ where the set of vertices corresponds to the set of tasks, and the set of disjunctive edges E represent couples of tasks which cannot be executed simultaneously because they share some resource. The set of oriented arcs $(i, j) \in A$ is determined by the transitive closure of the precedence relations defined among tasks defining a job . A weighting function D associates to each vertex (task) i its execution time d_i. An example of the graph G is shown in Figure 3.4.

Let us introduce some definition in order to examine the structure of the graph. In the sequel we will denote by *independent set* a set of vertices which are pairwise

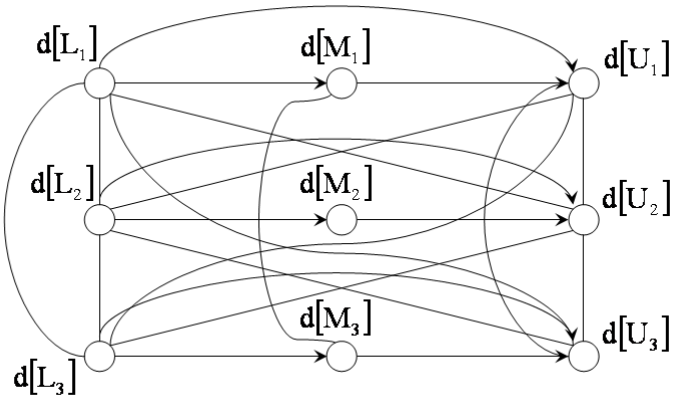

Figure 3.4: The general structure of the graph G.

independent (i.e. non connected), and by *clique* a set of vertices which induces a complete subgraph (i.e. a set of pairwise connected vertices). The weight of a clique is the sum of the weights of its vertices. The maximum weighted clique of G is a clique of maximum weight.

With respect to the disjunctive graph models already introduced for shop scheduling problems (e.g., see [65]), the graph G has a special structure. In fact it can be seen as constituted by a set of cliques whose vertices represent operations requiring a certain machine to be executed, which intersect in a "central" clique whose vertices represent tasks requiring both a machine and the robot.

Our first observation is that cliques correspond to set of tasks which have to be scheduled sequentially, while independent sets correspond to set of vertices which might be processed in parallel. The graph structure will be used to derive bounds on the heuristic algorithm performance.

Now, we give bounds on the performance of the presented algorithm. A straightforward lower bound on the value of the makespan is given by the maximum weighted clique of the graph G, that is by the set of pairwise connected vertices, of largest weight. The upper bound on the value of the makespan produced by $LMUA$ algorithm is given by the size of the maximum weighted *star* in G, where a star is the union of a vertex x and its neighborhood $N(x)$. The result can be proved by induction on the number of tasks, by the following theorems.

Lemma 3 *Let $F(I)$ denotes the value of the makespan produced by $LMUA$ algo-*

rithm on problem instance I. Then:

$$F(I) \leq \max_{x \in V} \{d_x + \sum_{y \in N(x)} d_y\}$$

Proof: Assume it holds at step $i-1$ of the algorithm, we prove it is true at step i.

Observe that, when scheduling the i^{th} task, say x, with starting time s_x, all time instant preceding s_x are occupied by vertices (tasks) $y \in N(x)$ (neighbors in G of vertex x).

More precisely, for all instant preceding s_x, if a machine is idle, then the robot is busy. For the lack of simultaneous idle times on both machines and robot the vertex x can be postponed no longer than the time to process $N(x)$. Then the greatest time to complete processing of x is less or equal to $d_x + \sum_{y \in N(x)} d_y$. Hence, the maximum completion time of the schedule is less or equal to $\max_{x \in V} \{d_x + \sum_{y \in N(x)} d_y\}$. □

Theorem 7

$$\frac{F(I)}{OPT(I)} \leq 2$$

Proof: Any star in G contains at most vertices of the clique induced by operations which require a robot and any vertex of one of the cliques induced by making operations. Clearly, the maximum weight of a star is bounded by twice the size of the maximum weighted clique. Using the maximum weighted clique as a lower bound on the optimal value of the makespan $OPT(I)$, the inequality follows. □

3.6 Computational results

In this section we present some computational experiments with $LMUA$ algorithm on randomly generated problems. We considered several cell configurations with $m \in [4, 10]$ machines and one robot tending all the machines. For each configuration we considered an increasing number of jobs $n \in [10, 80]$. Note that, for instance, 80 jobs corresponds to 240 tasks. Processing times for each loading and unloading operations are generated randomly, using a uniform distribution in the range $[20 - 70]$ time units. Processing times of Make operations are generated randomly using a uniform distribution in the range $[120 - 360]$ time units. We considered different scenarios, associated with the probability $p_r = 0\%, 10\%, 20\%, 30\%,$ and 40% that a generic part requires the robot during the Make operation. The algorithm implementation

was done in WINDOWS/C environment on a AMD Athlon PC running at 900 MHz. CPU times to find the optimum solution for each run were computed, and were very low. In fact, the maximum CPU time (in seconds) was $2.67s$, found for the combination $(p_r = 40\%, m = 6, n = 80)$, whereas the average time was $0.38s$.

Results are summarized in the following tables, in which are reported, for each cell configuration, the values of the makespan depending on the number n of jobs. Each table is associated with a scenario corresponding to a probability p_r.

$p_r = 0$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	1977.9	3265.5	4655.9	6774.6	8110.6	9300.4	10711.1	12077.3
6	1681.8	2733.4	4157.2	5460.4	6673.2	8117.8	9633.8	11220.8
8	1460	2712.1	3784.2	5400.4	6660	8014	9054.3	10482
10	1438.1	2678.2	3723.2	5314.5	6638.7	7754.5	8698.9	10458.3

Table 3.1: Scenario with $p_r = 0$

First let us consider a scenario in which jobs do not require the resource robot during the Make operation (Table 3.1).

Observe that, for a given m, the makespan increases as n increases, while it decreases, for a given n, as the number of machines m increases.

To evaluate the behaviour of the makespan we report the chart in Figure 3.5.

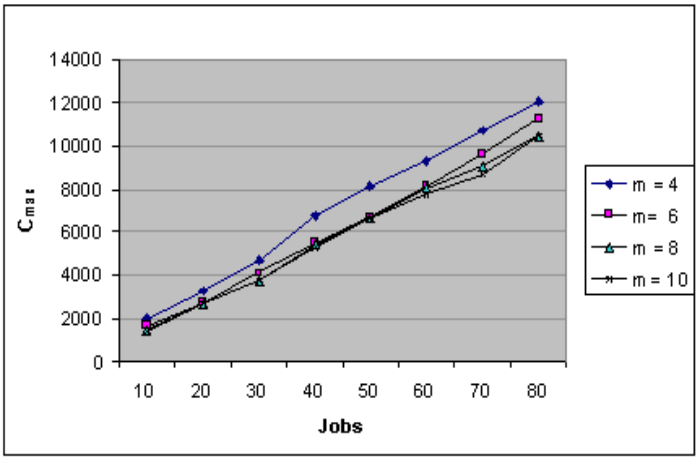

Figure 3.5: Trends of the makespan as the number n of jobs increases.

The makespan value, given a fixed number m of machines, seems to be linearly

dependent on the number of jobs. For a certain range of the number of jobs ($n \in [10, 30]$), the trends are very similar, and it seems that the number m of machines does not influence C_{max}. As n increases, instead, the trends are well distinguished, and the difference between the makespan values is much more observable when m passes from 4 to 6, than when it passes from 6 to 8 or 10.

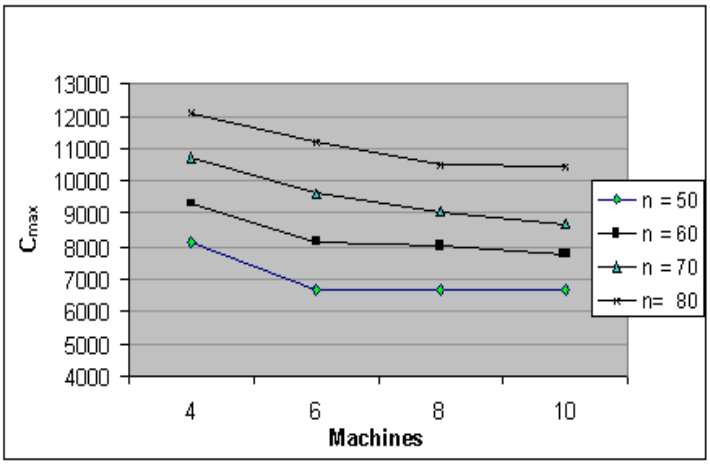

Figure 3.6: Trends of the makespan as the number m of machines increases.

Now we analyze the behaviour of the makespan referring to the increase of the number m of the machines, for a given n. Figure 3.6 shows how C_{max} decreases proportionally as the number m increases. Moreover, from Table 3.1 it is easy to see that when the number of jobs passes from $n = 30$ to $n = 40$ the variation of the associated makespan values is higher than in the other cases.

Finally we study what happens when the probability p_r of the existence of a job requiring the robot during the Make operation is greater than zero.

$p_r = 1$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	2069.9	3486.6	5118.2	7300	8923.4	9999.7	11690.6	13615.2
6	1747.8	3096.8	4536	6292.1	7830.5	9298.7	11094.8	13057.5
8	1534.3	3077.5	4487	6193.8	7510.2	8952.5	10382.5	12008.9
10	1506.4	2969.3	4322	6125.6	7503.8	8771.4	10138.7	11849.9

Table 3.2: Scenario with $p_r = 1$

Tables 3.2, 3.3, 3.4 and 3.5 summarize the results for each different scenario.

$p_r = 2$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	2152.5	3674.4	5320.9	7808.6	9428.6	10962.4	12574.6	14435.3
6	1818.3	3458.4	5143.6	7109.3	8806.4	10647.6	12018.1	14319
8	1775.3	3373.7	4954	6761.6	8430.1	10336.6	11880	13752
10	1591.9	3367.3	4919.1	6651.7	8283.1	9914.6	11270.6	13356.7

Table 3.3: Scenario with $p_r = 2$

$p_r = 3$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	2301.7	3743.1	5611.7	8289	10223	11656.9	13882.7	15894.9
6	1929.2	3644.8	5462.2	7798.4	9358.6	11496.9	13212	15787.6
8	1821	3465.1	5440.8	7341.7	9207.9	10969.7	12786.3	15046.8
10	1760.1	3412.8	5092	7252.4	9048.8	10923.3	12089.6	14451

Table 3.4: Scenario with $p_r = 3$

$p_r = 4$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	2383	4134	6073.6	9009.7	10859.4	12837.6	14857.8	17176.5
6	2024.3	3960.9	5876.7	8243.3	10232.2	12491.2	14268.5	16624.1
8	1837.3	3671.6	5734.9	7969.9	9644.2	11775	13719.8	15950.3
10	1815.5	3603.2	5406.4	7700.9	9577.2	11424.9	13693.4	15296

Table 3.5: Scenario $p_r = 4$

Note that as p_r increases C_{max} decreases proportionally. The chart in Figure 3.7 shows that the makespan values increase proportionally with the probability p_r and the number of jobs n.

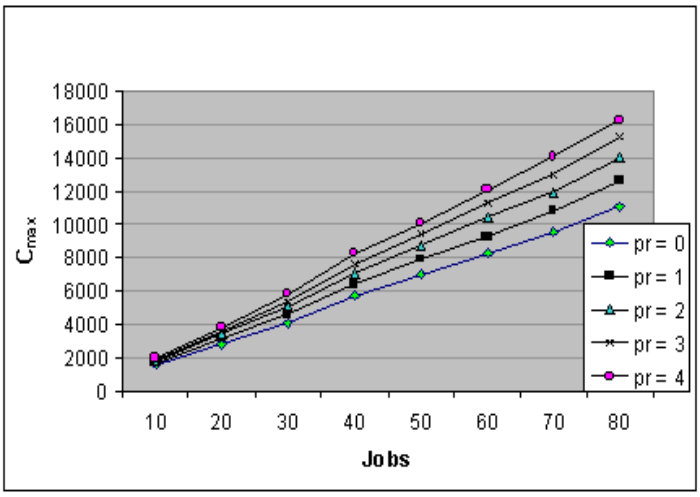

Figure 3.7: Makespan values, for a given m, as n and p_r increase.

The chart in Figure 3.8, instead, shows that the variation of the makespan, when the probability p_r increases, is not proportional to the number m of machines, for a given n. In fact, it can be seen how the influence of p_r on the makespan tends to decrease as m increases.

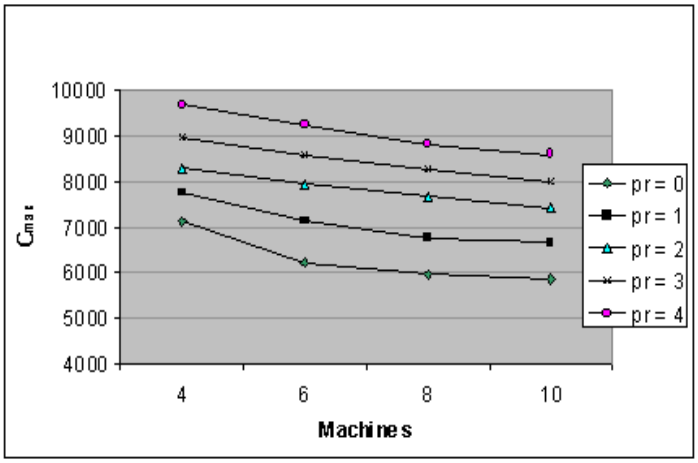

Figure 3.8: Makespan values, for a given n, as m and p_r increase.

A final analysis is devoted to study the possible efficiency improvement of the

robotic cell. In particular, we examine what is more profitable, whether to reduce the processing times of Make operations, improving the machines' efficiency, or to reduce the processing times for loading and unloading operations, modifying the Robot configuration.

First we analyze, in the case $p_r = 0\%$, what happens if the range of the uniform distribution for the processing times of Make operations is decreased of 30%, from $[120 - 360]$ to $[120 - 268]$ time units.

$p_r = 0$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	1600.1	2781.3	3831.6	5735.3	6806.3	7877.1	8862.9	10284.7
6	1412.1	2481	3707.8	5022.4	6300.3	7630	8586.4	10150
8	1263.9	2409.8	3585.1	4948.5	6274.4	7480	8405.4	9343.2
10	1210.7	2392.1	3459.2	4918.7	6256.2	7290.3	8082.4	9900.3

Table 3.6: Reducing processing times of Make operations

Table 3.6 summarizes the results obtained in this new scenario, and Table 3.7 reports the percentage of reduction of C_{max}.

$p_r = 0$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	19.10%	14.83%	17.70%	15.34%	16.08%	15.30%	17.25%	14.84%
6	16.04%	9.23%	10.81%	8.02%	5.59%	6.01%	10.87%	9.54%
8	13.43%	11.15%	5.26%	8.37%	5.79%	6.66%	7.17%	10.86%
10	15.81%	10.68%	7.09%	7.45%	5.76%	5.99%	7.09%	5.34%

Table 3.7: Percentage reduction of C_{max}

Now we analyze what happens when the range of the uniform distribution for the processing times of Load and Unload operations is decreased of 30%, i.e., from $[20 - 70]$ to $[20 - 55]$ time units.

Table 3.8 summarizes the results obtained in this case, and Table 3.9 reports the percentage of reduction of C_{max}.

It is easy to observe that a decrease of the processing times in both cases brings a reduction of C_{max} even if the latter is not proportional to the former. In fact, the maximum reduction obtained is 19.89% when processing times of loading and

$p_r = 0$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	1833.9	3071.5	4312.1	6273.3	7639.3	8816.5	10015.8	11228.9
6	1567.7	2400.4	3607.3	4847.7	5467.2	6682	7717.9	9035.2
8	1334.5	2398.5	3264.2	4475.5	5458.9	6653.9	7527.2	8697.5
10	1301.5	2378.7	3138.8	4298.5	5429.5	6354	7097.6	8415.5

Table 3.8: Reducing processing times of Load and Unload operations

$p_r = 0$	⟵**Jobs (n)** ⟶							
Mac. (m)	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
4	7.28%	5.94%	7.38%	7.40%	5.81%	5.20%	6.49%	7.02%
6	6.78%	12.18%	13.23%	11.22%	18.07%	17.69%	19.89%	19.48%
8	8.60%	11.56%	13.74%	17.13%	18.03%	16.97%	16.87%	17.02%
10	9.50%	11.18%	15.70%	19.12%	18.21%	18.06%	18.41%	19.53%

Table 3.9: Percentage reduction of C_{max}

unloading operations are decreased, and 19.10% when processing times of Make operations are decreased instead.

3.7 Conclusions

In this chapter we presented the general problem of sequencing multiple jobs where each job consists of multiple ordered tasks and tasks execution requires simultaneous usage of several resources. The case of an automatic assembly cell is examined. The NP-completeness in the strong sense of the problem is proved for an automatic assembly cell with four machines. A heuristic algorithm is proposed, with a guarantee approximation on the optimal solution. For this we give computational results for an assembly cell with different number of machines and one robot. The procedure at each iteration selects a task, based on the partial schedule obtained for the parts that had already been loaded for the assembly process. That characteristic of the proposed algorithm indicates that the presented approach can also be applied in on-line scenarios as well as in dynamic scheduling environment.

Further research will be devoted to the extension of this approach to different cell configuration, such us the case in which a task requires k additional resources out of a set of m available ones.

Bibliography

[1] AGNETIS, A., 2000, Scheduling no-wait robotic cells with two and three machines, *European Journal of Operational Research*, 123 (2), 303–314.

[2] AGNETIS, A., ARBIB, C., LUCERTINI, M., NICOLÓ, F., 1990, Part routing in Flexible Assembly Systems, *IEEE Transactions on Robotics and Automation*, 6 (6), 697–705.

[3] AGNETIS, A., LUCERTINI, M., NICOLÓ, F., 1993, Flow Management in Flexible Manufacturing Cells with Pipeline Operations, *Management Science*, 39 (3), 294–306.

[4] AGNETIS, A., MACCHIAROLI, R., 1998, Modelling and Optimization of the Assembly Process in a Flexible Cell for Aircraft Panel Manufacturing, *International Journal of Production Research*, 36 (3), 815–830.

[5] AGNETIS, A., PACCIARELLI, D., 2000, Part sequencing in three-machine no-wait robotic cells, *Operations Research Letters*, 27, 185–192.

[6] ANEJA, Y.P., KAMOUN, H., 1999, Scheduling of parts and robot activities in a two machine robotic cell, *Computer and Operation Research*, 26 (4), 297-312.

[7] ASFAHL, C.R., *Robots and Manufacturing Automation*, Wiley, New York, NY, 1985.

[8] ASKIN, R.G., STANDRIDGE, C.R., 1993, *Modelling and Analysis of Manufacturing Systems* (John Wiley and Sons).

[9] ALPERT, C.J., KAHNG, A.B., 1995, Recent directions in netlist partitioning: A survey, *VLSI Journal*, vol. 19, no. 1-2, pp. 1–81.

[10] BAKER, K.R., 1976, *Introduction to Sequencing and Scheduling* (John Wiley and Sons).

[11] BAKKER, J.J., 1989, DFMS: Architecture and Implementation of a Distributed Control System for FMS, Delft, Techn. Univ., Diss.

[12] BAUER, A., BOWDEN, R., BROWNE, J., J. DUGGAN and G. LYONS, 1991, *Shop Floor Control Systems, From design to implementation*, Chapman and Hall, London.

[13] BLAZEWICZ, J., KUBIAK, W., SETHI, S.P., SORGER, G., SRIKANDARAJAH, C., 1992, Sequencing of parts and robot moves in a robotic cell, *International Journal of Flexible Manufacturing Systems*, 4, 331–358.

[14] BRAUNER N., FINKE G., 2001, Optimal moves of the material handling system in a robotic cell, *International Journal of Production Economics*, 74, 269–277.

[15] BUI, T., JONES, C., 1993, A heuristic for reducing fill in sparse matrix factorization, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, pp. 445–452.

[16] CARAMIA, M., DELL'OLMO, P., ONORI, R., 2004, Minimum Makespan Task Sequencing with Multiple Shared Resources, *Robotics and Computer Integrated Manufacturing*, 20, 1.

[17] CASTELFRANCHI, C., 1995, Guarantees for Autonomy in Cognitive Agent Architecture, in M. Wooldridge and N. Jennings (Eds.) *Intelligent Agents: Theories, Architectures, and Languages*, Lecture Notes in Artificial Intelligence, Vol. 890, Springer-Verlag, Heidelberg, pp. 56–70.

[18] CHRYSSOLOURIS, G., 1991, An Approach for Allocating Manufacturing Resources to Production Tasks, *Journal of Manufacturing Systems*, 10, 5, p. 368.

[19] CHRYSSOLOURIS, G., 1992, *Manufacturing Systems, Theory and Practice*, Springer-Verlag, New York.

[20] CRAMA, Y., VAN DE KLUNDERT, J., 1997, Cyclic scheduling of identical parts in a robotic cell, *Operations Research*, 45, 952–965.

[21] CRAMA, Y., VAN DE KLUNDERT, J., 1997, Robotic flowshop scheduling is strongly NP-complete, Ten Years LNMB, W.K. Klein Haneveld, O.J. Vrieze and L.C.M. Kallenberg (eds.), CWI Tract 122, Amsterdam, The Netherlands, pp. 277–286.

[22] CHRISTENSEN J., 1994, Holonic Manufacturing Systems - Initial Architecture and standard directions, in Proceedings of the first European Conference on Holonic Manufacturing Systems, Hannover.

[23] CHU, C., PROTH, J.M., 1996, Single machine scheduling with chain structures procedence constraints and separation time windows, *IEEE Transactions on Robotics and Automation*, 12 (6), 835–844.

[24] DETAND J., 1993, A Computer Aided Process Planning System Generating Non- Linear Process Plans, PhD Thesis, KU Leuven, October 1993.

[25] EL MAGRINI H., and J. TEGHEM, 1996 (February 19-23). Efficiency of metaheuristics to schedule general flexible job-shop, Pre-prints of the Ninth International Working Seminar on Production Economics, Innsbruck, Austria, Vol.2, pp. 343–363.

[26] FIDUCCIA, C.M., MATTHEYSES, R.M., 1982, A linear-time heuristic for improving network partitions, in Pro-ceedings of the 19th ACM/IEEE Design Automation Conference, pp. 175–181.

[27] FRENCH S., 1982, *Sequencing and Scheduling, An Introduction to the Mathematics of the Job-Shop*, John Wiley and Sons, Chichester, England.

[28] GAREY, M.R., JOHNSON, D.S., STOCKMEYER, L., 1976, Some simplified NP-complete graph problems, *Theoretical Computer Science*, 1, pp. 237–267.

[29] GAREY, M.R., JOHNSON, D.S., 1979, *Computers and Intractability: A guide to the Theory of NP-completeness* (San Francisco: W.H. Freeman).

[30] GLASS, C.A., SHAFRANSKY, Y.M., STRUSEVICH, V.A., 2000, Scheduling for parallel dedicated machines with a single server, *Naval Research Logistics*, 47, pp. 304–328.

[31] GOLDBERG, M.K., BURSTEIN, M., 1983, Heuristic improvement techniques for bisection of vlsi networks, in Proc. IEEE Intl. Conf. Computer Design, pp. 122–125.

[32] GOLDRATT E.M., COX J., 1984, *The Goal, a Process of Ongoing Improvement, North River Press*, Croton-on-Hudson (N.Y.).

[33] HALL, N.G., KAMOUN, H., SRISKANDARAJAH C., 1997, Scheduling in robotic cells: classification, two and three machine cells, *Operations Research*, 45 (3), 421–439.

[34] HALL, N.G., KAMOUN, H., SRISKANDARAJAH C., 1998, Scheduling in robotic cells: complexity and steady state analysis, *European Journal of Operational Research*, 109, 43–65.

[35] HALL, N.G., POTTS, C.N., SRISKANDARAJAH, C., 2000, Parallel machine scheduling with a common server, *Discrete Applied Mathematics* 102(3), pp. 223–243.

[36] HARTLEY, J., 1983, *Robots at Work*, North-Holland, Amsterdam.

[37] HENDRICKSON, B., LELAND, R.W., 1993, A multilevel algorithm for partitioning graphs, tech. rep., Sandia National Laboratories, Alburquerque, NM.

[38] HENDRICKSON, B., LELAND, R.W., PLIMPTON, S., 1995, An efficient parallel algorithm for matrix-vector multiplication, *Int. J. High Speed Computing*, vol. 7, no. 1, pp. 73–88.

[39] HENDRICKSON, B., LELAND, R.W., 1995, The Chaco user's guide, version 2.0, tech. rep., Sandia National Laboratories, Alburquerque, NM.

[40] HERTZ, A., MOTTET, Y., ROCHAT, Y., 1996, On a Scheduling Problem in a Robotized Analytical System, *Discrete Applied Mathematics*, 65, pp. 285–318.

[41] HOITOMT, D., LUH, P., MAX, E. and PATTIPATI, K., 1989 (May). Schedule Generation and Reconfiguration for Parallel Machines, Proc. of the 1989 IEEE int. Conf. on Robotics and Automation, pp. 528–533.

[42] HOITOMT D., P. LUH, K. PATTIPATI, 1990 (May), A Lagrangian Relaxation Approach to Job Shop Scheduling Problems, Proc. 1990 IEEE Int. Conference on Robotics and Automation, Cincinnati, Ohio, USA, pp. 1944–1949.

[43] IOVANELLA, A., ONORI, R., 2005, Managing production and material handling on a two stage flexible manufacturing system, submitted to MAPSP 2005.

[44] JENG, W.D., LIN, J.T., WEN, U.P., 1993, Algorithms for sequencing robot activities in a robot-centered parallel-processor workcell, *Computer and Operations Research*, 20 (2), pp. 185–197.

[45] KARYPIS, G., KUMAR, V., AGGARWAL, R., SHEKHAR, S., Hypergraph partitioning using multilevel approach: applications in VLSI domain, *IEEE Transactions on VLSI Systems*, to appear.

[46] KARYPIS, G., KUMAR, V., GRAMA, A., GUPTA, A., 1994, Introduction to Parallel Computing: Design and Analysis of Algorithms. Redwood City, CA: Benjamin/Cummings Publishing Company.

[47] KARYPIS, G., KUMAR, V., 1998, MeTiS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 3.0, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis.

[48] KARYPIS, G., KUMAR, V., AGGARWAL, R., SHEKHAR, S., 1998, hMeTiS A Hypergraph Partitioning Package Ver-sion 1.0.1. University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis.

[49] KATS, V., LEVIT, V.E., LEVNER E., 1997, An improved algorithm for cyclic flowshop scheduling in a robotic cell, *European Journal of Operational Research*, 97, 500–508.

[50] KERNIGHAN, B.W., LIN, S., 1970, An efficient heuristic procedure for partitioning graphs, *The Bell System Technical Journal*, vol. 49, pp. 291–307.

[51] KERNIGHAN, B.W., SCHWEIKERT, D.G., 1972, A proper model for the partitioning of electrical circuits, in Proceedings of the 9th ACM/IEEE Design Automation Conference, pp. 57–62.

[52] KISE, H, 1991, On an automated two-machines flowshop scheduling problem with infinite buffer, *Journal of the Operations Research Society of Japan* 34 (3), 354–361.

[53] KISE, H., SHIOYAMA, T., IBARAKI, T., 1991, Automated two machines flowshop scheduling: a solvable case, *IIE Transactions*, 23 (1), 10–16.

[54] KOESTLER, A., 1967, *The Ghost in the Machine*, Hutchinson and Co, London.

[55] LEI M., MITCHELL M.T., 1997, A preliminary Research of Market Appraoch for Mass Customization Assembly Planning and Scheduling, Technical Report Hong Kong University of Science and Technology, Dept. of Industrial Engineering and Engineering Management.

[56] LEITAO, P., RESTIVO, F., 1999, A Layered Approach to Distributed Manufacturing, in proceedings of 1999 Advanced Summer Institute, Belgium.

[57] LENGAUER, T., 1990, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley and Sons.

[58] LEONDES, C., 2001, *Computer-Integrated Manufacturing*, CRC Press.

[59] LUH P., D. HOITOMT, 1993, Scheduling of Manufacturing Systems Using the Lagrangian Relaxation Technique, *IEEE Trans. on Automatic Control*, V. 38, N. 7.

[60] MALONE, T. and CROWSTON, K., 1994, The Interdisciplinary Study of Co-ordination, *ACM Computing Surveys*, Vol. 26(1), March , pp. 87–119.

[61] MAKAROV, I.M., RIVIN, E.I., 1990, *Modeling of Robotic and Flexible Manufacturing Systems*, Hemisphere Pub.

[62] MESA, 1997, Manufacturing Execution Systems, http://www.mesa.org/.

[63] NWANA, H., LEE, L. and JENNINGS, N., 1996, Coordination in Software Agent Systems, *BT Technology Journal*, 14(4), pp.79–88.

[64] OKINO, N., 1993, Bionic Manufacturing System, in J. Peklenik (Ed.) CIRP, *Flexible Manufacturing Systems: Past-Present-Future*, pp. 73–95.

[65] PINEDO, M., 1994, *Scheduling: theory, algorithms and systems*, Prentice Hall.

[66] QU, C.W., RANKA, S., 1997, Parallel incremental graph partitioning, *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 8, pp. 884–896.

[67] REMBOLD, U., B.O. NNAJI, A. STORR, 1993, *Computer Integrated Manufacturing and Engineering*, Addison-Wesley Publishing Company, Wokingham, England.

[68] SEIDEL D., M. MEY, 1994, *IMS - Holonic Manufacturing Systems: System Components of Autonomous Modules and their Distributed Control*, HMS project (IMS-TC5).

[69] SIHN, W., 1997, The Fractal Factory: A Practical Approach to Agility in Manufacturing, Proceedings of the 2nd World Congress on Intelligent Manufacturing Processes and Systems, pp. 617–621.

[70] SOUSA, P. and RAMOS, C., 1999, A Distributed Architecture and Negotiation Protocol for Scheduling in Manufacturing Systems, *Computers in Industry*, 38, 2, pp. 103–113.

[71] SOUSA, P., SILVA, N., HEIKKILA, T., KOLLINGBAUM, M., VALCKENAERS, P., 1999, Aspects of cooperation in Distributed Manufacturing Systems, in Proceedings of the 2nd International Workshop on Intelligent Manufacturing Systems (IMS-Europe'99), Leuven, Belgium, pp. 695–717.

[72] SYCARA, K., 1989, Multi-Agent Compromise via Negotiation, in L. Gasser and M. Huhns (Eds.) *Distributed Artificial Intelligence 2*, Morgan Kaufmann.

[73] THARUMARAJAH, A., J. WELLS, L. NEMES, 1996. Comparison of bionic, fractal and holonic manufacturing system concepts, *International Journal of Computer Integrated Manufacturing*, Vol. 9, No.3, pp. 217–226.

[74] VALCKENAERS, P., 1993, Flexibility for Integrated Production Automation, PhD. thesis K.U.Leuven.

[75] VALCKENAERS, P., VAN BRUSSEL, H., BONNEVILLE, F., BONGAERTS, L. and WYNS, J., 1994, IMS Test Case 5: Holonic Manufacturing Systems, Proceedings of the IMS Workshop at IFAC'94, Vienna.

[76] VAN DE KLUNDERT, J., 1996, Scheduling problem in automated manufacturing, Dissertion no. 96-35, Faculty of Economics and Business Administration, University of Limburg , Maastricht.

[77] WARNEKE, H.J., 1993, *The Fractal Company*, Springer-Verlag.

[78] WILHELM, W.E., 1987, Complexity of sequencing tasks in assembly cells attended by one or two robots, *Naval Research Logistics*, 34, pp. 3447–3463.

[79] WILLIAMSON, D.T.N., 1967, System 24 - A New Concept of Manufacture, *Proceedings of the 8th International Machine Tool and Design Conference*, pp. 327-376, Pergamon Press.