Structural Analysis of Combinatorial Optimization Problem Characteristics and their Resolution using Hybrid Approaches

ALESSIO GUERRI

Advisor	Research Supervisor
Prof. Ing. Maurelio Boari	Prof. Ing. Michela Milano

A Thesis submitted to the University of Bologna
in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy in Computer Science

Alma Mater Studiorum - University of Bologna
DEIS - Department of Electronics, Computer Science and Systems
April, 12, 2007

Cycle XIX ING-INF/05

Dissertation for the Degree of Doctor of Philosophy in Computer Science presented at the University of Bologna in 2007.

A. Guerri, *Structural Analysis of Combinatorial Optimization Problem Characteristics and their Resolution using Hybrid Approaches*, 204 pp, Bologna, 2007.

Abstract

Many combinatorial problems coming from the real world may not have a clear and well defined structure, typically being dirtied by side constraints, or being composed of two or more sub-problems, usually not disjoint. Such problems are not suitable to be solved with pure approaches based on a single programming paradigm, because a paradigm that can effectively face a problem characteristic may behave inefficiently when facing other characteristics. In these cases, modelling the problem using different programming techniques, trying to "take the best" from each technique, can produce solvers that largely dominate pure approaches. We demonstrate the effectiveness of hybridization and we discuss about different hybridization techniques by analyzing two classes of problems with particular structures, exploiting Constraint Programming and Integer Linear Programming as solving tools and Algorithm Portfolios and Logic Based Benders Decomposition as integration and hybridization frameworks.

Keywords: Constraint Optimization, Constraint Programming, Integer Linear Programming, Algorithm Portfolios, Benders Decomposition, Combinatorial Auctions, Allocation and Scheduling

Guerri Alessio
DEIS - Department of Electronics, Computer Science and Systems
Alma Mater Studiorum - University of Bologna
V.le Risorgimento 2, 40136, Bologna, Italy

April, 12, 2007

Previous Publications

I composed this dissertation by myself. Parts of this dissertation appeared, or are in the process of being published, in the following publications, subject to peer review.

- A. Guerri and M. Milano. CP-IP techniques for the bid evaluation in combinatorial auctions. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP2003)*, pages 863–867, Kinsale, Ireland, September 2003.
- C. Gebruers, A. Guerri, B. Hnich, and M. Milano. Making choices using structure at the instance level within a case based reasoning framework. In *Proceedings of the 1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, (CPAIOR 2004)*, pages 380–386, Nice, France, May 2004.
- A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI 2004)*, pages 475–479, Valencia, Spain, August 2004.
- C. Gebruers and A. Guerri. Machine learning for portfolio selection using structure at the instance level. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, page 794, Toronto, Canada, September 2004.
- L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI05)*, pages 1517–1518, Edinburgh, Scotland, August 2005.
- L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling

for MPSoCs via decomposition and no-good generation. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP2005)*, pages 107–121, Sitges, Spain, September 2005.

- M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *Proceedings of the conference on Design, automation and test in Europe (DATE06)*, pages 3–8, Munich, Germany, March 2006.

- L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation, scheduling and voltage scaling on energy aware MPSoCs. In *3rd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR2006)*, pages 44–58, Cork, Ireland, June 2006.

- A. Guerri, and M. Milano. The importance of Relaxations and Benders Cuts in Decomposition Techniques: Two Case Studies. In *Proceedings of the Doctoral Programme of the 12th International Conference on Principles and Practice of Constraint Programming (CP2006)*, pages 162–167, Nantes, France, September 2006.

- L. Benini, D. Bertozzi, A. Guerri, M. Milano, and M. Ruggiero. A fast and accurate technique for mapping parallel applications on stream-oriented MPSoCs platforms with communication-awareness. In *International Journal of Parallel Computing*, to appear.

Contents

List of Figures

List of Tables

Part I

Introduction & Preliminaries

Chapter 1

Introduction

The thesis defended in this dissertation is the following:

Constraint Programming and Integer Linear Programming are effective programming paradigms for dealing with Combinatorial Optimization Problems. The structural characteristics of some classes of Combinatorial Optimization Problems can guide the selection of the solving approach. The structure can also suggest to develop solvers based on both the approaches, or to split the problem in two or more subproblems solved with different approaches.

This Chapter introduces the arguments discussed in the dissertation and is organized as follows: Section 1.1 discusses about the motivations underlying our work; Section 1.2 introduces the problems analyzed to support our thesis and Section 1.3 gives an overview of the dissertation organization.

1.1 Modelling the world

Dealing with decisions is an important part of the real life. Taking a decision that will not clash with other people preferences is of basic importance for economic life. Examples of decision making are; deciding which customers will be visited by a travelling salesman and in which order, what goods and in what amount will be delivered to each of them; deciding which workers to assign to each stage of an assembly line; packing a set of items in a bag when taking a journey; scheduling the set of activities everyone has to deal with during the day. Even without realizing it, it is clear that decisions are made continuously during our daily life.

In general, these decisions are constrained. Let us suppose, for example, that we want to schedule the lectures given in a university. The final schedule must fulfill a number of requirements: the university has a limited number of rooms with a given number of seats; each lecture must be hold in a room having enough seats for students attending it; preparatory courses must be attended before other courses for propaedeutical reasons; lecturers may have preferences on the day of the week and on the part of the day where they have to teach.

These kind of problems are classified, in computer science, as optimization or feasibility problems. Finding the best solution, or simply a solution satisfying all the constraints involved in the problem, is hard. Modelling a problem is the aim of a huge part of computer science. Since 1960 the Operation Research (OR) community has worked on these problems introducing Linear Programming (LP) [28, 29], where problems are modelled using linear inequalities involving variables ranging over the real numbers. A solution to a LP problem is an assignment of a real value to each variable so as to satisfy all the inequalities. Often real decisions deal with discrete choices and disjunctive decisions; only an integer number can represent the amount of goods to be delivered; a stage of the assembly line can be assigned to worker A or to worker B, but not to both of them. These problems are called **combinatorial** and, to deal with them the OR community proposed the so called Integer Linear Programming (IP), where, beside the LP model, some variables are constrained to assume only integer values. This simple constraint turns a polynomial problem (as LP is) into a NP-Hard problem. In over fifty years the OR community proposed lots of techniques to solve the IP problems.

From the seventies, the Artificial Intelligence community started looking at combinatorial problems, defining the Constraint Satisfaction process. In the late eighties, the Constraint Programming (CP) paradigm was proposed. CP models a problem posting a set of constraints over a set of variables. Each variable is annotated with a domain of possible values it can assume. These values can be of any kind: real, integer or symbolic. A solution to a CP problem is an assignment of a value to each variable so as to satisfy all the constraints.

The same problems solved using an IP-based approach can be solved using a CP approach. From the literature, it is well known that, depending on the structure of the problem to be solved, one approach can be significantly better than another. The problem structure suggests the best solving technique and, for some classes of problems, the best approach can vary from an instance to another.

When modelling a real world problem it is of primary importance to choose the best solving technique. The first way one can try is to find some similarities between the problem faced and an existing problem for which it is well known, from the literature, the best solving approach: it is likely that the same approach can successfully solve both the problems. Often the difference between an appropriate solving technique and an unadvisable one lies not only in the solving time, but even in the capability to solve the problem or not.

Unfortunately, the theory is most often far from practice. It is not always possible to recognize similarities between different problems; furthermore, a very little difference can change so much the problem characteristics that the same solving approach has completely different behaviours. To cite an example, let us consider the well known Travelling Salesman Problem (TSP): considering complete methods, the TSP is best solved by an IP approach [3], and we can solve instances with up to some tens of thousands cities with IP [100], while we can hardly face problems with only 50 cities using a CP based approach. But when temporal constraints are introduced in the problem, facing the so called TSP with time windows (TSP-TW), CP becomes effective: in this case, a CP based approach, or an approach integrating CP and IP, can outperform a pure IP based approach [38, 39].

The problem structure can suggest the best solving approach, but for some classes of problems a single approach can be inadequate to solve them. The structure can suggest to combine together two different approaches, or to use one approach rather than another depending on the particular instance we are solving. In this dissertation evidence will be given, to support our thesis, that Constraint Programming and Integer Linear Programming are suitable programming paradigms for solving combinatorial optimization problems and, in particular, can be combined together to develop advanced solving tools for the classes of problems described above.

We will support the thesis analyzing two classes of combinatorial optimization problems: the Bid Evaluation Problem in Combinatorial Auctions, a generalization of the Winner Determination Problem with temporal constraints; the Allocation and Scheduling Problem in Multi Processor System-on-Chip platforms, considering two variants with and without voltage scaling. These problems will be described in Section 1.2. As we will see in Section 1.2, for the first class of problems the best approach depends on the single instance and not on the problem, while the second presents a structure where it is possible to recognize two well known sub-problems, the allocation and the scheduling, best solved respectively by an IP and a CP based approach. Trying different approaches for the former problem, and hybridizing the two techniques for the latter, can lead to a great enhancement of the solvers behaviours and quality.

1.2 Problem descriptions

In the following we will introduce the two problems analyzed during our research activity, describing the characteristics making them suitable for supporting our thesis.

1.2.1 Bid Evaluation Problem

The Bid Evaluation Problem (BEP) is a problem rising in the context of electronic auctions, in particular of Combinatorial Auctions (CA). In CAs the auctioneer puts up for auction a set of items at the same time and the bidders can bid on combination of them, indicating a price for the whole bundle of proposed items. In this context, the so called Winner Determination Problem (WDP) rises: given a set of items put up for auction and a set of bids, each proposing a price for buying (resp. selling) a subset of the items, the goal is to find the combination of winning bidders such that all the items are sold (resp. bought) at the maximum (resp. minimum) revenue (resp. cost). The WDP is NP-hard [70]. The BEP is a WDP where the items put up for auction are services that must be executed: each bidder associates to each item (service) appearing in its bid a temporal window inside which he can start supplying the service and the duration. The auctioneer imposes temporal precedences between couple of services. Let us think, for example, of a transportation service from city A to city C. When buying the service from A to B and from B to C, of course the former service, from A to B, must be supplied first. The problem is to find the set of bids covering all the services at the maximum (resp. minimum) revenue (resp. cost) such that all the temporal constraints (time windows and precedences) are met. In the Chapter 6 we will give a formal description of the WDP and the BEP.

For what introduced above, the BEP is a WDP with temporal side constraints. The side constraints can be seen as constraints added to a model that can describe, even without the side constraints, an optimization problem as well. Side constraints are, for example, temporal constraints, problem specific constraints, labor union rules.

The WDP is a well known problem, equivalent to a set partitioning problem, with a very clear structure, and, as we will see in Section 4.1.1, the best complete approach to solve the WDP is Integer Programming (IP). As soon as the temporal side constraints are added to the model, the structure is lost and it is no more straightforward that IP is the best solving strategy. The BEP can be efficiently solved by a Constraint Programming (CP) based approach. Depending on the instance structure, and in particular depending on the predomination of constraints coming from the covering or the temporal part of the problem, one approach (either IP or CP) can perform better than the other. This is exactly the class of problems where an Algorithm Portfolio (see Section 4.2) can be used.

In the second part of this dissertation we will analyze in deep the BEP structure developing a portfolio of algorithms, based on IP and CP, to solve the BEP. Our idea is to select the fastest algorithm in the portfolio by analyzing the instance structure before starting the search. If a way to automatically select the best algorithm can be found, it is possible, instead of running in parallel all the algorithms in the portfolio, to execute only the best one speeding up the search. The automatic method to select the best strategy we developed is based on Decision Trees, a Machine Learning approach, and on the off-line analysis of the instance structure.

This work has been subject of some publications: in [52] we proposed the different algorithms we developed to solve the BEP, defining the algorithm portfolio and describing a preliminary selection strategy; in [45] and [53] we exploited two different Machine Learning based algorithm selection techniques, namely Case-Based Reasoning (CBR) and Decision Trees (DT); in [44] we presented other selection techniques, namely Binomial Logistic Regression and Weighted Random, comparing them with CBR and DT and founding that DT provides the best results for the problem considered.

1.2.2 Allocation and Scheduling on a Multi-Processor System-on-Chip

The Multi-Processor System-on-Chip (MPSoC) is the state of the art for the system design technology. A set of homogeneous processors lie on the same silicon die, together with memories and an interconnection bus. When the MPSoC platforms are used to perform always the same application (or set of applications), an important design choice is to decide, for each process to be executed on top of the platform, which resources (memories and processors) to use. In fact, MPSoCs are typically used for realtime applications and finding the best allocation of resources to processes can lead to a significant cutting down of, for example, the power consumption or the inter-processors communication overhead, and, essential thing, to the compliance with the realtime constraints. Recent MPSoCs platforms can tune their speed changing the working frequency. Another degree of choice for the designer is therefore the working frequency of each process.

The Allocation and Scheduling problem (ASP) on a MPSoC is the problem of finding, given the characterization of the set of processes running on top of the platform, a feasible (or the optimal) allocation of processors and memories to processes, scheduling the process execution and respecting the capacity of all the resources (processors, memories and interconnection busses) as well as the realtime constraints. If we are interested in the optimal solution, the objective function could be the minimization of a time-related quantity as the makespan (the end of the last task), the tardiness (the delay of a process ending after the deadline), the late processes (the number of processes ending after the deadline). Another important objective function can be the minimization of the total amount of data transferred on the interconnection bus. The bus is a shared resource, and when a number of processes try to use it at the same time, collisions may occur and therefore a bus arbitrage mechanism is needed. The execution time of the processes becomes thus higher due to the arbitrage overhead and, if the number of collision becomes considerably higher, the real time constraints can be violated. Minimizing the total amount of data transferred on the shared bus reduces the collision probability.

If we consider a MPSoC able to change the working frequency, besides the resource allocation, the assignment of a frequency to each process is another degree of choice. This problem is called the Dynamic Voltage Scaling Problem (DVSP). In the DVSP, one might be interested in minimizing the total power consumed; in fact, MPSoCs can be embedded in mobile devices, where the power consumption reduction is the main issue.

Differently from the BEP, the structure of this problem is almost the same for every instances, but presents a very interesting characteristic: we can recognize two sub-problems, the allocation of resources, and eventually frequencies, to tasks and the scheduling of the tasks execution using the allocated resources over the time. The allocation problem is an optimization problem and is best solved by an IP approach, while the scheduling, dealing with temporal constraints, is best faced by a CP approach. Solving the overall problem using a single solving technique (either IP or CP) is very inefficient, but, as the structure suggests, it is possible to split the two sub-problems and to solve them separately using the most appropriate technique for each sub-problem.

In the third part of this dissertation we will analyze and solve the problem, focussing in particular on the questions connected with the problem splitting and with the interaction between the sub-problems. In fact, the allocation and the scheduling part are, in general, linked together sharing some constraint and hence they must interact in order to find the optimal solution for the problem overall. We exploit the Logic-Based Benders Decomposition technique [56] to make the two sub-problems cooperating.

This work has been subject of some publications: in [13] and [14] we proposed a Logic-Based Benders Decomposition approach to solve the ASP, comparing it with pure CP and IP based approaches; in [113] we validated the approach simulating our solutions on a real MPSoC platform. In [15] we proposed a Logic-Based Benders Decomposition approach for the DVSP, validating it in [16].

1.3 Organization of the Dissertation

This dissertation is divided into three parts, organized as follows.

Part I - Introduction & Preliminaries

Chapter 1, Introduction: we introduce our research and we briefly describe the problems we face in the dissertation.

Chapter 2, Constraint Programming: we report the background knowledge on Constraint Programming necessary to read the dissertation. We present constraint satisfaction and optimization problems, some techniques and algorithms to solve them, ordering heuristics and global constraints.

Chapter 3, Integer Linear Programming: we report the background knowledge on Integer Linear Programming necessary to read the dissertation. We present linear programming, the duality concept, integer programming and techniques to solve these problems, with particular stress on decomposition methods.

Chapter 4, Integration of Constraint and Integer Linear Programming: we summarize Constraint Programming and Integer Linear Programming strong and weak points and we introduce the integration techniques used.

Part II - The Bid Evaluation Problem in Combinatorial Auctions

Chapter 5, Introduction: we introduce some auction mechanisms and the combinatorial auctions.

Chapter 6, Problem description and modelling: we describe the BEP and we introduce the CP and IP models for the problem. We discuss on previous works related to the arguments of the Part II.

Chapter 7, Algorithms and Experimental Results: we introduce the solving tools we developed and we show the experimental results obtained. We compare our tools with another BEP solving tool.

Chapter 8, Algorithm Portfolio Analysis: we discuss and present the algorithm portfolio we developed and the machine learning tool we used to select the best algorithm in the portfolio on the basis of the instance structure. We show some experimental results.

Part III - The Allocation and Scheduling Problem on a Multi-Processor System-on-Chip

Chapter 9, Introduction: we introduce the MPSoC platform and simulator and we describe the problems faced in the Part III. We conclude discussing on previous works related to the arguments of the Part III.

Chapter 10, ASP model: we describe the ASP problem and its model based on decomposition. We discuss about design choices and simplifying assumptions.

Chapter 11, ASP Results: we compare the results obtained when modelling and solving the ASP using hybrid or pure approaches. We validate the efficiency of our solving tool and the executability of the solutions found by comparing our results with those found by the MPSoC simulator.

Chapter 12, DVSP model: we describe the DVSP problem and its model based on decomposition. We discuss about design choices and simplifying assumptions.

Chapter 13, DVSP Results: we compare the results obtained when modelling and solving the DVSP using hybrid or pure approaches. We validate the efficiency or our solving tool and the executability of the solutions found by comparing our results with those found by the MPSoC simulator.

Chapter 14, Conclusions and future works: we conclude the dissertation presenting our contribution, discussing on lessons learnt, strong and weak points of our work and presenting some future extension and lines of research.

Chapter 2

Constraint Programming

Introduction

In this chapter a formal background on Constraint Programming will be given. In section 2.1 Constraint Satisfaction and Optimization Problems will be defined; in sections 2.2 and 2.3 the concepts and techniques required to solve a Constraint Satisfaction Problem and a Constrained Optimization Problem will be described; section 2.4 introduces the Global Constraints.

2.1 Constraint Satisfaction Problem

In a Constraint Satisfaction Problem (CSP) we have a set of variables, each with a domain of possible values, and a set of constraints involving a subset of the variables. A constraint is a relation between some variables limiting the set of values the variables can assume. A constraint can involve any number of variables. A solution to a CSP is an assignment of one value to all the variables such that each constraint is met.

More formally:

Definition 1 *A Constraint Satisfaction Problem (CSP) consists of:*

- *a set of variables* $X = \{X_1 \dots X_n\}$*;*
- *a set of finite domains of values* $D = \{D_{X_1} \dots D_{X_n}\}$*, one for each variable* X_i*. The cartesian product of all the domains* $D_{X_1} \times D_{X_2} \times \cdots \times D_{X_{n-1}} \times D_{X_n}$ *is called* **search space***;*

- *a set C of constraints imposed on the variables. Each constraint $c_i(X_1^i \dots X_j^i) \in C$, imposed on a subset $X_1 \dots X_j$ of the variables X, defines a subset of the domains $D_{X_1} \dots D_{X_j}$ containing only the combination of values allowed.*

A CSP is thus described by a tuple $\{X, D, C\}$. When a variable X_i assumes a value in D_{X_i} we have an **assignment**. If all variables are assigned, we have a **total assignment** and, if all the constraints are met, a solution to the CSP is found.

If we are not only interested in whatever solution, but we want to find the best solution w.r.t. a given objective function, the problem is called Constraint Optimization Problem (COP).

Solving a CSP (or a COP) is usually NP-Hard, that is it does not exist (unless P=NP) a method to solve the problem in a time polynomial in the size of the problem. A CSP is intractable, is thus necessary to define search strategies to prune the search space and to reach a solution in a reasonable time.

2.2 Search, Consistency and Constraint Propagation

A CSP conceptually uses a search tree, i.e. a tree where a node represents a variable, an edge starting from a node represents an assignment and each leaf is a total assignment. If the total assignment meets all the constraints, the leaf represents a solution to the problem.

The simplest technique to solve a CSP is to perform a complete assignment traversing the search tree from the root node to a leaf, and checking *a posteriori* if the assignment is feasible and, thus, is a solution. If it is unfeasible, the search proceeds ascending the tree (this technique is known as **backtracking**) until the first node with an alternative branch is found, hence repeating the search on unexplored paths. This technique is called **Generate and Test**: a solution is generated and then tested for the feasibility. It is easy to understand that this technique, in the worst case, traverses the whole search tree enumerating and testing all the possible total assignments.

Another technique, slightly better than Generate and Test, is the **Standard Backtracking**: each time an assignment is performed, the compatibility with all the other assignments done so far is checked, avoiding to traverse a path that will surely lead to a failure.

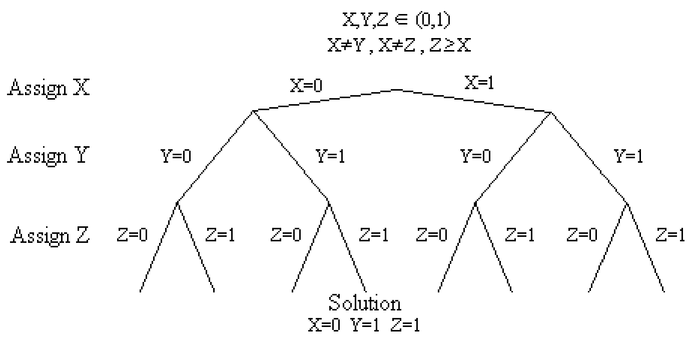


Figure 2.1: Example of search tree

It is possible to use the constraints in a smart fashion, in order to reduce the search tree. Figure 2.1 represents a small example of a search tree for a problem with 3 variables (X, Y, Z) with domains $\{0, 1\}$ and 3 constraints $(X \neq Y, X \neq Z, Z \geq X)$. Considering this small example, we can easily understand that Generate and Test technique will backtrack 4 times before finding a solution, while Standard Backtracking will backtrack only twice. We can note that, if after the assignment $X = 0$ the value 0 from the domains of variables Y and Z would be immediately removed, since they are both incompatible with the constraints imposed, the solution could be found without backtracking.

In practice, no CP solvers implement the techniques described above, but more efficient techniques able to prevent failures removing the infeasible values as soon as possible and thus exploring a lower portion of the search tree. These techniques are called **Consistency** and **Constraint Propagation Techniques**.

2.2.1 Consistency Techniques

The Consistency Techniques (CT) propagate the constraints before starting the search, removing from the domains those values that will not lead to any feasible solution. CT therefore derive a smaller, and thus simpler, problem from the original one.

Conceptually, the CT are based on a constraint graph: each node represents a variable and each arc represents a constraint. Arcs can be directed or not, e.g. the constraint $\neq$ is represented by a bidirectional (or not directed) arc, while the constraint $\leq$ is represented by a directed arc. Unary constraints (e.g. $0 \leq X \leq 10$) are represented by an arc starting and ending in the same node, while binary constraints (e.g. $X \geq Y$) are represented by an arc joining 2 nodes. The simplest level of consistency, namely the Node Consistency or consistency with degree 1, is obtained when all the values in a variable domain are consistent with the unary constraints involving the variable.

Starting from a constraint graph, several degrees of consistency are reached by the commercial CP solvers.

- **Arc Consistency (AC)**: consistency with degree 2. A constraint graph is arc consistent if it is node consistent and all the arcs in the graph are consistent: an arc between 2 nodes (i.e. a constraint between 2 variables) is consistent if, for each value in the domain of a variable, exists at least one value in the domain of the other variable satisfying the constraint. If it is not the case, the value must be removed from the domain of the first variable. Removal due to AC check can lead to a graph that is no more node consistent; thus node consistency must be checked again, as well as AC, iterating the analysis until the graph converges to a stable node consistent state.
- **Bound Consistency (BC)**: is a relaxation of the AC. BC enforces AC only on the outer bounds of the variables domains. BC was originally proposed for continuous domains and then extended to discrete ones with the only requirements that the domains must be ordered. Achieving BC is less time and space consuming than AC.
- **Generalized Arc Consistency (GAC)**: is a generalization of the AC to deal with constraints of arity higher than 2. A constraint $c(X_1, \dots, X_n)$ is GAC if, taken a variable X_i, for each assignment to the remaining $n-1$ variables

  $$X_1 = v_1,\ \dots,\ X_{i-1} = v_{i-1},\ X_{i+1} = v_{i+1},\ \dots,\ X_n = v_n$$

  exists a value d in the domain of X_i such that the constraint

  $$c(v_1, \dots, v_{n-1}, d, v_{n+1}, \dots, v_n)$$

  is satisfied.

2.2.2 Constraint Propagation Techniques

Constraint Propagation Techniques remove from the variable domains those values that, by virtue of the partial assignments done so far, will lead to a failure. Typically, in a CSP, consistency is first checked, then propagation is performed and, if a solution is still not found, an assignment is tried. Consistency, propagation and assignments iterate until a solution is found or a failure occurs: in the latter case, backtracking is performed.

Among the Constraint Propagation Technique, Forward Checking (FC) and Look Ahead (LA) are the most used. FC removes, from the domains of all the variables till not assigned, all the values incompatible with the last assignment done. If a domain becomes empty, backtracking is performed. LA performs FC and, besides, checks if the values left in the domains can still lead to a solution or, due to the assignments and propagation done they will lead to a failure. LA achieves the AC.

The main difference between FC and LA is that FC propagates only the constraints involving the last variable assigned, while LA also propagates all the constraints involving at least one unassigned variable. It is easy to understand that LA is more powerful than FC to avoid backtracks but, on the other side, it requires an higher computational effort: it is necessary to find the technique that supplies the best tradeoff between search space reduction and computational effort.

2.3 Constraint Optimization Problem

2.3.1 Objective functions

As introduced in Section 2.1, a Constraint Optimization Problem (COP) is a CSP with an objective function. An objective function is a function of the variables that must be minimized, or maximized. In a COP we are interest in finding the optimal solution, the solution with the minimum (or maximum) value of the objective function. In a COP, objective functions are handled in a naive way: the first solution is found, the objective function value is retrieved and a new constraint in the problem is imposed stating that the new solution objective function must be lower (or greater) than the current one. The constraint is updated each time a solution is found and, when the problem becomes infeasible, the last solution found is the optimal one.

2.3.2 Branch and bound

Branch and bound (B&B) is a general method for finding, in a reasonable time, optimal solutions for various optimization problems, especially for COPs. It belongs to the class of implicit enumeration methods and was first proposed in [32] for linear programming. The general idea is to apply a procedure that requires two phases (Branching and Bounding).

Branching is a smart way of splitting the problem in several smaller sub-problems. Each time an assignment is done in a node of the tree, an edge starting from the node is traversed (branch) and the underlying sub-tree is explored. If the procedure is repeated recursively for each possible edge of each node in the tree, the union of all the subtrees partitions the original search space.

Bounding is a way of finding an upper (lower) bound for the optimal solution within a subtree. The core of the approach is a simple observation that, for a minimization problem, if the lower bound for a subtree A of the search tree is greater than the solution for any other previously examined subtree B (or if it is greater than a solution already found), then A may be safely discarded from the search. This step is called pruning. It is usually implemented by maintaining a global variable m that records the minimum solution found among all subtrees examined so far; any node whose lower bound is greater than m can be discarded.

Ideally the procedure stops when all nodes of the search tree are traversed, but it is possible to terminate the procedure after a given time; at that point, an incomplete search is performed and the best solution found is returned.

The efficiency of the method depends critically on the effectiveness of the B&B algorithm used; bad choices could lead to repeated branching, without any pruning, until the subtrees become very small. In that case the method would be reduced to an exhaustive enumeration of the domain, which is often impractically large. B&B methods may be classified according to the bounding methods and according to the ways of creating/inspecting the search tree nodes: the way of creating a search tree depends on the variables and variable values selection heuristics, while the way of inspecting the tree depends on the search algorithms.

2.3.3 Variables and variable values selection heuristics

The B&B technique can avoid many parts of the search space, but it is of the utmost importance to create the search tree in the "right" way in order to have tight bounds up at the first nodes of the tree. To create a tree, nodes and edges must be ordered: a node represents a variable to be assigned, while an edge starting from a node represents a trial value for the variable in the node. To order a tree is therefore sufficient to decide the order in which the variables will be assigned, as well as the order in which the values in the domain of a variable will be tried. This ordering is done using the **variables** and **variable values selection heuristics**.

Heuristics can be static or dynamic: static heuristics order the search tree before starting the search and the order remains the same over all the search. Dynamic heuristics choose, at each node of the search tree, which is the best edge to branch on depending on the status of the search. A dynamic heuristic is, obviously, better w.r.t. a static one, but finding the perfect heuristic, that is the heuristic that always suggests the right choice, has the same complexity as the original problem: it is therefore important (as usual) to find the right tradeoff between heuristic goodness and complexity.

It is very important to choose a good heuristic, that reflects on the number of nodes that will be visited. In fact, when dealing with a CSP, a heuristic that reliably estimates the distance of a node from a solution allows us to select, at each node, the shortest way to a feasible solution. Similarly, when dealing with a COP, finding a feasible solution early in the search allows us to find an upper (or lower) bound for the objective function and thus to remove some parts of the search tree.

Among the heuristics usually considered, we can find:

- **First Fail Principle** (FFP), tries to solve first the harder subproblems, those that are likely to lead to a failure;

- **Least Constraining Principle** (LCP), tries to choose first the least constraining paths, those paths that propagate less on the variable domains;

- **Most Constraining Principle** (MCP), opposite to LCP, tries to choose first the most constraining paths, those paths that propagate more on the variable domains.

Heuristics are used to order the tree by selecting, at each node, a variable and a trial value for the variable. Among the most common variables selection heuristic principles, are used the FFP, that tries to assign first the variables whose domains have the lower cardinality, or the MCP, that tries to assign first the variables involved in a greater number of constraints. These principles tries to assign first the variables that are likely to lead to a failure. As variable values selection heuristic is usually used the LCP, the values with the higher probability to appear in a feasible solution are tried first. Besides, problem-dependent heuristics can be defined, either specializing the common heuristics adapting them to the problem characteristics or creating a new one.

2.3.4 Tree Search algorithms

Even though smart selection heuristics can order the tree in such a way that the search will lead directly (or after a small number of backtracking) to a solution, it is also important, when a failure occurs, to decide where to go on with the search. Search algorithms define the way the search tree will be inspected, in particular when, in backtracking, a node is re-explored.

The search algorithms differ in their completeness or incompleteness. A complete algorithm explores the whole search space that is not pruned by propagation, while an incomplete one explores only a portion of the space and the best solution found in that portion is returned; the higher the probability for the solution to be the optimal one (or within a given percentage w.r.t. the optimum), the better the incomplete algorithm is. While some algorithms, for example Local Search, can typically perform only an incomplete search, some others can perform both a complete and incomplete search, depending on limitations imposed, such as maximum number of nodes explored or maximum depth of a search path.

A great number of search algorithms exists in literature. In the following some of them, based on tree search, will be explained in detail.

Depth First Search

Depth First Search (DFS) is the simplest search algorithm. The search is performed in depth, i.e. at each node a variable is bound and an edge starting from the node is traversed until a solution is found or a failure occurs. Ascending the tree in backtracking, as soon as a node with at least one unexplored edge is reached, the left-most unexplored edge is traversed. This algorithm visits the leaves of the tree tidily from the left-most to the right-most. Considering Figure 2.2, where a simple binary tree is depicted and leaves are labelled, DFS will visit the leaves in the lexicographic order.

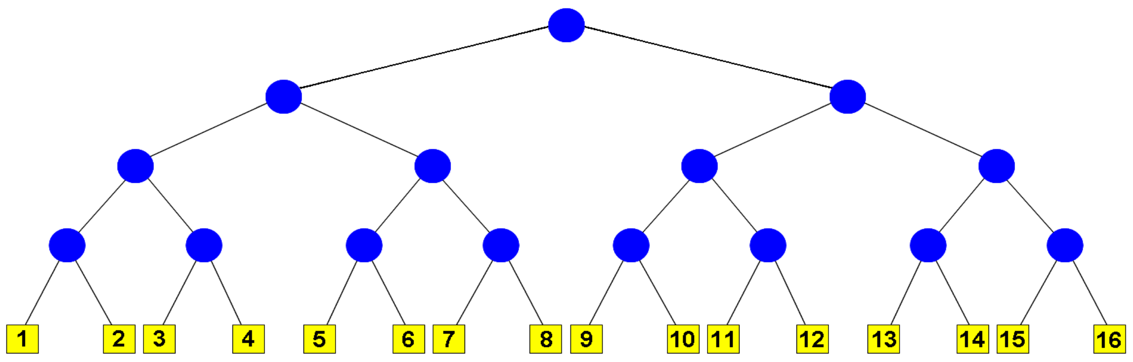


Figure 2.2: Example of search tree

Limited Discrepancy Search

Proposed in [48], Limited Discrepancy Search (LDS) explores the tree allowing, at each search iteration, up to a given number of discrepancies in the path. A discrepancy is a branch where the value selection heuristic is not followed; in Figure 2.2, if the left branch represents the value suggested by the heuristic, each right branch is a discrepancy. LDS starts searching for the solution with 0 discrepancies, that is the left-most leaf, then searches for all the solutions with 1 discrepancy, and so on increasing the maximum number of allowed discrepancies at each iteration. LDS traverses the tree as DFS but prunes, at each iterations, all the paths with a number of discrepancies higher than the maximum allowed. In Figure 2.2 leaves are explored in the following order: $1, 2, 3, 5, 9, 4, 6, 7, 10, 11, 13, 8, 12, 14, 15, 16$. Considering a non-binary tree, if all the edges are ordered from left to right according to the values selection heuristic, the discrepancy of each branch is the ranking of the edge among its siblings.

LDS can be implemented as an incomplete algorithm by limiting the maximum number of discrepancies allowed.

A problem when implementing LDS is that, in backtracking, some nodes are explored more than once. Furthermore, LDS does not discriminate discrepancies on the basis of the level of the tree where they occurs. To eliminate these limitations, LDS has been enhanced by Improved LDS (ILDS) [76] and Depth-bounded Discrepancy Search (DDS) [122].

Discrepancy-Bounded Depth First Search

Proposed in [7], Discrepancy-Bounded DFS (DB-DFS) is strongly based on LDS but performs an important integration: LDS explores the search tree using DFS increasing the maximum number of allowed discrepancies by one unit at a time, while DB-DFS increases it by k units at a time. k is called *discrepancy step*. So, given a value k, at iteration $i = [0, \ldots, n]$ DB-DFS explores all the nodes with a number of discrepancies between $i * k$ and $(i + 1)k - 1$ inclusive. If $k = 1$, we obtain LDS. In Figure 2.2, with $k = 2$ the leaves are explored in the following order: $1, 2, 3, 5, 9, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16$. DB-DFS is also known as **Slice Based Search (SBS)**.

2.4 Global Constraints

A huge impulse to CP is certainly due to Global Constraints (GC). GC were introduced in [9] and allow to express a symbolic constraint over a set of variables. The power of the GC is twofold: declaratively, they can express the constraint concisely; operationally, they encapsulate a specialized global propagation technique with a complexity polynomial in space and time, that is therefore very efficient. As an example, we will describe here three GC.

- **AllDifferent**: with arity 1. The argument is a list of variables: the constraint states that all the variables in the list must have different values one each other. This constraint is much more concise than imposing a $\neq$ constraint for each pair of variables in the list and the encapsulated propagation technique can remove much more infeasible values. Let us consider, for example, the constraint $AllDifferent([X_1, X_2, X_3, X_4])$, where the variables X_1, X_2 and X_3 have the same domain $[1, 2, 3]$, while X_4 has the domain $[1, 2, 3, 4]$; it is possible to deduce that X_4 must necessary be equal to 4. The same reasoning could not be done if the $\neq$ binary constraints were used. In [108] the author proposes a polynomial algorithm based on the flow theory able to achieve the Generalized

Arc Consistency for the AllDifferent global constraint. The space complexity is $O(nd)$ and the time complexity is $O(n^2d^2)$, where n is cardinality of the list of variables and d is the maximum cardinality of the variable domains.

- **Global Cardinality Constraint**: [109] with arity 4. The global cardinality constraint is $gcc(Var, Val, LB, UB)$, where Var is a list of variables, Val a list of values, LB and UB two lists of values. The gcc constrains the number of occurrences of each value Val_i among the variables Var to be within the interval $[LB_i \dots UB_i]$. In other words, the gcc limits the number of occurrences of a list of values among a list of constrained variables. Let consider 5 variables $X_1 = [1, 2]$, $X_2 = [1, 2, 3]$, $X_3 = [1, 2, 3]$, $X_4 = [3, 4]$, $X_5 = [4]$, and the constraint gcc(X,[1,2,3,4],[0,0,2,2],[2,2,3,2]). The constraint forces values 1 and 2 to appear at most twice among the variables X; value 3 can appear at least twice and at most three times, while value 4 must appear exactly twice (in fact the lower and the upper bound for value 4 are equal). Since value 4 must appear twice, both X_4 and X_5 must take the value 4. Value 3 can now appear at most twice, and this complies with the constraint: thus $X_2 = X_3 = 3$. Finally, X_1 can assume both the values 1 and 2, so the example has two feasible solutions ($X_1 = 1$, $X_2 = 3$, $X_3 = 3$, $X_4 = 4$, $X_5 = 4$) and ($X_1 = 2$, $X_2 = 3$, $X_3 = 3$, $X_4 = 4$, $X_5 = 4$). Figure 2.3 is a bipartite graph where the higher nodes represent the variables and the lower nodes represent the values. The graph depicts the gcc constraint, the constraint propagation and a feasible solution. Each line represents an assignment: the dotted lines are pruned by propagation while the bold lines represent a feasible solution. In [108] the author proposes a polynomial algorithm based on the flow theory able to achieve the Generalized Arc Consistency for the Global Cardinality Constraint. The space complexity is $O(nd)$ and the time complexity is $O(n^2d)$, where n is cardinality of the $Vars$ list and d is the cardinality of the Val list.

- **Cumulative**: with arity 4. The first three arguments are lists of n values representing, respectively, the start time, the duration and the resource requirement of n different activities. The last argument is a value and represents the maximum quantity of resource available (this value can vary over the time). The constraint tries to schedule the activities so as to meet the

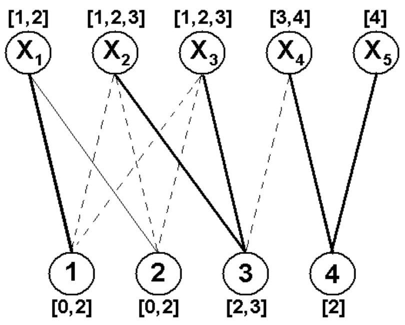


Figure 2.3: Example of the Global Cardinality Constraint

constraint on the resource availability. Let us consider four activities $A_1 \ldots A_4$ having duration 5, 4, 3 and 4 and needing respectively 3, 2, 4 and 2 units of a resource having maximum availability equal to 8. If we want to schedule these activities finding each starting time, we can use the Cumulative global constraint: $cumulative([X_1, X_2, X_3, X_4], [5, 4, 3, 4], [3, 2, 4, 2], 8)$. Variables X represent the starting times; a feasible assignment is $X = [0, 3, 0, 3]$ and is depicted in Figure 2.4. Several algorithms exist to propagate the Cumulative constraints: Time-Table constraint, Not-First Not-Last constraint, Edge Finding constraint [6] to cite few, based on energetic reasonings and on the obligatory parts of a schedule. These techniques can reduce the variable domains, but can not achieve the Generalized Arc Consistency in polynomial time.

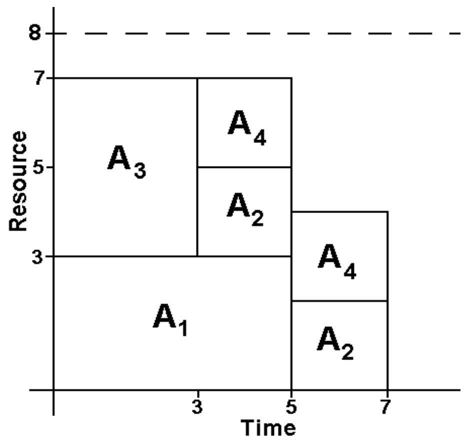


Figure 2.4: Example of the Cumulative global constraint

Chapter 3

Integer Linear Programming

Introduction

In this chapter a formal background on Integer Linear Programming will be given. In section 3.1 Linear Programming will be defined and in sections 3.2 and 3.3 we will focus on two advanced methods in the context of Linear Programming, respectively the duality theory and the decomposition methods. Section 3.4 introduces Integer Programming.

3.1 Linear Programming

Linear programming (LP) is an important field of optimization. Many practical problems in Operations Research can be expressed as linear programming problems. The standard form of an LP problem is:

$$min : c^T x \tag{3.1a}$$

$$s.t. : Ax \geq b \tag{3.1b}$$

$$x \geq 0 \tag{3.1c}$$

where $x \in \Re^n$, $c \in \Re^n$, $b \in \Re^n$ and $A \in \Re^{m \times n}$. (3.1a) is the Objective Function (OF), a linear function of the variables x that must be minimized and (3.1b) are the linear constraints imposed on the variables x.

We can graphically explain a LP problem by an example. Suppose we have the

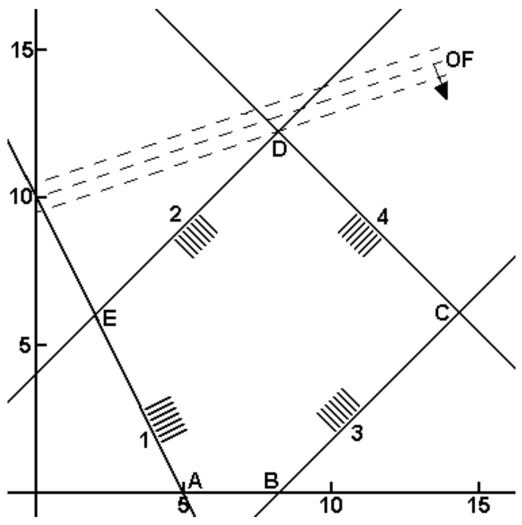


Figure 3.1: Example of a LP problem in the Cartesian plane

following LP problem involving two variables:

$$min : x_1 - 3x_2 \tag{3.2a}$$

$$4x_1 + 2x_2 \geq 20 \tag{3.2b}$$

$$x_1 - x_2 \geq -4 \tag{3.2c}$$

$$- x_1 + x_2 \geq -8 \tag{3.2d}$$

$$- x_1 - x_2 \geq -20.5 \tag{3.2e}$$

$$x \geq 0 \tag{3.2f}$$

Figure 3.1 depicts the example in the Cartesian plane. Each inequality in the model defines a hyperplane. The four lines labelled from 1 to 4 define respectively the inequalities (3.2*b*), (3.2*c*), (3.2*d*) and (3.2*e*). Each inequality is fulfilled in the hyperplane denoted by the short parallel segments. The intersection of all the hyperplanes defines a convex polytope; the lower part is limited by the constraint (3.2*f*), forcing the polytope to lie only in the first quadrant of the Cartesian plane. The dotted parallel lines represent the OF (3.2*a*). These are isocost lines defined by the equations $x_1 - 3x_2 = K$, where K is a constant. The arrow denotes the rise direction of the OF value associated to each isocost line. The optimal solution is the point of the polytope where the OF value is minimum; since only one isocost line intersects each point in the Cartesian plane, the optimal solution is the point in the polytope where the value of K is minimum. For the example, the optimal solution is represented by the vertex $D = (8.25, 12.25)$.

The Minkowski-Weyl theorem ensures that, if a LP model defines a convex polytope, the optimal solution always lies in a vertex [111]. Some particular or degenerate cases can happen:

- If the OF isocost lines are parallel to a side of the convex polytope, all the points of the side have the same OF value and, if it is the minimum one, the problem has infinite optimal solutions. See Figure 3.2a.
- If the polytope is not closed, the problem does not have a finite solution and the problem is called **unbounded**. See Figure 3.2b.
- If the semiplanes associated to the inequalities do not define any convex polytope, the problem does not have a feasible solution and is called **infeasible**. See Figure 3.2c.

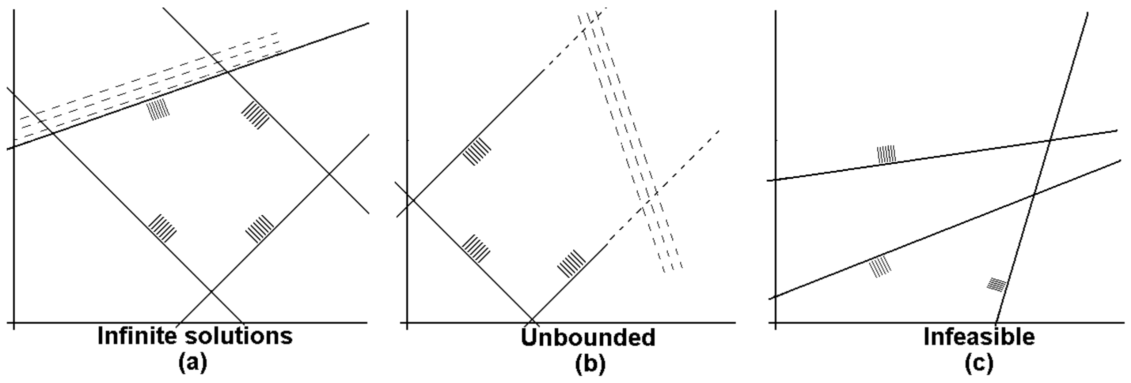


Figure 3.2: LP problem degenerate cases

To solve a LP problem, in the 1947 George B. Dantzig developed the simplex method [28, 29], with a complexity exponential in the size of the problem in the worst case, but very efficient in practice. In 1979, Leonid G. Khachiyan [71] proposed a polynomial method to solve a LP problem based on a previous method proposed by Naum Shor, namely the ellipsoid method in nonlinear optimization. Even if this method was polynomial, it was inefficient in practice due to the high degree of the polynomial and its performances was worse w.r.t. the simplex algorithm, but the important thing was that this method opened up an interest in finding new polynomial algorithms to solve LP problems. In 1984 the so called Karmarkar's algorithm, polynomial in the worst case, was proposed in [69].

Even if there have been attempts for faster or specialized algorithms, the simplex method is still used to solve LP problems. It is not the aim of this dissertation to enter into detail of the simplex algorithm, we only need to say that among the methods connected or derived from LP the most important are duality theory and decomposition methods, for sure the driving forces behind the success of the simplex method.

3.2 Duality Theory

Given the primal LP problem described by (3.1), the corresponding dual LP is given by:

$$max : \lambda^T b \tag{3.3a}$$

$$s.t. : \lambda^T A \leq c^T \tag{3.3b}$$

$$\lambda \geq 0 \tag{3.3c}$$

where $\lambda \in \Re^m$. Similarly to the primal model, (3.3a) is the Objective Function and (3.3b) are the linear constraints. The duality theory has three important properties:

Theorem 1 (Symmetry) *The dual of the dual is the primal problem.*

Theorem 2 (Strong duality) *If any of the primal or dual has a finite optimal solution, so does the other and both have the same objective function value.*

If the primal is unbounded or infeasible, the Strong duality theorem can not be applied and it is therefore useful the following theorem:

Theorem 3 (Weak duality) *If x and λ are feasible solutions for the primal and dual respectively, then $c^T x \geq \lambda^T b$.*

From theorem 3 we can argue that the OF of the dual solution is a lower bound for the OF of any feasible primal solution. Viceversa, the OF of the primal solution is an upper bound for any feasible dual solution. From theorems 2 and 3 it follows:

Corollary 1 *When solving the primal and the corresponding dual LP problems, only the following cases can happen:*

(i) Both the problems have a finite optimal solution. If x^* and λ^* are feasible solutions for the primal and the dual respectively, and if $c^T x^* = \lambda^{*T} b$, then x^* and λ^* are the optimal solutions.

(ii) The primal is unbounded and the dual is infeasible.

(iii) The dual is unbounded and the primal is infeasible.

(iv) Both the primal and the dual are infeasible.

Starting from the solution of the dual we can extract the so called **reduced costs** associated to variables x. The reduced cost of a variable is the minimum change in the OF if the variable value is increased by one unit in the current solution. The formula to extract the reduced cost rc_i associated to the variable x_i is $rc_i = c_i - \lambda^{*T} A_i$, where λ^* is the optimal dual solution.

3.3 Decomposition methods

The underlying idea of a decomposition method is to split the problem variables in two disjoint subsets and to solve the two subproblems separately. In the context of LP, the most famous decomposition method is the so called Benders Decomposition [11], presented in 1962. Benders Decomposition (BD) applies to problems in which the subproblem is linear:

$$min : c^T x + f(y) \tag{3.4a}$$

$$s.t. : Ax + F(y) \geq b \tag{3.4b}$$

$$x \geq 0 \tag{3.4c}$$

$$y \in Y \tag{3.4d}$$

The constraints involving the variables x define the LP subproblem, while those involving the variables y can be of any kind and defines the so called master problem. The BD technique fixes the variables y to values $\bar{y}$ compatible with the master problem constraints, then solves to optimality the following LP sub-problem containing only the variables x (being variables y fixed to the trial values):

$$min : c^T x + f(\bar{y}) \tag{3.5a}$$

$$s.t. : Ax \geq b - F(\bar{y}) \tag{3.5b}$$

$$x \geq 0 \tag{3.5c}$$

From the solution of the dual problem:

$$max : \lambda^T (b - F(\bar{y})) + f(\bar{y}) \tag{3.6a}$$

$$s.t. : \lambda^T A \leq c^T \tag{3.6b}$$

$$\lambda \geq 0 \tag{3.6c}$$

we obtain a lower bound for the OF when $y = \bar{y}$. It is provable that the same lower bound remains valid for any y.

If the dual has a finite solution $\bar{\lambda}$, we have the valid lower bound $\bar{\lambda}(b - F(y)) + f(y)$ for the OF. If the primal is infeasible we have the valid inequality $\bar{\lambda}(b - F(y)) \leq 0$. These inequalities are called Benders cut and are added to the master problem model. The process iteratively solves the master problem finding the values $\bar{y}$, then solves the subproblem fixing the variables y to $\bar{y}$ finding a lower bound for the OF and adding the Benders cut to the master problem. The process converges to the optimal solution for the original problem when the master problem OF equals the last bound found. Degenerate cases can happen: if the master problem is infeasible, the original problem is infeasible; if the subproblem dual is infeasible, the original problem is unbounded.

The BD technique is based on the work of Dantzig and Wolfe [30], where the original problem is a LP problem as well, having the property that can be decomposed in two LP subproblems tied together by a smaller number of constraints w.r.t. those imposed on the original problem.

BD can therefore be seen as a generalization of the Dantzig-Wolfe method where the master problem can be of any kind. Hooker [56] applied BD to problems where the subproblem as well can be of any kind. These method is called Logic-Based Benders Decomposition (LB-BD); in Chapter 9.4 we will describe LB-BD in detail.

3.4 Integer Programming

An Integer Programming (IP) problem is defined as follows:

$$min : c^T x \tag{3.7a}$$
$$s.t. : Ax \geq b \tag{3.7b}$$
$$x \geq 0 \tag{3.7c}$$
$$x \text{ integer} \tag{3.7d}$$

We can easily see that an IP problem is an LP problem (3.1) augmented with the integrality constraints ($3.7d$) forcing all the variables to assume only integer values. The integrality constraints are non-linear; they can be mathematically expressed as $sin(\pi x_i) = 0$, $\forall i$. If the integrality constraints involve only a subset of the x variables, the problem is called Mixed Integer Linear Programming (MILP).

Many real life problems are easily modelled in IP using the so called decision variables, i.e. variables that can assume only the values 0 and 1. Decision variables are usually associated to choices: if a variable is equal to 1, than the corresponding option is chosen.

By removing the integrality constraint ($3.7d$) we obtain the so called Linear Relaxation (LR) of the IP problem. The LR is a LP problem.

Let us consider the same example of Section 3.1 augmented with the integrality constraints.

$$min : x_1 - 3x_2 \tag{3.8a}$$
$$4x_1 + 2x_2 \geq 20 \tag{3.8b}$$
$$x_1 - x_2 \geq -4 \tag{3.8c}$$
$$-x_1 + x_2 \geq -8 \tag{3.8d}$$
$$-x_1 - x_2 \geq -20.5 \tag{3.8e}$$
$$x \geq 0 \tag{3.8f}$$
$$x \text{ integer} \tag{3.8g}$$

The graphical representation of the example is shown in Figure 3.3. It represents the same polytope, and in addition all the integer points inside the polytope are depicted. The optimal solution is now the integer point with the lower OF value. Following the same line of reasoning used for the LP example 3.2, we can see that the optimal solution is represented by the point $P = (8, 12)$. The solution of the LR, in this case $(8.25, 12.25)$ (see the example (3.2), in general is not a valid solution for the IP problem because it may violate some integrality constraints, but nevertheless it can be used to bound the IP problem OF using branch and bound.

The optimal solution found for the LP problem $(8.25, 12.25)$ is a super-optimal solution for the IP problem.

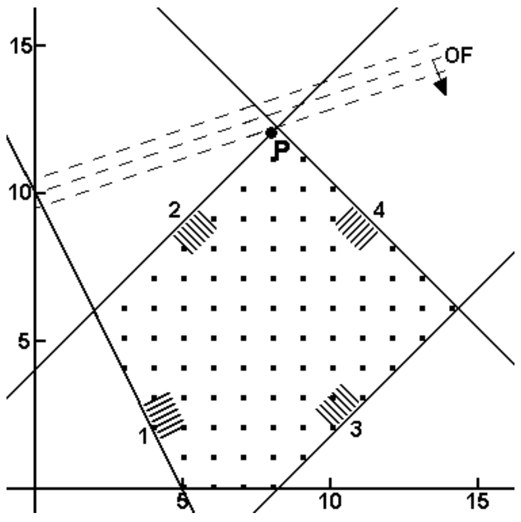


Figure 3.3: Example of an IP problem

Due to the presence of the integrality constraints, that are non-linear constraints, it is not possible to use the simplex algorithm (or any another method developed for the LP) to solve an IP or a MILP problem. Solving an IP problem is, in the general case, NP-hard [70] and several advanced algorithms have been developed to solve an IP problem, the most important being Branch and Bound.

3.4.1 Branch and Bound

Branch and Bound (B&B) is based on the LR of an IP problem. The B&B scheme interleaves two steps: solving the linear relaxation and branching. When the LR is solved, if the OF value is worse than the best solution found for the original problem, then we can stop searching because the LR solution is a bound for the IP problem OF. If it is not the case, than a variable x_i with a non-integer value v is chosen and the problem is split in two subproblems; the first subproblem contains the original model plus the constraint $x_i \leq \lceil v \rceil - 1$, while the second contains the original model plus the constraint $x_i \geq \lceil v \rceil$. The subproblems are solved again via B&B until all variables take an integer value. To select, at each iteration, the variable to branch on, a ranking criterion, called search heuristic, must be decided.

3.4.2 Reduced Costs

We recall that from the dual model solution of a LP problem we can extract the reduced cost associated to each variable of the primal model, being it the minimum change in the OF if the value of the variable is increased by one unit in the current solution. In IP, we can obtain the reduced cost of a variable solving the LR of the IP problem.

The reduced cost of a decision variable (a variable that can assume only values 0 and 1), represents the minimum change in the OF if the variable value is set to 1. Of course, variables having value 1 in the optimal solution have a reduced cost equal to 0 and variables having value 0 have a reduced costs greater or equal than 0. The reduced cost of a decision variable can be seen as the minimum cost we will pay to change our decision by choosing another option.

Chapter 4

Integration of Constraint and Integer Linear Programming

Introduction

In this chapter we will discuss about Constraint Programming and Integer Programming integration methods. In Section 4.1 we will summarize the strong and weak points of CP and IP and in Section 4.2 we will introduce the integration techniques used in our research.

4.1 Solving techniques: pros and cons

In this Section we will summarize Constraint Programming and Integer Linear Programming, introduced in Chapters 2 and 3, discussing about their strong and weak points.

4.1.1 Integer Linear Programming

As introduced in Chapter 3, Integer Linear Programming (IP) models a problem using numeric variables and linear inequalities representing the constraints, and one linear function representing the objective function (the objective can be, for example, to minimize a cost, minimize a time, maximize a revenue). A solution to an IP problem is an assignment of values to variables such that all the constraints are satisfied and the value of the objective function is minimized (or maximized).

In IP the variables are forced to assume only integer values, (e.g. if a variable models a decision it can assume only values 0 or 1, if a variable models the number of worker employed on a task it can obviously assume only integer values); it is demonstrated that these problems, called Combinatorial Optimization Problems, are in the general case NP-hard ([43]). The Operation Research (OR) community analyzed a wide number of combinatorial optimization problems and proposed a number of algorithms to solve them, namely the Simplex algorithm, the Branch and Bound method, the cutting planes, the column generation technique to cite few.

When combinatorial optimization problems have a very clear and regular structure, IP is an efficient approach to model and solve them, but often an optimization problem involves side constraints that break the regularity of the model structure. The side constraints enlarge and complicate the problem model, and the IP solving algorithms usually worsen their behaviours when the number of constraints becomes too large or the regularity of the model structure is broken.

Summarizing, IP is an effective method to face optimization problems with a clear geometric structure (for example set packing, set covering, travelling salesman) but raises difficulty when side constraints are introduced in the problem breaking the regularity of the model.

4.1.2 Constraint Programming

Constraint Programming (CP) models a problem using generic variables that can not only assume numeric values, but also, for example, sets of values or symbolic values. Constraints imposed over the variables are not restricted to linear inequalities like in IP, but can range on mathematical relations, logical constraints, symbolic constraints and Global Constraints.

Thanks to specialized filtering algorithms, able to remove infeasible values as soon as they are recognized, the time spent in trying infeasible assignments in considerably reduced. CP should be the technique of choice when the main difficult of a problem is to find a feasible solution; when a problem has so many constraints that even finding a solution can require a large amount of computational effort, smart propagation techniques can speed up the search. On the contrary, if the problem has a great number of feasible solutions and the main difficulty is to find the optimal one, CP should not be the technique of choice. In fact, CP faces optimization problems in a very naive way: CP finds the first feasible solution and then, each time a solution is found, adds to the model an additional constraint simply stating that, from now on, each feasible solution must have an objective function value better than the best one already found. Using search heuristics (see Section 2.3.3), the search can be guided towards the portion of the search space that most probably contains the optimal solution, but nevertheless when a problem model contains a stronger optimization part w.r.t. the feasibility part CP should not be the preferred technique to solve it.

When side constraints are added to a model, the density of feasible solutions decreases (side constraints can only render infeasible some combinations of assignments). The feasibility part of the problem becomes therefore more prominent and at the same time the optimization part becomes simpler, because the optimum must be searched within a smaller set of possible solutions. So, CP can take advantage of the introduction of side constraints in pure optimization problems.

4.2 Integration of Constraint Programming and Integer Programming

In the previous subsections we have described the strong and the weak points of CP and IP summarizing that IP is more suitable for optimization problems and CP for feasibility. Typically, real problems involve both feasibility and optimization, it could therefore pay off to integrate the two techniques to solve a problem, especially when it is not clear whether optimization or feasibility is the main issue.

The underlying idea when integrating different techniques is to take the best from each technique. The simplest way is to somehow recognize the best approach and use it. This is done in the so called Algorithm Portfolios, where a set of algorithms based on different paradigms are developed. When facing an instance of the problem, two different ways can be followed:

- All the algorithms in the portfolio start the search in parallel and, when the fastest one finds the solution, all the others are stopped and the solution found is returned. This technique recognizes *a posteriori*, only when the search is finished, the best algorithm.

- The fastest algorithm for each problem instance is recognized *a priori* in order to use only it. The selection can be done by analyzing some characteristics of the instance model, for example the structure, or the search space dimension. Recognizing the best algorithm is not a trivial task, and in general advanced techniques must be explored.

Another integration method is to develop a solver based on both the techniques. IP can be used by CP to rank the variables and the variable values, CP and IP can interleave their execution during the search so that each technique can take advantage of the information gained by the other.

Some problems may have a structure where it is possible to recognize two or more sub-problems, best solved by different techniques. In this case, the integration methodology is to use the most appropriate paradigm to solve each sub-problem, building a communication mechanism allowing the two solvers to co-operate in order to find the best solution for the problem overall. This is typical for Decomposition Techniques, where a problem is decomposed in two sub-problems independent one each other or sharing a limited number of constraints. In Section 9.4 we will further discuss about Decomposition Techniques.

Of course, integration is not limited to CP and IP. Integration of CP and local search has been proposed, for example, by P. Shaw, in [118], where the author defines the so called *large neighborhood search*, and by G. Pesant and M. Gendreau [103] in the context of the TSP-TW. As far as our research is concerned, we will only investigate IP and CP.

Part II

The Bid Evaluation Problem in Combinatorial Auctions

Chapter 5

Introduction

This Part of the dissertation is devoted to the Bid Evaluation Problem (BEP), quickly introduced in Section 1.2.1. Through the analysis of the problem we will give evidence that both IP and CP are suitable programming paradigms for solving the BEP. As introduced in Section 1.2.1, the structure of the BEP, due to the presence of the temporal side constraints, is not regular: an IP approach, usually suitable for optimization problems, can thus worsen its performances because of the introduction of the side constraints. On the contrary, a CP approach, good for feasibility problems, can take advantage of the side constraints, because they reduce the number of feasible solutions and thus feasibility becomes the major issue of the problem resolution.

In the following chapters we will describe the BEP and we will present two models, based on IP and CP. We will develop several solving tools based on these models and we will show some experimental results pointing out that some of the developed algorithms are not dominated by the others on all the instances of the problem. We will put these algorithm in an Algorithm Portfolio. Next step is to find an automatic way to select, given an instance, the best algorithm to solve it. This is possible by analyzing the instance structure before modelling it using either IP or CP. Exploiting a Machine Learning approach on few structural characteristics of the instance we are able to select the best algorithm in over the 90% of the cases.

The research described in this Part of the dissertation supports the thesis that

Constraint Programming and Integer Linear Programming are ef-

fective programming paradigms for dealing with Combinatorial Optimization Problems. The structural characteristics of some classes of Combinatorial Optimization Problems can guide the selection of the solving approach....

In this chapter we will give an overview of some auction mechanisms described in literature, focussing in particular on Combinatorial Auctions (CAs) and bidding languages for CAs.

5.1 Auctions

Business to business e-commerce applications are becoming more and more popular. Among them, auctions are an important way of allocating items among autonomous and self-interested agents. Items are not limited to goods, but can represent also resources and services. Traditionally, auctions are aimed at selling or buying a single item; the auctioneer tries to maximize his/her profit if selling an item or minimize his/her outcome if buying an item. Since bidders make bids on a single item, it is easy to choose the best bid, i.e., the one providing the highest revenue. This kind of auction follows the *sequential auction mechanism*. However, it is difficult to bid in these auctions when more than one item is needed since one bidder can have preferences on bunches of items. In this case, a bidder should make hypothesis on what the other bidders will bid.

To partially solve the problem, the *parallel auction mechanism* has been proposed, where bidders can bid on a set of items simultaneously. Again, it is easy to choose the best bid by simply selecting the best one. A problem in parallel auctions can arise: it can happen that no bidding should start since all bidders wait for other bids to perform the best offer.

Recently, a third kind of auction mechanism has been proposed, the so called *combinatorial auctions* (CAs) (see [115] for an overview). In our research, we face the BEP, rising in the context of CAs, not only for its structural characteristics but also because there is a growing interest in auctions, as introduced above.

In the following we will give an overview of the most common type of sequential auctions, while in the next Section we will focus our attention on CAs.

The simplest type of auction, the one we all know, is called **English Auction**.

The auctioneer sells one item at a time, starting the auction from a *reserve price* (the lowest acceptable price) and accepting higher and higher bids from the bidders until no one will increase the offer. The last bid (the highest) is the winning one. If the auctioneer wants to buy an item the auction starts from the highest price the auctioneer is willing to pay and each bid must be lower than the last one proposed. Several variants of the English Auction exists: in the **Absolute Auction**, no reserve price is stated. In the **Dutch Auction**, the auctioneer tries to sell the item at the highest price and then lowers the price until a bidder accepts the offer. In this case the auctioneer receives only one bid, the winning one. The English and Dutch auctions are also known as Ascending and Descending auctions respectively.

The great disadvantage of the English auction mechanism (and its variants), is that each bidder and the auctioneer must be in communication one each other over the course of the auction, which can be expensive or difficult. To overcome this limitation, several auction mechanisms have been proposed: in the **First-Price Sealed-Bid Auction** (FPSB) each bidder proposes a single bid for the object without knowing the other bids. The highest bid is the winning one and pays the proposed price. The **Vickrey auction** is identical to the FPSB auction except that the winning bid pays the second highest price instead of its own. This auction boosts the bidders to bid for the true value of the item, but does not maximizes the auctioneer revenue that, in the extreme case, can be 0 if all the bidders but one do not bid for the object.

The Vickrey and the English auctions are mathematically equivalent because in both cases the winner obtains the item at the price proposed by the runner-up. This is evident for the Vickrey auction; regarding the English auction, the mechanism encourages the bidder to propose the last accepted price plus an increment. When no other bids are proposed, the price of the winner is thus equal to the second-place bidder plus the last increment.

5.2 Combinatorial Auctions

The auctions described in the last section allow to bid for an item at a time. Sometimes it could be useful to put up for auction a set of items at the same time, for example if the auctioneer wants to be sure to sell all the items he holds (or to buy all the items he needs). From the bidder side, the possibility to bid for a set of items allows to better express his preferences. Let us consider an auction for a transportation service: if a bidder have to transfer a good from city A to city C, going through city B, it is completely useless to buy only the service from A to B or from B to C. The bidder can make a bid on both the services; he is therefore sure to buy, in case he wins the auction, exactly what he needs.

Combinatorial Auctions (CAs), first proposed in [107] to solve the take off and landing time slots allocation problem in an airport, allow the bidders to submit a bid for a bundle of items proposing a price for the whole bundle: $B(S, p)$, where S is the set of proposed items and p the price. Either the bid is accepted and all the items in the subset are sold for the proposed price or the bid is refused and no items are sold. In this context rises the Winner Determination Problem: the auctioneer opens an auction for a set of items and his goal is to accept a set bids that cover all the items at the maximum revenue or minimum cost.

In CAs, the auctioneer can therefore maximize his profit. On the other side, bidders are free to propose bids reflecting their preferences in order to maximize their own profit. The real auctions may have some characteristics for which bidders are prone to give a particular evaluation to some bundle of items. For example, if we consider an auction for a set of similar items, the bidder will prefer to buy only one item rather than two or more similar items. In this case, the bidder gives an higher evaluation to two disjoint sets rather than their union. More formally:

Definition 2 *A valuation function v is a function that returns the price a bidder is willing to pay for a set of items. Given a bid (S, p), $v(S) = p$.*

Definition 3 *Two disjoint sets of items, S and T, are called complementary or substitutes for a bidder respectively if:*

- $v(S \bigcup T) > v(S) + v(T)$
- $v(S \bigcap T) < v(S) + v(T)$

Two sets of items are complementary if the bidder prefers to win both of them rather than only one; two sets are substitutes if the bidder prefers to win only one of them rather then both.

To express these preferences, several bidding languages have been proposed. In [97] the author defines six bidding languages subsuming all other bidding languages considered in the literature, and using which each bidder can express his preferences on every kind of auction. The bidding languages described in [97] are:

- **Atomic bid**: a simple bid $B = (S, p)$ where the bidder proposes a price p for a subset S of the items put up for auction.
- **OR bid**: a bidder can submit an arbitrary number of atomic bids $B_i = (S_i, p_i)$. An arbitrary number of these atomic bids can be accepted, with the obvious limitation for all the subsets S_i of the winning bids to be disjoint. An OR bid is equivalent to a set of atomic bids proposed by different bidders. With an OR bid it is impossible to express substitutability.
- **XOR bid**: a bidder can submit an arbitrary number of atomic bids $B_i = (S_i, p_i)$, but at most one of them can be accepted. With a XOR bid it is possible to express both complementarity and substitutability, but to express some kinds of valuation functions an exponential number of atomic bids in XOR one each other is needed. This is the case, for example, when a bidder gives a unary value to each item and thus $v(S) = |S|$: we need 2^m atomic bids to express the preference using a XOR bid, where m is the number of the items, while it is possible to express the same bid using an OR bid with only m atomic bids.
- **OR-of-XORs bid**: a bidder can submit an arbitrary number of XOR bids, willing to win an arbitrary number of them.
- **XOR-of-ORs bid**: a bidder can submit an arbitrary number of OR bids, willing to win at most one of them.
- **OR/XOR formula**: the most generic bids. The bidder can propose any combination of OR and XOR bids. All the other bids described above are special case of an OR/XOR formula.

Using there bidding languages it is possible to define auctions reflecting real world situations such as auctions for paths extending in the space or for lots of land, e.g. railways routes assignment, network bandwidth allocation, gas pipeline networks distribution, drilling rights, where the bidders prefer to bid on contiguous paths or adjacent lots rather that sparse or overlapping ones; auctions for rights on services for a limited slot of time, e.g. airport take-off and landing, resources allocation in the job-shop scheduling, where the bidders prefer to have several rights at the same time rather that few rights for a large amount of time.

5.3 Overview of the Part II

Part II is organized as follows: in Chapter 6 we will introduce the IP and CP models for the BEP, showing and modelling a simple combinatorial auction. In chapter 7 we will describe the implemented algorithms and we will compare themselves and with an existing tool to solve the BEP. Finally, in Chapter 8, we will present our algorithm portfolio analysis and the tool we developed to select the best algorithm in the portfolio.

Chapter 6

Problem description and modelling

Introduction

In this Chapter we will introduce the BEP. In Section 6.1 we formally describe the Bid Evaluation Problem and we will present a simple example of a combinatorial auction on coordinated services. In Section 6.2 we will introduce the CP and IP models for the WDP and in Section 6.3 we will extend these models showing the CP and IP models for the BEP. Finally, in Section 6.4, we will discuss about related works on the WDP and the BEP and on the Algorithm Portfolio analysis.

6.1 Problem description

Combinatorial Auctions (CA) are auctions where bidders have the possibility to express their preferences in a more expressive way w.r.t. classical auctions mechanisms, but the problem of selecting the winning bids, the so called Winner Determination Problem (WDP), is NP-hard. In the WDP the auctioneer has to find the set of winning bids covering all the items put up for auction; usually the auctioneer considers one or more optimization criteria.

When the items put up for auction are services to be executed, another problem arises, namely the Bid Evaluation Problem (BEP). In the BEP, beside a WDP, we have time windows and temporal constraints to take into account.

Different variants of combinatorial auctions exist. We consider the **single unit**

reverse auctions, where the auctioneer wants to buy a set M of distinguishable items (services) which are sequenced by temporal precedence constraints and are associated to temporal windows and durations, minimizing the cost. In single unit auctions the items are distinguishable while in multi unit auctions there are several units of each item. The auctions we consider are called reverse since the auctioneer has to buy items, while in traditional auctions items are sold.

We now give a formal description of the BEP.

We have one auctioneer, a set B of bidders ($|B| = n$) and a set M of services ($|M| = m$) that must be bought by the auctioneer during the auction. Without loss of generality, we assume that each bidder j posts only one bid $B_j = (S_j, Est_j, Lst_j, D_j, p_j)$, where $S_j \subseteq M$ is proposed to be sold at the price p_j. Est_j and Lst_j are lists of earliest and latest starting time of the services in S_j and D_j their duration.

The auctioneer posts an auction for buying all services in M. In addition, the auctioneer considers temporal constraints on the services in M. For example, between two services i and k there might be a precedence constraint $Start_i + Duration_i \leq Start_k$. These constraints are not communicated to bidders that can therefore submit bids not fulfilling these constraints. When the auctioneer selects the set of winning bids, covering M, he/she should check that temporal windows provided by the bidders satisfy the constraints.

The problem is to find the set of bids covering M at the minimum cost, respecting all the temporal constraints.

We describe here a simple example of a BEP, where the auctioneer wants to buy 3 services, t_1, t_2 and t_3, minimizing the total cost. A precedence constraint is imposed, stating that t_3 must be executed after both t_1 and t_2 are completed. Figure 6.1 shows the precedence graph (private to the auctioneer) for the example, while Table 6.1 shows some bids that are received for this auction.

Some qualitative considerations follow:

- Each bidder gives a single price for a bundle of services.
- Each bidder provides an earliest start time (**Est**), a latest start time (**Lst**) and a duration (**D**) for each service individually.
- Bid b_3 must be a winner because it is the only one proposing the service t_3.

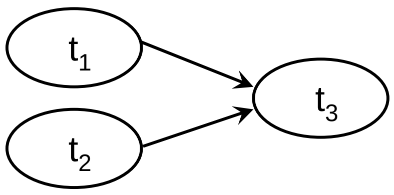


Figure 6.1: Precedence graph for the example in Table 6.1

Bid	**Services**	**Est**	**Lst**	**D**	**p**
b_1	t_2	110	135	120	290
b_2	t_2	140	160	140	150
b_3	t_1	15	30	110	300
	t_3	225	250	95	
b_4	t_1	10	40	100	120

Table 6.1: Example of bids on three services

- Bids b_1 and b_2 cannot be accepted together because they both provide the service t_2. Each service must be covered by exactly one bid. For the same reason, bids b_3 and b_4, both providing the service t_1, cannot be accepted together.

- Bids b_2 and b_3 cannot be accepted together, because the precedence relation $t_2 \prec t_3$ would be violated. This happens because the earliest time b_2 could complete t_2 is 280, while the latest time b_3 could start t_3 is 250.

- In the optimal solution for this problem the winning bids are (b_1, b_3). Service t_1 starts at 15, ends at 125 and is executed by b_3; service t_2 starts at 110, ends at 230 and is executed by b_1; service t_3 starts at 230, ends at 325 and is executed by b_3. t_3 starts at 230 and not at 225 (the early start time proposed by b_1) because the execution of t_2 ends at 230 and t_3 must start after the end of t_2. The total cost is 590.

In the example in Table 6.1, if we do not consider the temporal constraints, we obtain a WDP; in this case, the solution found for the BEP is still a feasible solution, being the WDP a sub-problem of the BEP, but now (b_2, b_3) is the optimal solution since the cost, 450, is lower.

6.2 Modelling the Winner Determination Problem

As introduced, the BEP is a WDP with temporal side constraints. In this section, we introduce the IP and CP models for the WDP and we will extend them to cope with temporal constraints in the next section.

6.2.1 IP model

In the integer linear model of the WDP we have decision variables x_j taking the value 1 if the bid $B_j = (S_j, Est_j, Lst_j, D_j, p_j)$ is winning, 0 otherwise. The IP model for the WDP is the following:

$$\min \sum_{j=1}^{n} p_j x_j \tag{6.1}$$

$$s.t.: \sum_{j|i \in S_j} x_j = 1 \ , \ \forall i \tag{6.2}$$

$$x_j \in \{0, 1\} \ , \ \forall j \tag{6.3}$$

The objective function (6.1) minimizes the total cost which is computed as the sum of prices p_j of winning bids. Constraints (6.2) state that the number of winning bids containing the same item (service) should be equal to one. This means that all services should be covered and each service should be covered by exactly one bid.

We can see that the model structure is very simple and very clear. It is the formulation of a set partitioning problem that is a structured, well known and widely studied problem in the Operations Research community [18], best solved by an IP approach.

An important assumption that can be done in combinatorial auctions is that of *free disposal*. In this case, not all services should be covered. In the reverse combinatorial auctions, under the free disposal assumption items can be bought more than once by the auctioneer. Thus, if free disposal would be assumed, symbols $=$ in constraints (6.2) in the above model are transformed in $\geq$. The only constraint imposed on winning subsets of bids is that the union of all the subsets should be equal to the set of services put up for auction.

6.2.2 CP model

The WDP can be easy modelled also in Constraint Programming. We have a set of m variables $X_1, \ldots, X_m$ representing the services to be bought. Each variable X_i ranges on a domain containing the bids mentioning service i. We have a set of n variables $Cost_1, \ldots, Cost_n$ representing the cost of the bid in the solution. Each variable $Cost_j$ can assume either the value 0, if bid j is a losing bid, or p_j, if it is a winning one. The CP model for the WDP is the following:

$$min \sum_{j=1}^{n} Cost_j \tag{6.4}$$

$$X_i = j \rightarrow X_k = j \ , \ \forall i, \ \forall k \in S_j \tag{6.5}$$

$$X_i = j \rightarrow Cost_j = p_j \ , \ \forall i \tag{6.6}$$

The objective function (6.4) minimizes the sum of the variables $Cost_j$. (6.5) and (6.6) are channelling constraints modelling the following ideas: if a service is taken from the bid B_j, all other services in S_j should be taken from the same bid (6.5) and the cost of the bid in the solution should be p_j (6.6).

This model completely describes the WDP, but another important constraint that can trigger an effective propagation is a specialization of the global cardinality constraint (gcc) [109], introduced in 2.4. We briefly recall that gcc limits the number of occurrences of a set of values among a set of variables within a given interval. The specialization we use, namely the *Distribute* constraint, has the same heading of gcc:

$$Distribute(Var, Val, LB, UB) \tag{6.7}$$

The *Distribute* constraint forces the number of occurrences of each value Val_i among the variables Var to be either LB_i or UB_i. In other words, the only difference between *Distribute* and gcc is that gcc imposes $Val_i \in [LB_i, \ldots, UB_i]$, while *Distribute* imposes $Val_i \in \{LB_i, UB_i\}$.

In our CP model, we can use the *Distribute* constraint as follows:

$$Distribute(X, [1, \ldots, n], 0, |S|) \tag{6.8}$$

where X is the array of variables representing services to be sold, the second parameter is an array of numbers tidily from 1 to n, n being the number of bids, and $|S|$ is an array where each element $|S_j|$ is the cardinality of the set of services contained in the bid j. This constraint holds iff the number of occurrences of each value $j \in [1, \ldots, n]$ assigned to X is exactly either 0 or $|S_j|$. In other words, the constraints state that, if the bid B_j is chosen as winning, the number of variables X_i taking the value j is exactly the cardinality of the set S_j. Otherwise, if the bid B_j is not chosen as winning, that number is 0. For example, let us consider a bid b_1 providing three services $S_1 : [1, 2, 3]$. If b_1 is a winner, variables X_1, X_2 and X_3

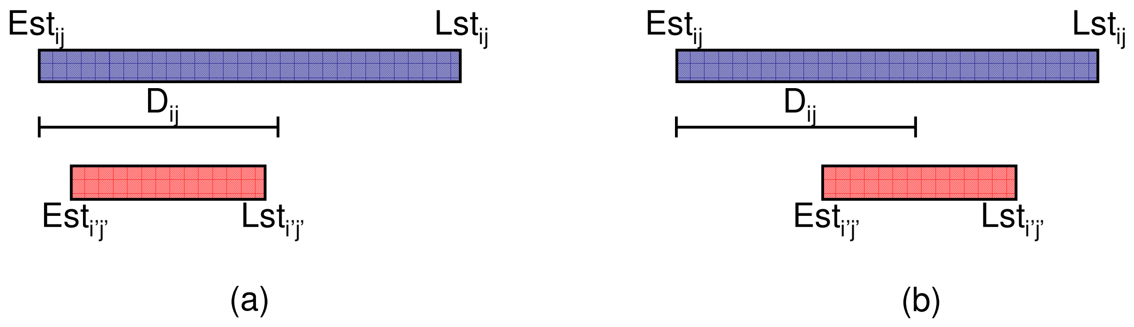


Figure 6.2: Example of temporal overlapping windows

must take the value 1, while if it is a loser no X_i variables will take the value 1. The Distribute constraint states that the value 1 (first element of the second argument), must appear among the X_i variables either 0 or $|S_1|$ (first element of the last argument) times, being $|S_1| = 3$.

6.3 Modelling the Bid Evaluation Problem

In this section, starting from the models described in Sections 6.2.1 and 6.2.2, we introduce the IP and CP models used to solve the BEP.

6.3.1 IP model

The BEP is a WDP augmented with temporal constraints, thus the IP model for the BEP contains the constraints (6.1), (6.2) and (6.3), defining the IP model for the WDP. In addition, we also have temporal constraints, introduced as follows: we have variables $Start_{ij}$ associated to each service $i = 1 \ldots m$ taken from each bid $j = 1 \ldots n$. These variables range on the temporal windows $[Est_{ij}, Lst_{ij}]$. For each pair of services i and i' linked by a precedence constraint, where i' must be executed after the end of i, we find all pairs of bids j and j' containing that services; if S_j and $S_{j'}$ have an empty intersection we compute $Est_{ij} + D_{ij} - Lst_{i'j'}$, where D_{ij} is the duration of i in bid j. In case the result is positive (see Figure 6.2(a), where an example of temporal overlapping windows is given), that is the domains of $Start_{ij}$ and $Start_{i'j'}$ do not contain any pair of values that could satisfy the precedence relation, we introduce the constraint (6.9) which prevents both bids from appearing in the same solution; otherwise, if the result is zero or negative (Figure 6.2(b)), we introduce the constraint (6.10), where M is a large number. The term $M(x_j + x_{j'})$ makes the constraint satisfied in cases where either $x_j = 0$ or $x_{j'} = 0$.

$$x_j + x_{j'} \leq 1 \tag{6.9}$$

$$Start_{ij} + D_{ij} - Start_{i'j'} + M(x_j + x_{j'}) < 2M \tag{6.10}$$

Therefore, recalling that $B_j = (S_j, Est_j, Lst_j, D, j, p_j)$, $j = 1 \dots n$, the complete IP model for the BEP is the following:

$$\min \sum_{j=1}^{n} p_j x_j \tag{6.1}$$

$$\sum_{j | i \in S_j} x_j = 1 \ , \ \forall i \tag{6.2}$$

$$x_j \in \{0, 1\} \ , \ \forall j \tag{6.3}$$

$$\forall \ i, i', j, j' | i \prec i', S_j \cup S_{j'} = \emptyset, i \in S_j, i' \in S_{j'}$$

$$\begin{cases} x_j + x_{j'} \leq 1, & \text{if } Est_{ij} + D_{ij} - Lst_{i'j'} > 0 \quad (6.9) \\ Start_{ij} + D_{ij} - Start_{i'j'} + M(x_j + x_{j'}) < 2M, & \text{if } Est_{ij} + D_{ij} - Lst_{i'j'} \leq 0 \quad (6.10) \end{cases}$$

$$Est_{ij} \leq Start_{ij} \leq Lst_{ij} \ , \ \forall j \ , \ \forall i \in S_j \tag{6.11}$$

We can see that the BEP model structure is much more complex w.r.t. the WDP; in fact, the temporal side constraints introduce some irregularities in the structure that worsen the IP behaviours.

The IP model for the example in Table 6.1 is the following:

$$minimize\ (290x_1 + 150x_2 + 300x_3 + 120x_4) \tag{6.12a}$$

$$x_3 = 1 \tag{6.12b}$$

$$x_1 + x_2 = 1 \tag{6.12c}$$

$$x_3 + x_4 = 1 \tag{6.12d}$$

$$x_2 + x_3 \leq 1 \tag{6.12e}$$

$$Start_{13} + 110 - Start_{33} + M(x_3 + x_3) < 2M \tag{6.12f}$$

$$Start_{14} + 100 - Start_{33} + M(x_3 + x_4) < 2M \tag{6.12g}$$

$$Start_{21} + 120 - Start_{33} + M(x_1 + x_3) < 2M \tag{6.12h}$$

$$Start_{22} + 140 - Start_{33} + M(x_2 + x_3) < 2M \tag{6.12i}$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\} \tag{6.12j}$$

$$Start_{21} \in \{110..135\} \tag{6.12k}$$

$$Start_{22} \in \{140..160\} \tag{6.12l}$$

$$Start_{13} \in \{15..30\} \tag{6.12m}$$

$$Start_{33} \in \{225..250\} \tag{6.12n}$$

$$Start_{14} \in \{10..40\} \tag{6.12o}$$

6.3.2 CP model

Similarly to the IP models, also for CP we start from the WDP model, defined by the constraints (6.4), (6.5), (6.6) and (6.8). To deal with the temporal constraints, we introduce the variables $Duration_i$ and $Start_i$, associated to each service i. $Duration_i$ ranges on the set of all duration D_{ij} for service i taken from all bids j mentioning i. $Start_i$ ranges on the union of all temporal windows $[Est_{ij}, Lst_{ij}]$ for service i taken from all bids j mentioning i.

In addition, if two services i and i' are linked by a precedence constraint, then the constraint (6.13) is introduced. Obviously, variables $Start$, $Duration$ and X are connected by channelling constraints, in the sense that, if a value j is assigned to a variable X_i, the domain of $Start_i$ should be set to $[Est_{ij} \ldots Lst_{ij}]$ (6.14), and $Duration_i$ should be set to D_{ij} (6.15).

$$Start_i + Duration_i \leq Start_{i'} \tag{6.13}$$

$$X_i = j \rightarrow Start_i :: [Est_{ij}, Lst_{ij}] \tag{6.14}$$

$$X_i = j \rightarrow Duration_i = D_{ij} \tag{6.15}$$

The complete CP model for BEP is the following:

$$min \sum_{j=1}^{n} Cost_j \tag{6.4}$$

$$X_i = j \rightarrow X_k = j \ , \ \forall i \ , \ \forall k \in S_j \tag{6.5}$$

$$X_i = j \rightarrow Cost_j = p_j \ , \ \forall i \tag{6.6}$$

$$Distribute(X, [1, \ldots, n], 0, |S|) \tag{6.8}$$

$$Start_i + Duration_i \leq Start_{i'} \ , \ \forall i, i' | i' \succ i \tag{6.13}$$

$$X_i = j \rightarrow Start_i :: [Est_{ij}, Lst_{ij}] \ , \ \forall i \tag{6.14}$$

$$X_i = j \rightarrow Duration_i = D_{ij} \ , \ \forall i \tag{6.15}$$

$$X_i :: \{j | i \in S_j\} \ , \ \forall i \tag{6.16}$$

$$Cost_j :: [0, p_j] \ , \ \forall j \tag{6.17}$$

$$Start_i :: \left\{ [Est_{ij}..Lst_{ij}] | i \in S_j \right\} \ , \ \forall i \tag{6.18}$$

$$Duration_i :: \{D_{ij} | i \in S_j\} \ , \ \forall i \tag{6.19}$$

We can see that the CP model is not complicated too much by the temporal side constraints. We simply introduce the precedence constraints (6.13) and the channelling constraints (6.14) and (6.15).

The CP model for the example in Table 6.1 is the following:

$$minimize\ (Cost_1 + Cost_2 + Cost_3 + Cost_4) \tag{6.20a}$$

$$Distribute([X_1, X_2, X_3], [1, 2, 3, 4], [0, 0, 0, 0], [1, 1, 2, 1]) \tag{6.20b}$$

$$Start_1 + Duration_1 \leq Start_3 \tag{6.20c}$$

$$Start_2 + Duration_2 \leq Start_3 \tag{6.20d}$$

$$X_1 :: [3, 4]\ ,\ X_2 :: [1, 2]\ ,\ X_3 :: [3] \tag{6.20e}$$

$$Cost_1 :: [0, 290]\ ,\ Cost_2 :: [0, 150]\ ,\ Cost_3 :: [0, 300]\ ,\ Cost_4 :: [0, 120] \tag{6.20f}$$

$$Start_1 :: [10..40]\ ,\ Duration_1 :: [100, 110] \tag{6.20g}$$

$$Start_2 :: [110..135, 140..160]\ ,\ Duration_2 :: [120, 140] \tag{6.20h}$$

$$Start_3 :: [225..250]\ ,\ Duration_3 :: [95] \tag{6.20i}$$

augmented with the channelling constraints (6.5), (6.6), (6.14) and (6.15).

6.4 Related work

The aim of our work is twofold. We will develop a portfolio of BEP solvers and an algorithm selection tool based on machine learning. In this Section we will discuss previous works related to WDP and BEP solvers and to algorithm portfolios and algorithm selection tools.

6.4.1 Existing tools to solve the WDP and the BEP

CAs are receiving always growing attention since the computational power of computers and sophisticated optimization methods enable the solution of hard problems in a reasonable time. The WDP is equivalent to a set partitioning problem, a well known problem in Operation Research, that is best solved by Integer Programming techniques. In literature, the WDP is largely analyzed and both IP-based approach and specialized search algorithms are presented. In [20] the authors apply a stochastic local search algorithm, called Casanova, to the WDP. In [101], after describing and analyzing various CA mechanisms, the authors address bidding languages and efficiency matters. They discuss search strategies for solving the WDP and describe five real world applications where CAs can be successfully applied. In [121] a survey on CA is presented, describing design methods and WDP solving techniques based on IP. In [115] the author presents a search algorithm based on a structure called BidTree and some heuristics to improve the search, and tests them on a variety of different bid distributions. Starting from this work, in [95] the authors implement the WDP in ECL$^{\mathrm{i}}$PS$^{\mathrm{e}}$ and they solve the problem using a BidTree-based solving algorithm introduced above, comparing it with general selection heuristics such as the Most Constrained Bid (MCB) and the Most Valuable Bid (MVB) heuristics; at each branch, the former heuristic chooses the bid involved in the greatest number of covering constraint, while the latter chooses the bid with the lowest value of the price divided by the number of items, that is the cheapest bid. We will use the MVB heuristic for our research (see Sections 7.1 and 7.3).

Combinatorial Auctions Test Suite (CATS)

In [88] the authors present CATS, a suite of distributions for modelling realistic bidding behaviours able to generate realistic WDP instances. With CATS it is possible, by setting several parameters, to generate WDP instances where the bidders use a particular bidding language (see Section 5.2) and can express complementarities or substitutability on the subsets of item put up for auction. CATS can also generate instances reflecting previous distributions described in literature, for example those listed in [114], [41], [19]. These distributions do not reflect real situations, but make use of different functions to select the number of items each bid will contain: to cite some of them, the Uniform distribution randomly selects the number of items in the interval $[1 \ldots n]$, where n is the number of items put up for auction; the Decay distribution adds successive items to a bid with a decaying probability; the Exponential distribution creates bids with x items with a probability inversely proportional to an exponential function of x.

Multi AGent Negotiation Testbed (MAGNET)

As described above, the WDP has been largely analyzed in the literature. On the contrary, the BEP received much less attention. To the best of our knowledge, the only solving tool addressing the BEP is MAGNET (Multi AGent Negotiation Testbed) [24], a commercial tool, developed at the University of Minnesota, that provides support for complex agent interactions and is able to generate and solve BEP instances. MAGNET is based on Integer Programming and can perform both complete and incomplete search implementing Simulated Annealing, a search strategy that overcomes the problem of local optima allowing, during the search, to select branches that worsens the objective function. This strategy is inspired by metal annealing process.

The traditional IP solver is based on branch and bound and provides the optimal solution, if able. The incomplete strategy is an anytime algorithm: we can stop the search when a timeout occurs or when no improvements are found within a given time, so also sub-optimal solutions can be found.

6.4.2 Algorithm portfolio

In [49] an algorithm portfolio is defined as "... a collection of different algorithms and/or different copies of the same algorithm running on different processors.". The algorithm portfolio design appears first in [60]. The authors consider a portfolio of algorithms and, using the notion of economic risk, derive the probability distribution for an algorithm to end the search within a given time. Using information from this probability distribution they interleave the algorithms to solve instances of the graph coloring problem. Experimental results show a performance increasing of about 30% w.r.t. using a single algorithm.

[49] consider stochastic algorithms and hard combinatorial problems such as the Quasigroup Completion Problem. They show that, running the algorithms in parallel or interleaving them on a single processor (this technique is called the *restart technique* [36]) achieves strong computational advantage.

[42] propose a dynamic algorithm portfolio method, where the algorithms run in parallel on different machines and, at each time slot, the relative priority between them is updated on the basis of their expected closeness to the solution.

Usually, in the context of algorithm portfolio, the algorithm selection problem is considered. The algorithm selection problem, that is the problem of selecting the best, or simply a good, algorithm to solve a problem, can arise in a large variety of cases. When defining an algorithm portfolio for the selection problem it is important to include in the portfolio only the algorithms that could be selected. In particular, if, given a set of algorithms, one of them is dominated by at least one of the others on all the experiments performed, it must not be included in the portfolio because it will never be the best algorithm.

In literature, the algorithm selection problem has been studied since the seventies. It is first formulated in [110], where an abstract model for the problem is defined.

Algorithm selection is particularly useful when solving hard combinatorial optimization problems where the difference between a good algorithm and a bad one determines the capability of solving the problem or not. Typically, the algorithm selection is performed finding a way to predict the computational effort an algorithm spends to solve a particular instance of a problem. [90] analyze a set of well known Branch and Bound algorithms, taking advantage on the Knuth sampling method [73] to estimate the size of the search tree. Experimental results show that the proposed method is effective both on randomly generated and realistic highly structured problems. [37] describes a statistical technique for algorithm selection in planning and transportation problems. [58] propose a complete knowledge-based tool for problem specification and algorithm description and selection for scientific software.

Machine learning as been extensively used for algorithm performance prediction and algorithm selection. [99] use a Bayesian model to predict the algorithm run time on the basis of structural characteristics of the instances and applied the technique to hard CSP and SAT problems with high running time variance. A Bayesian approach is also used in [22] in the context of scheduling problems. The authors use a low-knowledge approach: only few and inexpensive measurements can be done to identify the features describing the algorithms. [79] exploit reinforcement learning for dynamic algorithm selection in the context of sorting problems. In [123] WEKA, a machine learning tool for data mining, is used for algorithm selection in planning problems.

[99] propose a portfolio of algorithms for SAT problems. They identify a set of easy-to-compute features to describe the empirical hardness of SAT instances and they use a regression technique to predict the algorithms running time. [61] show that machine learning techniques for algorithm selection can be effective also with stochastic algorithms. They analyzed the Bayesian Linear Regression and the Gaussian Process Regression methods.

[8] address the question of selecting an algorithm from a predefined set that will

have the best performance on a scheduling problem instance. The aim is to use low-knowledge dynamic techniques. Each algorithm, is run for a fixed number of CPU seconds on the problem instance. The results of each run are then used to select the algorithm that will achieve the best performance given the time remaining.

As we will describe in deep in Section 8.1, our work is strongly based on [85]: in particular, we will take advantage of the structural features of a WDP instance described in [85] to characterize our BEP instances. In the same paper, the authors propose a statistical regression method to predict the CPLEX running time to solve WDP instances. This is a first step in the direction of automatic algorithm selection, since the authors do not select the algorithm, but predict the algorithms run-time to deduce whether it will perform well or not on a given instance.

The same authors, together with Andrew and McFadden, in [84] propose a portfolio of algorithms for the same problem and compare the predicted running times. This work is extended in [83] exploiting another machine learning paradigm, namely the Boosting technique [116], for algorithm running time prediction.

A survey of these works also appears in [87] and in [86]. In the latter the authors claim that decision tree based selections, performing off-line classifications, "... penalize misclassifications equally regardless of their cost." Their idea is to minimize the average portfolio running time, not the selection tool misclassification. Here we will give evidence that our tool is effective for selecting the best algorithm and this minimizes the average running time. In fact, we will show that, when our tool misses the right classification, the algorithms running times are very close one each other.

[54] face the Most Probable Explanation problem for bayesian networks. They consider six solving algorithms, one of which is complete, and try to answer two questions: is the instance exactly solvable? If the answer is yes the complete algorithm is chosen, otherwise they try to answer another question: which incomplete algorithm is the best one to solve the instance? To answer the latter question they use and compare six machine learning methods, one of which is Decision Trees, based on few easy-to-compute features describing the instance. Experimental results show that Decision Trees is the best learning method, having the higher prediction rate. Of course, results show that the prediction rate is higher when the features accurately describe the instance structure [45].

In our opinion, [85] and [54] motivate our work. In fact, [85] propose a set of parameters that accurately describe a WDP instance while, as introduced above, [54] show that Decision Trees is an appropriate technique for algorithm selection when used in conjunction with accurate features. For these reasons, given that the BEP can be seen as a WDP generalization, we believe that our approach can lead to satisfactory results.

Chapter 7

Algorithms and Experimental Results

Introduction

In this Chapter we will describe the implemented algorithms to solve the BEP and we will show the experimental results. In Sections 7.1 and 7.2 we will describe a CP based algorithm, some experimental results and a comparison with another existing tool to solve the BEP. In Section 7.3 we will describe two algorithms, one based on IP and an Hybrid one based on both IP and CP, to solve the BEP, presenting some experimental results and comparing their behaviours in Section 7.4. We will see that some of the developed algorithms are not dominated by the others on all the instance. Depending on the instance, one algorithm can be better than another; this is one of the cases where an algorithm portfolio approach (see 8.1) can pay off. In Section 7.5 we will draw some considerations on the portfolio, referring the reader to the next Chapter for a deep dealing with the argument.

7.1 CP algorithms

We implemented a pure CP based approach based on the CP model described in Section 6.3.2. Variables and variable values selection heuristics are defined on the variables X. We recall that each variable represents a service to be bought and the domain ranges on all the bids proposing the service.

The variable selection heuristic is based on the First Fail Principle, that selects the variable with the smallest remaining domain. We try to buy first the services that are sold by the lower number of bidders.

For the variable value selection we used a specialized heuristic, namely the Most Valuable Bid (MVB) principle introduced in Section 6.4.1, that selects first the value representing the bid j with the lower $p_j/|S_j|$ value, that is the price-for-service value, the cost for each single service in the bid, assuming that all the services in a bid have the same cost. This assumption is not misleading since each bid can only be accepted as a whole, or rejected. The objective is to minimize the cost for buying the services, so we first try to buy the services from the cheaper bidders, those with the lower price-for-bid value. If choosing the cheaper bidders is consistent with the problem constraints, the optimal solution is found, otherwise other bidders, with an higher price-for-service value, are tried until the optimal solution is found.

We implemented two variant of this CP solver exploring the search tree using Depth First Search (DFS) and Slice Based Search (SBS)[1] with a discrepancy step experimentally set to 4 (see Section 2.3.4). In the following, this two solving tools will be referred to respectively as CP-DFS and CP-SBS.

7.2 Experimental results

In this Section we will describe the tool used to generate our data set, we will show the results solving the BEP with the two CP approaches described above and we will compare them with MAGNET, introduced in Section 6.4.1.

7.2.1 Data set generation

The most important parameters when generating BEP instances are: the number of bids and services; the number of services included in each bid; the type of the services, i.e. an estimation of their execution hardness and thus of their mean duration and price proposed by a bidder.

To generate the problem instances we will solve, we used MAGNET [24], mainly because it is able to generate and solve BEP instances and therefore we can directly compare the results obtained when solving the same instances with MAGNET and with our solvers on the same machine. With MAGNET it is possible to set some

[1]also known as Depth Bounded DFS (DB-DFS)

parameters to differentiate the instances, e.g. the bid-size variability, that is a number, ranging from 0 to 1, defining the variability of the number of services proposed by each bidder; it is also possible to label the services with a type.

We generated four kinds of instances: the first, very easy to solve, with 5 services requested by the auctioneer and a number of bids varying from 13 to 19; the second with 10 services and a number of bids varying from 29 to 41; the third with 10 services and a number of bids varying from 88 to 109; the last, very hard to solve, with 20 services and a number of bids varying from 372 to 421[2].

7.2.2 Comparing CP and Magnet

In this section we will show and compare the results when solving the instances described above with our CP solvers (CP-DFS and CP-SBS) and with MAGNET in its complete and incomplete version. The complete one, based on Integer Programming, will be referred to as M-IP, while the incomplete one, based on Simulated Annealing, will be referred to as M-SA. We ran our experiments on a 2.4GHz Pentium 4 with 512MB RAM and using ILOG Solver 5.3 [65] as CP solving tool.

In Table 7.1 each line represents a BEP instance. The Services, Bids and Opt columns report the number of services requested by the auctioneer, the number of bids and the optimal solution respectively. The Best M-SA column reports the best solution found by M-SA and, if it is not optimal, the percentage w.r.t. the optimal solution is shown. The other four columns represent the search time (in ms) respectively for MAGNET implementing IP (column M-IP), MAGNET implementing SA (M-SA) and the CP-DFS and CP-SBS algorithms (columns CP-DFS and CP-SBS). The symbol '-' in the Time column means that the algorithm was not able to find a feasible solution within the time limit, set to 15 minutes. The last two columns represent the number of failures occurred when solving the problem using CP-DFS or CP-SBS. We do not have the same information from MAGNET.

In the first two sets of experiments (5 services and 15 bids on average and 10 services and 35 bids on average), the M-IP approach always finds the optimal solution, while in the third (10 services and 100 bids on average) it does not provide any optimal solution within the time limit. In the first set also M-SA provides the optimal solution, while, in the second set, it finds the optimal solution only in the

[2]These instances are available on the web at http://www-lia.deis.unibo.it/Staff/AlessioGuerri/BEP_LIB

Services	**Bids**	**Opt**	**Best M-SA**	**Time (ms)**				**Failures**	
				MAGNET		**CP**		**CP**	
				M-IP	**M-SA**	**CP-DFS**	**CP-SBS**	**CP-DFS**	**CP-SBS**
5	13	6624	6624	80	10	10	10	2	3
5	14	10311	10311	70	10	10	10	4	7
5	15	8496	8496	60	40	30	30	16	3
5	16	9508	9508	70	30	30	30	19	17
5	16	9622	9622	71	30	10	10	6	8
5	16	10920	10920	141	311	10	10	1	1
5	16	12319	12319	60	10	10	10	3	3
5	16	12979	12979	70	30	11	10	6	6
5	17	11384	11384	60	50	10	10	2	2
5	19	10333	10333	90	40	10	10	3	8
10	29	17653	17653	160	2100	10	10	8	8
10	30	19758	19758	100	1582	421	40	220	27
10	31	15318	15318	160	1532	20	10	5	5
10	31	15317	16532 (92%)	80	1933	10	10	0	0
10	32	15172	15172	100	1091	20	20	6	6
10	33	17297	17297	90	831	10	10	4	4
10	34	16492	16492	281	1993	10	10	4	4
10	34	19115	22927 (83%)	140	1812	60	40	59	44
10	36	17795	18059 (98%)	100	2085	20	20	7	7
10	41	15865	17005 (93%)	150	1352	10	10	4	4
10	88	14088	18037 (78%)	-	3265	761	51	16414	230
10	98	14107	16746 (84%)	-	4186	12458	490	99300	2543
10	98	17519	20643 (85%)	-	1131	6609	70	57486	398
10	100	16065	20862 (77%)	-	2173	10	10	1	1
10	100	12106	14031 (86%)	-	3175	20	10	20	17
10	106	19468	22521 (86%)	-	1873	1372	40	11119	138
10	109	15274	17994 (85%)	-	3245	341	30	2148	76
10	110	13815	19063 (72%)	-	2093	20	30	4	4

Table 7.1: Results on instances generated by MAGNET

60% of the cases. In the third set of experiments M-SA never computes the optimal solution, but solutions that are quite far (between 72% and 86%) from the optimum. Our algorithm always finds the optimal solution for all the instances and the time to produce it is always the lowest.

These results point out that our approach always outperforms MAGNET both in search time and in solution quality for the instances considered.

We analyzed harder instance to further give evidence that our approach outperform MAGNET. Table 7.2 shows the results on the fourth instance set: each line represents an instance with 20 services and 400 bids on average. These instances are

Services	Bids	Best solution found			Search time (ms)			Failures	
		Magnet	CP		Magnet	CP		CP	
		M-SA	CP-DFS	CP-SBS	M-SA	CP-DFS	CP-SBS	CP-DFS	CP-SBS
20	372	55%	88%	19490	8700	25	100	30	454
20	372	-	96%	28036	-	25	256049	39	1.1M
20	377	56%	24789	96%	5038	381036	90618	2.09M	249k
20	378	72%	97%	26859	9006	25	32	33	29
20	393	68%	24718	99%	2956	25	46202	28	152k
20	401	69%	75%	26833	5560	30	7354	64	28991
20	407	63%	98%	25997	6297	1018	591627	4157	1.3M
20	408	24467	98%	24467	6450	25	30	10	10
20	421	66%	99%	23887	8131	575051	48	2.5M	67

Table 7.2: Results on hard instances generated by MAGNET

very hard to solve so it is not possible to find the optimal solution (or to prove the optimality of a solution) with any of the approaches considered within the time limit set to 15 minutes. In the columns Best Solution Found we report the best solutions found by M-SA, CP-DFS and CP-SBS within the time limit; for each instance, the best solution obtained by one of the three algorithms is reported, while the other two results are described as percentage w.r.t. the best solution found. In this table, we compare only M-SA, CP-DFS and CP-SBS since M-IP does not compute any solution within the time limit. The letter k in the failure column means 10^3 while M means 10^6. It is worth noting that our solutions are, on average, 30% better than those produced by M-SA. Moreover, the time to produce the best solution is in general considerably lower than 15 minutes.

The relative quality of M-SA with respect to CP-DFS and CP-SBS is also depicted in Figure 7.1, where we show the trend of the solution quality for hard instances solved using M-SA, CP-DFS and CP-SBS. The y-axis represents the solution quality in terms of percentage of the solution w.r.t. the best solution found for the instance considered. The x-axis represents the percentage of occurrence of a given solution quality. For example, M-SA finds a solution with a quality of 80% in the 12% of the cases, CP-DFS in around the 90% of the cases and CP-SBS in the 96%: viceversa, in the 70% of the cases CP-SBS ensures a solution quality of 100% (i.e. the optimal solution), while CP-DFS ensures a quality of 97% and MAGNET only 60%.

In this Section we have given evidence that MAGNET is always outperformed by our CP approaches both in search time and solution quality for all the kind

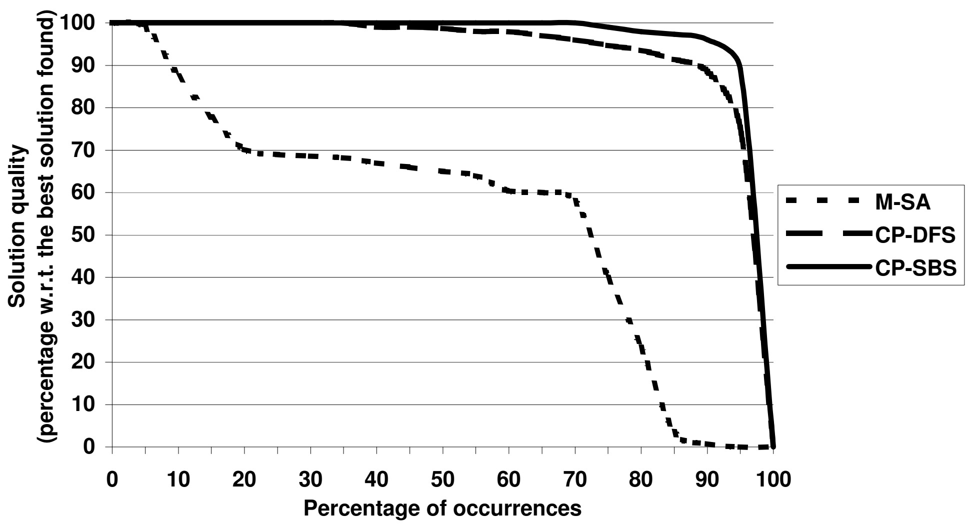


Figure 7.1: Trend of the solution quality for instances of 20 services and 400 bids

of instances considered, from the simplest with only 5 services and 10 bids to the hardest with 20 services and 400 bids. As far as our analysis is concerned, we will not include MAGNET in the algorithm portfolio. Furthermore, we can see that CP-SBS always outperforms CP-DFS, therefore in the following we consider only SBS as search strategy and we will refer to the CP-SBS algorithm as CP.

7.3 IP based and hybrid algorithms

In the last section we have given evidence that our CP based approaches always outperform MAGNET. In this Section we will describe two algorithms based on the IP model and one hybrid algorithm based on both the CP and IP models, introduced in Chapter 6.

7.3.1 IP based algorithms

The first IP algorithm, based on the IP model presented in Section 6.3.1, implements the IP Branch and Bound. As introduced in Section 3.4, the Branch and Bound algorithm selects the variables on the basis of a search heuristic: similarly to the CP based algorithm, we used the Most Valuable Bid (MVB) principle, choosing first the cheaper bid according to the price-for-service value, that is the bid price divided by the number of services proposed. The variable x_j which associated bid has the lower

$p_j/|S_j|$ value are selected first. In the following, this algorithm will be referred to as IP.

We developed an incomplete IP algorithm based on the IP model presented in Section 6.3.1. The algorithm implements Branch and Bound and is based on the reduced costs of the variables x. It works as follows: first, the linear relaxation of the WDP subproblem of the BEP instance is solved to optimality (temporal constraints are removed for efficiency reasons). Variables x_j are ranked according to their reduced cost rc_j from lower to higher. Then, we solve the IP problem considering only the first $r\%$ variables, where r is a parameter to be experimentally tuned. The other variables are fixed to 0. We solved the LR relaxing also temporal constraints since the search time to solve the LR with temporal constraints was sensibly higher and the rankings obtained in the two cases are very similar. The Branch and Bound algorithm selects first the variable x_j with the lower rc_j. In the following, this algorithm will be referred to as LR+IP.

7.3.2 Hybrid algorithm

The last algorithm we implemented is a hybrid algorithm based on both the IP and CP models presented in Section 6.3. The hybrid algorithm is very similar to the CP algorithms presented in Section 7.1, but integrates the linear relaxation of the IP model and exploits its results in the CP solving strategy. First, the LR of the WDP is solved to optimality (temporal constraints are relaxed as for LR+IP); for each bid we consider the minimum between $p_j/|S_j|$ and the reduced cost rc_j of the associated IP variable x_j, both normalized w.r.t. the maximum of each value over all bids. We used these numbers to rank the variable values, starting from the variable with the lower value. The heuristic integrates the MVB principle and the reduced costs based ranking. We explore the search tree using SBS with a discrepancy step set to 4. In the following, this algorithm will be referred to as HCP[3].

7.4 Experimental results

In this Section we will show the results solving the BEP with the IP-based and hybrid approaches described above and we will compare them with the CP based

[3]indeed it is strongly based on CP

one, introduced in Section 7.1.

7.4.1 Data set generation

We consider again the instances generated using MAGNET and besides we tried to generate other instances with a different structure. In fact, even though MAGNET can set some parameters, the instances generated are quite similar one another and, in particular, the average number of services for bid is typically lower than 2. We used CATS (see Section 6.4.1) to generate more realistic instances with a very different structure one another. With CATS we can specify the minimum, maximum and average number of services for bid and the variance of the average: to do that, when creating a bid, the minimum number of services is added and than any subsequent service is added with a certain probability until possibly the maximum number is reached. The probability can be fixed or can increase or decrease each time a service is added.

Unfortunately, CATS produces only WDP instances, i.e. bids without temporal information: we overcome this limitation generating WDP instances using CATS, producing BEP instances with the same number of services using MAGNET and finally extracting the temporal information from MAGNET instances and including them in the CATS instances.

We generated a large variety of realistic instances with a number of services ranging from 15 to 30 and a number of bidders ranging from 400 to 1000.[4].

7.4.2 Comparing IP and CP

In this section we will show and compare the results when solving the instances described above with our solvers (IP, LR+IP, CP and HCP). We do not consider MAGNET having shown, in section 7.2, that the CP algorithm always outperforms it. We ran our experiments on a 2.4GHz Pentium 4 with 512MB RAM and using ILOG Solver 5.3 [65] as CP solving tool and ILOG CPLEX 8.1 [63] as IP solving tool using its default parameters except for the variable selection heuristic.

In Table 7.3 we compare the four algorithms on instances with 15 services and 500 bids, 20 services and bids varying from 400 to 1000 and 30 services and 1000 bids. Each line shows the mean search time over a set of 10 instances. We also

[4]These instances are available on the web at http://www-lia.deis.unibo.it/Staff/AlessioGuerri/BEP_LIB

report the mean services for bid (S/B) value shown in the column S/B, with a variance of 1%, that will be useful in later discussions. Columns CP, HCP, IP and LR+IP show the search time (in milliseconds) to solve the problems to optimality. The symbol '-' means that the optimal solution was not found within a time limit of 15 minutes. Column r% represents the percentage of variables considered in the LR+IP incomplete algorithm.

Services	**Bids**	**S/B**	**Search time (ms)**				**r%**
			CP	**HCP**	**IP**	**LR+IP**	
15	500	2.59	8539	11475	7423	1772	40
15	500	4.29	590	523	16740	1765	20
15	500	7.57	874	720	12022	9782	30
20	400	2.76	4118	5898	2754	357	40
20	500	4.69	1794	1694	-	56822	55
20	800	4.58	16453	9334	359437	21658	50
20	1000	1.12	13688	17063	1610	281	65
20	1000	1.15	-	-	687	360	95
20	1000	4.49	3085	2082	-	9319	20
30	1000	1.40	-	-	36328	1235	70
30	1000	3.34	-	-	900000	6975	25
30	1000	6.52	-	-	-	25969	30

Table 7.3: Comparison between algorithms

Some consideration follows from Table 7.3.

- When CP outperforms HCP, both IP and LR+IP outperform HCP and CP, so we can remove CP from the portfolio of algorithms because the best one always lies between IP, LR+IP and HCP.

- LR-IP is faster w.r.t. IP but it always happens that both of them outperforms HCP or are both outperformed by HCP, so we can remove LR+IP from the portfolio and consider only the complete algorithms IP and HCP.

We take for grant that, if we are interested in incomplete algorithms, it is possible to obtain them from HCP by limiting the maximum number of discrepancies allowed, and from IP simply considering the LR+IP algorithm.

7.5 Algorithm Portfolio

In this Section we analyze the algorithm behaviours when solving BEP instances with different S/B values. Table 7.4 shows the comparison of the two algorithms (HCP and IP) search times on instances with 20 services and 400 bids. Each line represents the mean over 10 instances having the same S/B value (reported in the columns S/B). All the instances have been solved to optimality by both the algorithms.

S/B	**HCP**	**IP**	**S/B**	**HCP**	**IP**
2,291	4391	750	2,497	10750	797
2,563	2828	907	2,705	5468	844
2,777	5468	844	2,846	12940	828
3,111	731	937	4,179	344	2406
4,356	641	2281	4,446	343	1453
6,935	250	9047	7,086	625	3563
7,181	5719	8282	7,696	407	8032
7,795	78	11047	7,876	63	5485
7,878	78	8891	8,052	235	12890

Table 7.4: Comparison between algorithms for instances of 20 services and 400 bids

We can observe a correlation between the S/B value and the best algorithm: in particular we notice that the higher the S/B value, the better the HCP approach performances. This result was expected because an higher S/B value leads to an higher number of side constraints complicating IP model and, at the same time, reducing the number of feasible solutions in the CP model.

We have two algorithms and none of them dominates the other over all problems, and we can see a correlation between the instance structure and the best solving approach. The algorithms are therefore good candidates for an algorithm portfolio design where the best algorithm can be recognized *a priori* analyzing the instance structure.

We used the S/B value to guide the selection of the best algorithm between IP and HCP and we tested the accuracy on a test set of 280 instances. We found that the prediction of the algorithm is correct in the 72% of the cases. This result is encouraging but the error rate is unacceptable, and moreover this analysis can be hardly generalized if other algorithms or different problems will be considered. In fact, the S/B value is a good indicator in this particular case because optimization and feasibility hardness are directly affected by this value. Let us think what happens removing the temporal constraints, thus obtaining the WDP. The S/B value remains unchanged but now we know that the best algorithm is always IP.

Our claim is that, given an instance, it is possible to select the best algorithm by statically analyzing the structure of the instance itself, and our goal is to develop an automatic tool to discern the best algorithm on the basis of the instance structure, but in order to obtain a generalizable tool we need to find a more accurate description of the instance structure other than the S/B indicator.

Next Chapter is devoted to the algorithm portfolio analysis and the automatic selection tool development.

Chapter 8

Algorithm Portfolio Analysis

Introduction

In this Chapter we will describe in deep the algorithm portfolio approach to solve the BEP and the tool to automatically select the best algorithm. In Section 8.2 we will introduce the portfolio and we will list the features, describing a BEP instance, we will base the selection tool on. In Section 8.4 we will introduce the tool we have developed to select the best algorithm before starting the search and we will show some experimental results. In Section 8.6 we will give evidence that our selection tool is flexible and extensible by testing it on BEP and WDP instances with differentiated structural characteristics.

8.1 Algorithm Portfolio

As introduced in Section 4.2, the Algorithm Portfolio (AP) is a method to integrate different programming paradigms. A portfolio of algorithms is a collection of different algorithms or several instances of the same algorithm, differing in some parameters or in the initial state, able to solve the same problem: to find a solution, a common AP approach is to run all the algorithms (or a subset of them) in parallel on different processors. When the fastest one finds the solution, all the others are stopped. Of course, an AP containing different algorithms pays off when none of them dominates all the others.

An AP can also embed several repetitions of the same algorithm. This is typical when using, for example, stochastic algorithms, where the result depends on the initial state (defined by a random seed) of the search. In this case the AP runs several instantiation of the same algorithm starting with different seeds, or combines several short runs of the same algorithm. The latter technique is called *restart* for stochastic algorithms [36]. In [49] the authors provide results showing that, for some classes of problems, an AP approach implementing the restart technique can lead to strong computational advantage.

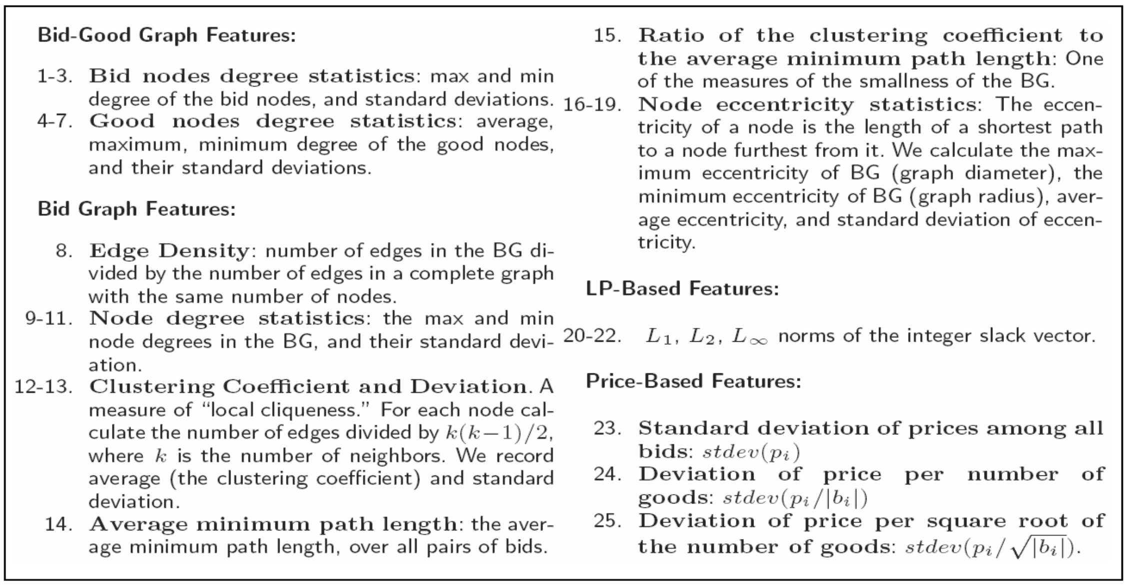
The restart technique is also effective if we are interested in finding a solution quickly. The first algorithm runs searching for a solution and when a given event, for example the maximum running time or the maximum number of nodes explored, occurs, the search is stopped and another algorithm is tried [92].

All the AP techniques described above run several different algorithms or different instantiations of the same algorithm, discovering the best algorithm only at the end of the search. Furthermore, running several algorithms in parallel requires a higher computational effort. Finding, for each problem instance, the best algorithm before the search can dramatically reduce the computational requirements of the portfolio providing the same results both in terms of solution quality and search time. Of course, finding the best algorithm given an instance, is not a trivial task. The selection must be based on information taken from the static analysis of the instance. One of the most distinctive characteristics of an instance is its structure. Starting from an instance model, several representations of the structure can be extracted: flow graph, constraint graph, parameter list.

In our research we have developed a portfolio of algorithms to solve the BEP, and an automatic approach based on machine learning (see Section 8.3) for selecting, given an instance, the best algorithm. The selection is based on information extracted from the constraint graph associated to the problem instance. In the following Section, we will introduce the constraint graph and some parameters that can be extracted.

8.2 The instance structure

In the last Chapter we have analyzed some algorithms to solve the BEP, and two of them are not dominated by the others on all the instances considered. As far as the Algorithm Portfolio (AP) analysis is concerned (see Section 8.1) they are good candidates to be included in a portfolio. Our idea is to run, for each problem instance we face, only the fastest algorithm: we therefore need to find some structure based characteristics able to discern a BEP instance best solved by HCP rather than IP and viceversa. We based our analysis on a list of 25 features presented in [85] to discern a WDP instance hardness.



Bid-Good Graph Features:

1-3. **Bid nodes degree statistics**: max and min degree of the bid nodes, and standard deviations.
4-7. **Good nodes degree statistics**: average, maximum, minimum degree of the good nodes, and their standard deviations.

Bid Graph Features:

8. **Edge Density**: number of edges in the BG divided by the number of edges in a complete graph with the same number of nodes.
9-11. **Node degree statistics**: the max and min node degrees in the BG, and their standard deviation.
12-13. **Clustering Coefficient and Deviation**. A measure of "local cliqueness." For each node calculate the number of edges divided by $k(k-1)/2$, where k is the number of neighbors. We record average (the clustering coefficient) and standard deviation.
14. **Average minimum path length**: the average minimum path length, over all pairs of bids.
15. **Ratio of the clustering coefficient to the average minimum path length**: One of the measures of the smallness of the BG.
16-19. **Node eccentricity statistics**: The eccentricity of a node is the length of a shortest path to a node furthest from it. We calculate the maximum eccentricity of BG (graph diameter), the minimum eccentricity of BG (graph radius), average eccentricity, and standard deviation of eccentricity.

LP-Based Features:

20-22. L_1, L_2, L_∞ norms of the integer slack vector.

Price-Based Features:

23. **Standard deviation of prices among all bids**: $stdev(p_i)$
24. **Deviation of price per number of goods**: $stdev(p_i/|b_i|)$
25. **Deviation of price per square root of the number of goods**: $stdev(p_i/\sqrt{|b_i|})$.

Table 8.1: BEP instance features

Table 8.1, taken from [85], lists the features. They are divided in four groups. The first group contains features extracted from the Bid-Good graph, a bipartite graph associated to a combinatorial auction problem with a node for each bid, a node for each service, and an edge between a bid and a service if the service is proposed by the bid; Figure 8.1(a) shows the Bid-Good graph for the example reported in Table 6.1 of Chapter 6. The second group is extracted from the Bid graph, that is the constraint graph of the BEP instance. This graph represents conflicts among bids: each node represents a bid and an edge between a couple of bids exists if the two bids appear together in one or more constraints. Figure 8.1(b) shows the Bid graph for the example in Table 6.1. The third group of features is based on the slack values of the linear relaxation of the problem; the slack values vector is calculated starting from the solution vector x of a LR of the problem and replacing each element x_i with $min(|x_i|, |1 - x_i|)$. The last group is based on the bid prices.

Since the BEP contains the WDP as a sub-problem, we based our AP analysis

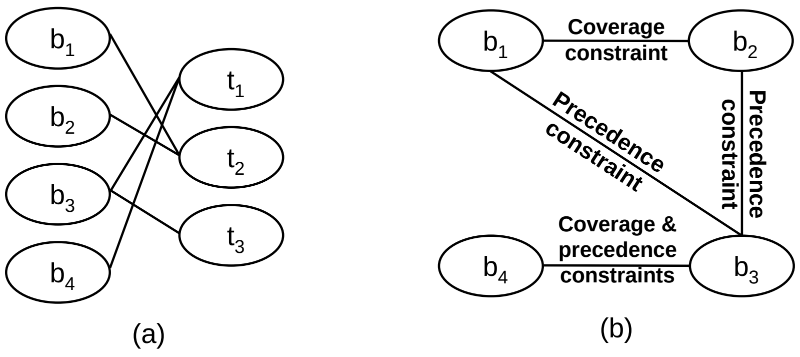


Figure 8.1: Bid-Good graph and Bid graph for the example in Table 6.1

on the same features. We extracted these 25 features from our BEP instances. Clearly it is not possible to manually select those attributes that are correlated to the instance structure. We need an automatic way to decide, given the list of values describing an instance, the best algorithm to solve it. We exploited the Decision Trees Machine Learning technique, in particular the software tool c4.5 [106], using the 25 features to describe the instances and find a classification rule.

8.3 Decision Trees

The machine learning technique for inducing a decision tree from data is called Decision Tree Learning (DT). DT is a predictive model; that is, a mapping of observations about an item to conclusions about the item's target value. DT conceptually creates a tree where each interior node corresponds to a attribute of the item; an arc to a child represents a possible value of that attribute. A leaf represents the predicted value of target variable given the values of the variables represented by the path from the root. Observations are provided as training set in form of attribute-value tuples, with the corresponding target value, the class.

At each node, the method recursively selects an attribute and divides the cases in subsets, one for each branch of the test until all cases in each subset belong to the same class or a stopping criterion is reached. At each node, the attribute selected for the test is the one that maximizes the information gain. The information gain is based on the notion of entropy. Intuitively, the entropy of a set of examples is the information about the non-uniformity of the collection itself. The entropy is minimum when all cases belong to the same class, otherwise the entropy is maximum when cases are uniformly classified over all possible classes. The entropy of a collection of cases S is defined as follows:

$$Entropy(S) = \sum_{j=1}^{k} \left(\frac{freq(C_j, S)}{|S|} \times \log_2 \left(\frac{freq(C_j, S)}{|S|} \right) \right)$$

where $freq(C_j, S)$ is the number of cases in S belonging to class $C_j,\ j = 1 \ldots k$.

Beside the information gain, other measures can be used, such as the Gini index [47] (a measure of the inequality of a distribution) or the misclassification measure. Once the decision tree has been computed, it is important to test its accuracy in classifying new instances. For this purpose, a test set is defined. A test set has the same structure of the training set. In this way, we can establish which is the error rate and which is the accuracy of the decision tree when analyzing unseen cases.

One important aspect of DT is the ability to generalize the results. In general, only a small subset of the attributes provided in the data set are indeed used in the resulting decision tree. Small trees, involving a small set of attributes, are preferred to large trees.

In our research we have used c4.5 [106], a decision tree learning system. c4.5 creates a decision tree as explained above and then prunes it finding a tree containing only those features that lead to a minimum of the entropy of the training set, therefore decision trees generated by c4.5 are in general very small.

The DT technique has several advantages: results are simple to understand and interpret; require little data preparation avoiding, for example, normalization; are robust and perform well with large data in a short time.

8.4 Decision Trees experimental results

We considered a data set of 400 instances, split in training set (70%) and test set (30%). Each instance in the training set is described by the best algorithm (either IP or HCP) to solve it and by the value of the 25 static features defined in [85] and listed in Table 8.1.

We built the decision tree using the training set, and we verify the quality of the resulting systems using the test set. We record the percentage of cases that are classified correctly. To avoid the generation of decision trees depending from the particular training and test sets used, we repeat the analysis randomly splitting the data set 10 times into different training and testing sets. We obtained ten trees with the same structure, i.e., at each level the same attribute is chosen for splitting the training set. Since we have continuous parameters, the ten trees slightly differ only in the threshold used for splitting. Therefore, the resulting tree has thresholds that are the mean of the thresholds computed in the ten experiments.

8.4.1 Constraint graph and features

As introduced in the previous Section, c4.5 generates a decision tree using only those features that lead to a maximum information gain. We describe here three features, out of the 25 presented in Table 8.1, that will appear in the decision trees generated by c4.5.

These features are extracted from the constraint graph, that is a graph with a node for each variable and an edge between each couple of nodes representing variables involved in one or more constraints.

Starting from this graph, we can extract the following structural features:

- **Standard Deviation of the Node Degree (ND)**: the ND is the number of edges starting from a node. Once collected in a vector this value for all the variables, the ND is the standard deviation of the vector. Given a set of numbers, their Standard Deviation gives the valuation of how scattered they are around their mean value. Given a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, and the mean of all its values $\bar{x}$, the Standard Deviation of the vector $\mathbf{x}$, σ_x, is defined as follows:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

  The ND for the graph in Figure 8.1(b) is the standard deviation of the vector $x = \{2, 2, 1, 3\}$, that is $\frac{1}{\sqrt{2}}$.

- **Edge Density (ED)**: the ED is the ratio between the number of edges in the graph and the number of edges in a complete graph with the same number of nodes. If k in the number of nodes, $k(k-1)/2$ is the number of edges in the complete graph. This value can range from 0 to 1. The ED for the graph in Figure 8.1(b) is $\frac{4}{6} = 0.\bar{6}$.

- **Clustering Coefficient (CC)**: the CC is a measure of the *local cliqueness* of the graph. It is computed as follows: for each node N we consider its neighborhood, formed by the nodes in the graph directly connected to N, and we compute the ED of the neighborhood. We compute this value for each node and the CC is the mean value. This parameter can range from 0 to 1. Let us consider the Figure 8.1(b): the neighborhood of b_1 is the subgraph formed by b_2 and b_3, that are connected in their turn by an edge. In this case the ED is 1. Analogously, the ED for the neighborhood of b_2 is 1, while for b_4 it is 0. The neighborhood of b_3 contains all the other three nodes and the subgraph contains only one edge. The ED is therefore $\frac{1}{3}$. The CC of the graph in Figure 8.1(b) is the mean value of $(1, 1, 0, 0.\bar{3})$, that is $0.58\bar{3}$.

We can see that extracting the ND and the ED is quite simple, in fact we have to traverse the graph extracting a simple value from each node and to apply a simple formula. Extracting CC is, on the contrary, more complex. We have to visit each node in the graph extracting the neighborhood, and then we have to traverse each neighborhood. This reflects on the computational time required to calculate the CC values.

8.4.2 Experimental results

We performed our analysis starting from a training set of 280 tuples with 25 attributes and we obtained a decision tree (referred to as DT-1) with a prediction error equal to 9%. The resulting tree has only one level consisting in a test on the Clustering Coefficient (CC). This means that for each new instance we need to extract only the CC to decide which algorithm best solves the instance itself.

Unfortunately, for some instances the CC is very expensive to be computed. In particular, for those instances with high S/B value the CC is very expensive, because an higher S/B value leads to an higher number of constraints in the model and of edges in the Bid graph, and therefore and higher features extraction time. We therefore removed from the training set the expensive features and we performed again the experiments. As a result, we get a decision tree (referred to as DT-2) with a prediction error equal to 11%. The features considered significant by c4.5 are only the Edge Density (ED) and the Standard Deviation of the Node Degree (ND).

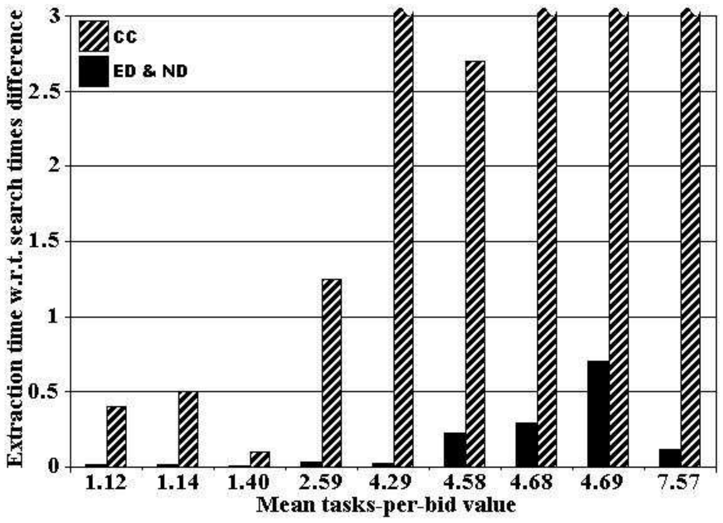


Figure 8.2: Extraction time for attributes

Figure 8.2 shows, for groups of instances with similar S/B values, the ratio between the feature values extraction times and the difference between the search times of the best and the worst algorithm. When the ratio is greater than 1 it is not worth extracting the feature since the time used in selecting the best algorithm plus the time to solve the instance using it is greater than the time used by the worst algorithm to solve the instance. The threshold above which extracting CC is not convenient has been fixed to 2. We can also see from the Figure that the ratio between the ED and the ND extraction time and the difference between the search times of the best and the worst algorithm is always lower than 1, thus it is always worth using DT-2 (except for the fact that the error rate is a bit higher w.r.t. DT-1).

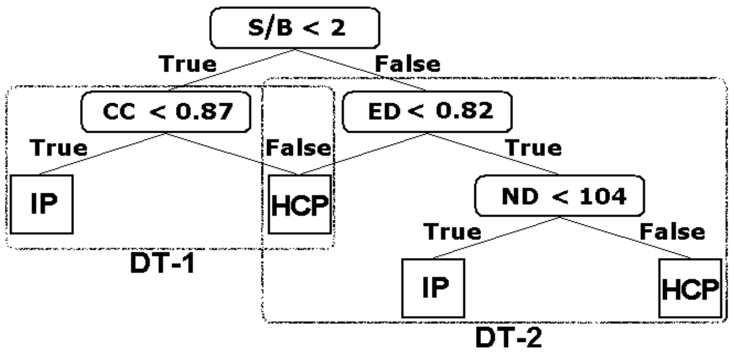


Figure 8.3: Decision tree for the BEP

The whole decision tree for the BEP is depicted in Figure 8.3. The left most sub-tree is DT-1, while the right most is DT-2. This tree can be easily translated in an automatic selection algorithm, reported in Figure 8.4.

```
if (S/B < 2)
   then if (CC < 0.87)
           then IP
           else HCP
   else if (ED < 0.82)
           then if (ND < 104)
                   then IP
                   else HCP
           else HCP
```

Figure 8.4: Selection algorithm for the BEP

These results support our claim that structure at the instance level may be enough to discriminate among solution strategies.

8.4.3 Time saving using Decision Tree Technique

We have shown in the last subsection that, using some instance structure based parameters, we can achieve a prediction rate of about 90%; on the other hand, using only the S/B parameter (S/B in the following), which extraction time is negligible, we have a prediction rate of 72%. It is therefore important to verify whether the prediction rate improvement achieved extracting the ED, ND and CC pays back with a reduction of the total search time (parameter extraction plus solution search).

We first analyzed the 10% of the instances incorrectly classified by DT. We found that all but two of them are incorrectly classified also by S/B; in the two instances where S/B predicts the right choice and DT does not, the search time of the worst algorithm is only one order of magnitude higher w.r.t. the best. On the contrary, analyzing the instances correctly classified only by DT (about 18% of our data set) we found that the 53% of them can be solved only by the best algorithm and in the remaining cases the worst algorithm search time is up to two order of magnitude w.r.t. the best one.

These results show that DT not only achieves an higher prediction rate, but can also predict the right algorithm for instances where a wrong choice causes a much higher solving time up to, in the worst case, the time deadline. As a further analysis we split the ratios between the parameters extraction time and the algorithm search time difference shown in Figure 8.2 for the cases correctly classified by S/B and DT, correctly classified only by DT and where both the approaches miss the classification. Table 8.2 summarizes the results. The first two columns report whether the approaches correctly classifies the instances or not, the third columns report the percentage of cases falling in the row, while the last two columns report the ratio between the parameters extraction time and the search time difference. The extraction time is calculated only for the instances where the parameter is used for classification (thus CC is calculated if S/B is lower then 2 and ED and ND otherwise). We consider in Table only instances solved by both the algorithms.

Approach			**Parameters**	
S/B	**DT**	**%**	**ED+ND**	**CC**
Yes	Yes	72%	1.40%	11.66%
No	Yes	18%	0.98%	13.72%
No	No	10%	1.66%	16.80%

Table 8.2: Comparison between classification approaches

We can see that, using the DT approach, we have a search time increase of 1.4% or 11.66% (depending on the parameters used) on the instances correctly classified by S/B, but we have a search time reduction of 99.02% or 86.28% on the instances where S/B fails. In addition, we have considered only the instances where both the algorithms can find a solution, so the time saved in the latter case is actually higher. The last row shows that, on the instance we can not classify we waste the 1.66% or 16.8% of the time extracting the parameters.

Summarizing the results presented in Table 8.2 we can see that, using the ED and ND parameters we have a time wasting of 1.4% in the 72% of cases and 1.66 in the 10%, but we have a time saving of 99.02% in the 18% of the cases. The mean of these values gives a time saving of 16.65% using the ED and ND parameters. The same value for the CC is 5.45%. Given that, in the instances considered, the S/B is lower then 2 in the 47% of the cases, we can summarize these vales stating that, using DT, we have a time saving of 11.39%. In addition, we remind again that we considered only the instances where both the algorithm can find a solution, so the percentages presented are lower than the real time saving value.

8.5 Comparison with other learning techniques

To further validate the strength of our approach, we compared Decision Trees with other learning techniques. namely Case Based Reasoning (CBR) [75] and Binomial Logistic Regression (BLR) [119], to select the best algorithm. In the following, we describe these techniques and we show the experimental results.

8.5.1 CBR Framework

CBR enables past problem solving experiences to be reused to solve new problems [75]. CBR has been successfully used in the context of, e.g. diagnosis and decision support [81], design and configuration [26]. CBR is based on the intuition that if two instances are similar, then it follows that the same technology should be appropriate for both instances.

Experiences are stored along with the problems they solve as *cases*. A case is a representative example of a group of instances (a group can contain any number of instances) that are similar to one another, but different from other groups of instances. A particular technology (CP or HCP for this analysis), is associated with one or more groups of instances i.e. cases.

A case is composed two parts; a problem part and an *experience* part. In the present context, the problem part consists of some information about a BEP instance in the form of a set of descriptive features. The experience part is what technology between HCP and IP solves that instance efficiently.

Two important decisions must be made in the design of a CBR system. Firstly how should problems be represented and secondly how should similarity between problems be computed. In this work, the set of features i.e. how to represent problems is decided for us. We use a similarity measure that computes the Euclidean distance between the features used to represent the two problems being compared.

A CBR system consists of a four step cycle; *retrieve*, *reuse*, *revise*, and *retain*. To solve a new problem, we *retrieve* a case from the casebase, whose problem part is most similar to the new problem. We then *reuse* the experience part to recommend what technology should be used to solve the new problem. The casebase may be *revised* in light of what has been learned during this most recent problem solving episode and if necessary the retrieved experience, and the new problem, may be *retained* as a new case. Every time a new problem instance is presented to the system, this cycle enables a CBR system to both learn new experiences and maintain or improve its ability to predict.

A CBR system typically must be trained before it can classify new instances. To facilitate training we divide our dataset into two sub-sets for training and testing purposes. In training mode, a casebase is assembled using the training problems. We expect that the instances retained in the casebase constitute examples of when to use the appropriate technology.

8.5.2 Binomial Logistic Regression

In [85], the authors use regression (linear and quadratic) to learn a real-valued function of the features that predicts the runtime. In [83], they extend their approach to allow them to predict an algorithm (among a given pool) that best solves a given instance. They do this by firstly predicting the runtime for each algorithm (using the approach outlined in [85]). This requires a different set of dynamic features for each algorithm. Then they simply choose the algorithm whose predicted runtime is smallest.

Using linear or quadratic regression for our task is inappropriate however. Unlike [85], where the dependent variable, the runtime, of the regression is continuous, here the dependent variable is discrete; either 0 for IP or 1 for HCP. Hence we must utilize a different form of regression called *Binary Logistic Regression* (BLR).

The logistic regression model has become the standard method of data analysis for describing the relationship between a discrete dependent variable that takes one of two values and one or more predictor variables. Logistic regression is used to model the probability p of occurrence of a binary or dichotomous outcome.

BLR is used to find a fit for data when the dependent variable (the choice of solution strategy in the present context) has two discrete values mapped to 0 or 1 [119]. Logistic Regression uses the *Logit model* to determine the relationship between the probability of an outcome occurring and the predictor variables (problem features in the present context).

8.5.3 Weighted Random

The weighted random selection technique looks at how often each solution strategy is the best for all instances in each dataset, and builds a corresponding frequency distribution for that dataset. The probability to suggest a strategy is based on this frequency distribution. In other words, the strategy that in the dataset is most often the best strategy has an higher probability to be suggested and so on for all the remaining strategies.

8.5.4 Experimental Results

In this section we will show the results obtained when predicting the best algorithm using the techniques described above. We used the same dataset described in 8.4 and, to directly compare the results, we consider only the three features described in 8.4 (CC, ED and ND) to describe our BEP instances.

- **CBR results**: To account for the fact that some features are more important than others, we introduce weights to control the importance of each feature when computing the similarity between two instances. Our CBR system must search for the best combination of feature weights. It does so by exhaustively trying all combinations of weights for all features. We exhaustively cycle through all possible weights for every feature from 0.00 to 1.00 in steps of 0.05 and thus find the optimal combination of weights for the features used. Note that we are able to afford this exhaustive search because we only consider 3 features. A more sophisticated weight learning method is required once the number of features grows much beyond 3.

  We use *weighted city block* as the similarity measure. Thus our similarity measure for two instances $\vec{p_p}$ and $\vec{c_i}$, and a set of problem features $pf = \{a, b\}$ for each; $\vec{p_p} = \langle a_1, b_1 \rangle$, $\vec{c_i} = \langle a_2, b_2 \rangle$ looks like this:

  $$sim(\vec{p_p}, \vec{c_i}) = (w_a(1 - \frac{|a_1 - a_2|}{a_{max} - a_{min}}) + w_b(1 - \frac{|b_1 - b_2|}{b_{max} - b_{min}}))/\{w_a + w_b\}$$

  Occasionally, the similarity between a given instance and some cases is the same. That is we have a tie for the best case. In this situation we decide which case to use from this set of best scoring cases by a majority voting scheme and break ties randomly.

The median prediction rate averaged over 10 trials is around 85% (with a median of 84% and standard deviation of 5.3%). The weights for the three features are 0.05 for ED, 0.1 for ND, and 0.21 for CC.

- **BLR results**: We use the Zelig library [66], an add-on to the *R* statistical package [62] to carry out the BLR calculations.

  With BLR, we obtain a 79% prediction rate averaged over 10 trials with a median of 79% and standard deviation of 5.7%.

  Table 8.3 compares the results obtained using the techniques described above.

Technique	DT-1	DT-2	CBR	BLR	WR
Prediction rate	91%	89%	85%	79%	49%

Table 8.3: Machine learning techniques comparison.

These results show that different machines learning techniques have different performances even within the same domain and the same dataset and, for the instances considered, Decision Trees seem to be the best approach.

Our analysis suggests that, using the decision trees technique in the context of the BEP, we are able to build a practical and useful system that is able to select the best solution strategy within a reasonable prediction rate using only static and cheap features. Static features make it possible to have an off-line prediction system that does not rely on runtime knowledge.

In the next Section we will show that our tool is flexible when used in the context of the BEP and is extensible to the WDP.

8.6 Modifying and eliminating the time windows

In Section 7.5 we noted that the number of temporal constraints between services affects the size of the IP model more than the CP model. We can explain this fact considering the precedence graph associated to the BEP. In the CP model, we introduce a constraint for each edge in the precedence graph, while in the IP model we introduce a constraint for each pair of services joined by a precedence constraint and for each pair of bids proposing them.

The latter constraints can be of different kinds, depending on the time windows. In fact, as described in section 6.3.1, considering two services i and i', appearing respectively in bids j and j', joined by the precedence constraint $i \prec i'$, three different situations regarding time windows can occur: all values in the windows are compatible, so we do not need to add any constraint; only some values are compatible, so we need to add the constraint $Start_{ij} + D_{ij} - Start_{i'j'} + M(x_j + x_{j'}) < 2M$ (6.10); all values in the windows are incompatible each other, so we need to add the constraint $x_j + x_{j'} \leq 1$ (6.9). The first constraint involves the $Start$ variables and uses the Big-M method, while the second is a very simple constraint involving only two decision variables: the first kind of constraints complicates the model more than the second, so depending on the temporal windows we can have different models with different computational complexity.

The instance structure, and thus the feature values, are therefore affected not only by the services-for-bid value, as mentioned in Section 7.5, but also by the width of the temporal windows associated to the services.

To further validate the strength of our approach, and in particular to show that it is extensible to problems having a different structure, we applied our selection tool to different instances of the same BEP, obtained widening the time windows, and to WDP instances, that can be seen as BEP instances having the time windows infinitely widened.

We tested the selection algorithm in Figure 8.4 widening the time windows of a set of instances. Starting from each instance we generated eleven different instances enlarging the time windows ten percent each time, up to doubling them. After this point, further enlarging the time window has the same effect obtained removing the time window. To enlarge time windows, we fixed the central point of each time windows and we simply enlarged its width by a coefficient ranging from 1.0 to 2.0 with a step of 0.1, consequently modifying Est_{ij} and Lst_{ij}, $\forall j = 1 \ldots n, \forall i \in S_j$. We extracted the ED, the ND and the CC from each instance with each time windows modification and we solved them with the algorithms in the portfolio. In Table 8.4 we show, for four instances with 20 services, 400 bids and different S/B values, reported in the first column, the ED, ND and CC value and thus the predicted algorithm, as well as the search time in seconds for the two portfolio algorithms, namely HCP and IP. Each column represents the instances with the time windows enlarged using the coefficient in the first row. We can see that there is a perfect correspondence between the predicted and the best algorithm (in bold), except for the third column of the instance 2.95, where the prediction is wrong. We can however see that in this case the two algorithms behaviours are very close one each other. In general, from the experimental results, we have seen that, when the decision tree misses the prediction, the algorithm behaviours are quite similar.

S/B		1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2	∞
	ED	0.196	0.113	0.112	0.110	0.109	0.108	0.107	0.106	0.105	0.104	0.103	0.069
	ND	29.367	19.028	18.849	18.649	18.425	18.270	17.986	17.838	17.731	17.548	17.438	9.956
	CC	0.736	0.787	0.787	0.787	0.787	0.788	0.788	0.788	0.789	0.790	0.791	0.818
1.07	Pred	IP	IP	IP	IP	IP	IP	IP	IP	IP	IP	IP	IP
	HCP	27.52	100.44	262.53	291.35	289.25	20.20	24.84	70.11	176.05	4.33	1.11	0.06
	IP	**1.39**	**1.77**	**7.09**	**19.80**	**3.36**	**7.09**	**17.23**	**5.25**	**3.61**	**0.84**	**0.94**	**0.01**
	ED	0.863	0.831	0.821	0.811	0.804	0.800	0.798	0.795	0.793	0.791	0.789	0.691
	ND	59.481	65.875	66.265	66.558	66.844	67.338	67.592	67.871	68.140	68.357	68.547	74.158
	CC	0.904	0.886	0.880	0.862	0.860	0.858	0.856	0.854	0.853	0.851	0.850	0.783
2.95	Pred	HCP	HCP	**HCP**	IP	IP	IP	IP	IP	IP	IP	IP	IP
	HCP	**43.64**	**26.45**	24.17	21.73	24.11	15.06	15.89	14.25	16.08	11.94	12.66	0.19
	IP	52.38	39.81	**21.14**	**7.70**	**6.05**	**5.14**	**5.09**	**6.44**	**6.94**	**8.30**	**7.59**	**0.05**
	ED	0.865	0.835	0.831	0.828	0.825	0.821	0.818	0.815	0.812	0.809	0.807	0.679
	ND	108.633	114.558	115.215	115.826	116.387	116.941	117.415	118.024	118.444	118.902	119.388	121.710
	CC	0.914	0.900	0.898	0.896	0.894	0.891	0.889	0.887	0.885	0.883	0.881	0.778
4.80	Pred	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	IP
	HCP	**4.32**	**4.01**	**3.34**	**3.42**	**5.41**	**4.41**	**3.42**	**3.43**	**4.44**	**5.45**	**3.05**	2.65
	IP	9.83	12.08	18.03	31.69	44.02	54.80	45.41	48.74	42.89	49.67	56.33	**0.09**
	ED	0.912	0.905	0.904	0.903	0.902	0.901	0.900	0.899	0.898	0.897	0.896	0.798
	ND	58.444	60.228	60.516	60.901	61.193	61.548	62.016	62.356	62.660	62.968	63.293	78.705
	CC	0.972	0.941	0.940	0.939	0.939	0.938	0.938	0.937	0.937	0.936	0.936	0.854
7.09	Pred	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	HCP	IP
	HCP	**4.20**	**7.97**	**16.84**	**49.06**	**1.74**	**2.30**	**5.22**	**5.84**	**12.19**	**14.89**	**19.58**	2.76
	IP	26.52	75.77	142.52	243.45	219.75	191.30	101.38	90.44	44.92	39.00	44.22	**0.37**

Table 8.4: Algorithm's comparison in instances with 20 services and 400 bids modifying time windows

We can note that the HCP behaviours when solving an instance with different time windows remain quite similar one each other, while, regarding the IP behaviours, we note an higher variability. This result was expected because the number of constraints, and so the computational hardness, in the CP model does not depend on the time windows width but only on the number of services involved in the precedence graph. On the contrary, in the IP model, different time windows lead to different constraints in the model (in fact we can have the constraints (6.9) or (6.10) depending on the time windows overlapping).

In the last column of Table 8.4 (column ∞) we report the search time widening the time windows to infinity, thus when solving the WDP, and we can see that also for the WDP the algorithm described in Figure 8.4 suggests the right solver, that is always IP.

In [85] the authors found that ED and CC are the most appropriate features to derive the computational hardness of a WDP problem, and here we show that these parameters can also discern the best algorithm, both for BEP and WDP. We have also shown that, using these features, it is possible to create a flexible and extensible automatic selection tool. In fact, adding new algorithms to the existing pool does not increase the number of considered features and modifying the instance structure the tool keep on suggesting the right solver.

Part III

The Allocation and Scheduling Problem on a Multi-Processor System-on-Chip

Chapter 9

Introduction

This Part of the dissertation is devoted to analyze allocation and scheduling problems rising in the context of Multi-Processor System-on-Chip (MPSoC) platforms, quickly described in Section 1.2.2. The structure of the problems has the interesting characteristic that it is possible to recognize two distinct sub-problems in it, the allocation and the scheduling problem. This suggests to apply a decomposition technique to solve the problem. We exploited logic-based Benders Decomposition (see Section 9.4); we decompose the problem into Allocation Master problem and Scheduling Sub-Problem, solving the former using IP and the latter using CP.

In the following chapters we will analyze two problems rising in the context of MPSoCs, namely the Allocation and Scheduling Problem (ASP) and the Dynamic Voltage Scaling Problem (DVSP) and we will describe the decomposition method we used, comparing the results with those obtained by solving the problem as a whole using only a single programming paradigm, either IP or CP, and validating our results on a real platform.

The research described in this Part of the dissertation supports the thesis that

. . . ***The structure can also suggest to develop solvers based on both the approaches, or to split the problem in two or more subproblems solved with different approaches.***

In this chapter we will describe the MPSoC platform we consider and MP-ARM, a multi-processor cycle-accurate architectural simulator we used to validate our experimental results. We will conclude giving a general overview of the problems we face and describing the Decomposition Techniques, we will take advantage of for our research.

9.1 The Multi-Processor System-on-Chip platform

Advances in very large scale integration (VLSI) of digital electronic circuits have made it possible to integrate one billion of elementary devices on the same chip, and currently integrated hardware platforms for high-end consumer application (e.g. multimedia-enabled phones) can contain multiple processors and memories, as well as complex on-chip interconnects. Integrating more than one processing core on the same silicon die can provide a high degree of flexibility and represents the most efficient architectural solution for supporting multimedia applications, characterized by the request for highly parallel computation.

These platforms are called Multi-Processor Systems-on-Chip (MPSoCs), and are finding widespread application in embedded systems (such as cellular phones, automotive control engines, etc.). Once deployed in field, usually these devices always run the same application, in a well-characterized context.

The multi-processor system we considered in our research is a reference template [112] for a distributed MPSoC architecture and consists of a pre-defined number of distributed computation nodes, as depicted in Figure 9.1. All nodes are assumed to be homogeneous and consist of ARM7 processor cores, low-power 32-bit RISC microprocessor cores (including instruction and data caches) optimized for cost and power-sensitive consumer applications, and of tightly coupled software-controlled scratchpad memories. This latter is a low-access-cost *scratchpad memory*, which is commonly used both as hardware extension to support message passing and as a storage means for computation data and processor instructions which are frequently accessed. Data storage onto the scratchpad memory is directly managed by the application, and not automatically in hardware as is the case for processor caches. The MPSoC platform embeds ARM7 processors.

Being the scratchpad memory of limited size, data in excess can also be stored externally in a remote on-chip memory, accessible via the bus. The bus for state-of-the-art MPSoCs is a shared communication resource, and serialization of bus

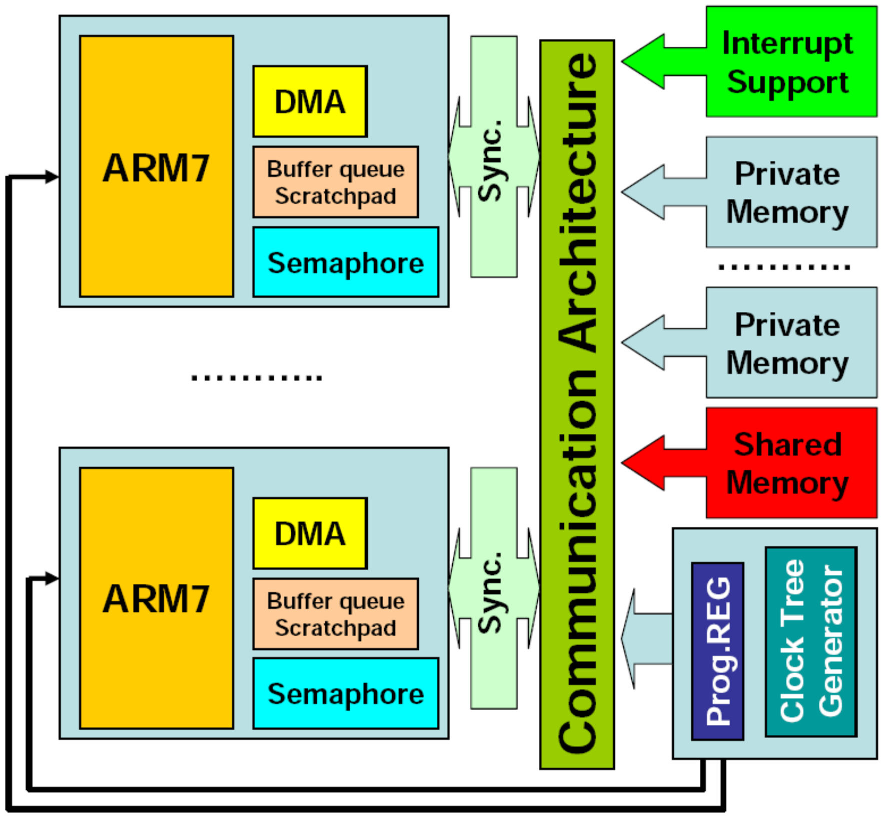


Figure 9.1: Single chip multi-processor architecture

access requests of the processors (the bus masters) is carried out by a centralized arbitration mechanism. The bus is re-arbitrated on a transaction basis (e.g., after single read/write transfers, or bursts of accesses of pre-defined length), based on several policies (fixed priority, round-robin, latency-driven, etc.).

If two tasks need to communicate, messages can be exchanged through communication queues [104], which can be allocated at design time either in scratch-pad memory or in remote shared memory, depending on whether tasks are mapped onto the same processor or not.

Recent MPSoCs platforms can also change the working frequency of each processor, making it possible to reduce the processor speed and, most of all, the power consumption, when the computational workload is low. These platforms are called *energy-aware MPSoCs*. The frequency of each processor core is derived from a baseline system frequency by means of integer dividers. Moreover, a synchronization module must be inserted between the bus and the processor cores to allow frequency decoupling (usually a dual-clock FIFO). The bus operates at the maximum frequency (e.g., 200 MHz). For each processor core, a set of voltage and frequency couples is specified, since the feasible operating points for these cores are not continuous but rather discrete. For modern variable voltage/variable frequency cores, this set is specified in the data-sheet.

Typically, MPSoCs always run the same set of applications, so it pays off to spend a large amount of time for finding an optimal allocation and scheduling off-line and then deploy it on the field. For this reason, many researchers in digital design automation have explored complete approaches for allocating and scheduling pre-characterized workloads on MPSoCs [124], instead of using on-line, dynamic (sub-optimal) schedulers [25, 27].

9.2 MP-ARM

In the last section we have introduced the bus, that is re-arbitrated on a transaction bases. We will see in the following chapters that modelling the bus at a so fine granularity would make the problem overly complex, so, as we will describe in Section 10.3.2, we modelled the bus as a shared resource: more than one process can use the bus at the same time, each consuming a fraction of the total bandwidth until the maximum is reached.

In general, modelling a real world scenario deciding some simplifying assumption could result in a misalignment between the expected behaviour and the real one.

To verify the effective executability of an optimal allocation and scheduling found, we need to execute it on the real platform. To simulate our solutions, we used MP-ARM [12], a MPSoC platform simulator. MP-ARM is a complete multi-processor cycle-accurate architectural simulator. Its purpose is the system-level analysis of design tradeoffs in the usage of different processors, interconnects, memory hierarchies and other devices. MP-ARM is based on SystemC as modelling and simulation environment, and includes models for several processors, interconnection busses, memory devices and support for parallel programming [1].

9.3 Problems description

In the following Chapters, we will analyze two allocation and scheduling problems on a MPSoC. Here we give a quick description of the problems:

- **Allocation and Scheduling Problem (ASP)**: We have a set of pre-characterized tasks to be executed on a MPSoC platform and a task graph representing communications (and thus precedences) among tasks. For each task, we know the worst case execution time (WCET), and the amount of memory needed to store program data, communication data and the internal state. In the platform we have a set of homogeneous processors. We know the dimension of each processor internal scratchpad memory, the dimension of the remote memory and the total bandwidth of the interconnection bus. Furthermore, we have deadline constraints on the tasks. The problem is to allocate and schedule each task on a processor and each memory requirement to a storage device such that all the constraints (precedences and resources availability) are met. The objective function is the minimization of the total amount of data transferred on the bus. We have a communication on the bus each time data are stored in the remote memory and each time two communicating tasks execute on different processors. The objective function minimizes the traffic on the bus, usually a bottleneck resource in a MPSoC.
- **Dynamic Voltage Scaling Problem (DVSP)**: We have a set of pre-characterized tasks to be executed on an energy-aware MPSoC platform and a task graph representing precedences among tasks. For each task, we know the worst case execution number of clock cycles (the WCET depends on the processor working frequency). In the platform we have a set of homogeneous processors. For each processor, we have a list of possible frequencies the processors can run at, and for each frequency we know the power consumption. We know the total bandwidth of the interconnection bus. Furthermore, we have deadline constraints both on processors and on tasks. The problem is to allocate and schedule each task on a processor deciding a working frequency for each task, such that all the constraints (precedences and resources availability) are met. The objective function is to minimize the total power consumed, because energy-aware MPSoCs are typically embedded in mobile or battery operated

devices, where the power consumption reduction is the main issue.

If the applications to be executed represent tasks to be repeated an unknown number of times, we schedule several repetitions of each task, to achieve a working rate configuration. This is the case, for example, for MPEG video stream encoding or for GSM encoding/decoding. In fact, we can not know in advance the duration of a video stream or a phone call.

In this context, our approach leverages a decomposition of the synthesis problem of on-chip multi-processor systems into two related sub-problems: (i) mapping of tasks to processors and of memory slots to storage devices and (ii) scheduling of tasks in time on their execution units. We then tackle each sub-problem with CP or IP, depending on which modelling paradigm best matches the sub-problem characteristics. The interaction is regulated by cutting planes and no-good generation and the process is proved to converge to the optimal solution. Our problem formulation will be compared with the most widely used traditional approaches, namely CP and IP modelling the entire mapping and scheduling problem as a whole, and the significant cut down on search time is showed.

In the next Section we will introduce the Decomposition Techniques we used to model our problems.

9.4 Decomposition Techniques

Decomposition techniques typically apply when facing a problem where it is possible to recognize two distinct sub-problems. The main idea is to split the problem and solve the two sub-problems separately. The two problems are called Master Problem and Sub-Problem. Typically the two sub-problems are not completely disjoint, but they share a limited number constraints w.r.t. the number of constraints involve in the original problem.

In 1961, Dantzig and Wolfe [30] proposed a decomposition algorithm where both the Master and the Sub-problem was based on Integer Linear Programming. In 1962, Benders [11] proposed the so called Benders Decomposition, extending the Dantzig-Wolfe algorithm for dealing with sub-problems of any kind. We presented the general model of the Benders Decomposition framework in Section 3.3.

Hooker and Ottosson [56] proposed, in 1995, the so called Logic-Based Benders

Decomposition (LB-BD), generalizing the classical Benders Decomposition for dealing with both master and sub problems of any kind. They applied this methodology to several planning and scheduling problems, with different objective functions, and in particular with objective functions depending on master problem variables, sub problem variables, or both.

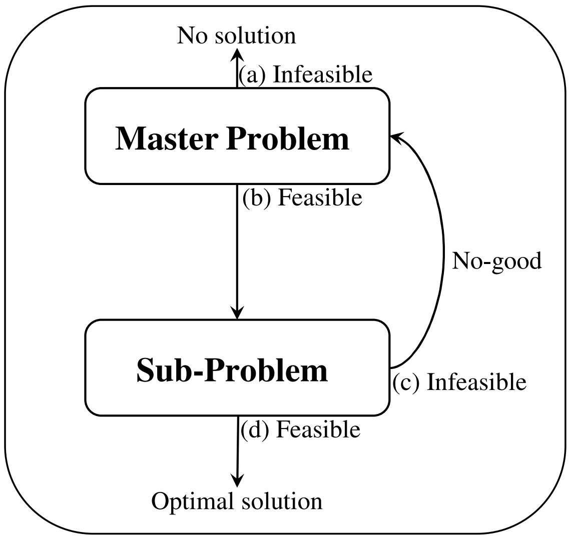


Figure 9.2: Benders Decomposition method when the objective function depends only on master problem variables

The general solving method when the objective function depends only on master problem variables is depicted in Figure 9.2. First, the master problem is solved to optimality. If it is infeasible, Figure 9.2(a), the whole problem is infeasible, being the master problem a relaxation of the original problem. If, on the contrary, we find an optimal solution for the master problem, we fix the master problem variables to the values found in the solution and we solve the subproblem, Figure 9.2(b). If it is feasible, the solution is the optimum for the original problem because it is the optimum for the master problem and the objective function depends only on master problem variables, Figure 9.2(d). If the sub-problem is infeasible a cut is generated, namely a Benders Cut, Figure 9.2(c). In this case the cut is simply a no-good: the solution, and all its symmetric, optimal for the master problem, are not feasible for the subproblem and so they must not be found again. The process iterates until the sub-problem becomes feasible.

Figure 9.3 depicts the decomposition technique when the objective function de-

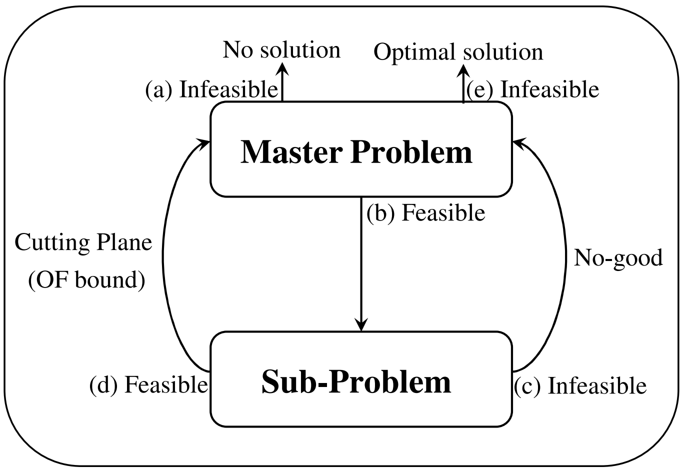


Figure 9.3: Benders Decomposition method when the objective function depends only both master and sub problem variables

pends on both master and sub-problem variables, or only on sub-problem variables. If an optimal solution for the Master is found but the subproblem is infeasible, similarly to the latter case a Benders cut is generated, Figure 9.3(a), (b) and (c). When we found an optimal sub-problem solution the search can not be stopped because the solution found is the optimum for the original problem unless a better one exists with a different master problem solution. A Benders cut is generated, Figure 9.3(d), with information on the best solution found, and thus on the Objective Function bound. The process iterates and it is proven [46] that converges to the optimal solution for the original problem when the master problem becomes infeasible, Figure 9.3(e). In this case the last solution found is the optimal one.

9.5 Related work

The synthesis of distributed system architectures has been studied extensively in the past. The mapping and scheduling problems on multi-processor systems have been traditionally modelled as integer linear programming problems. An early example is represented by the SOS system, which used mixed integer linear programming (MILP) model [105]. SOS considers processor nodes with local memory, connected through direct point-to-point channels. The algorithm does not consider real-time constraints. Partitioning under timing constraints has been addressed in [80]. A MILP model that allows to determine a mapping optimizing a trade-off function between execution time, processor and communication cost is reported in [10].

Extensions of the ILP formulation have also been used to account for memory allocation requirements, besides communication and computation ones. A hardware/software co-synthesis algorithm of distributed real-time systems that optimizes the memory hierarchy (caches) along with the rest of the architecture is reported in [89]. An integer linear programming model is used in [94] to obtain an optimal distributed shared memory architecture minimizing the global cost to access shared data in the application, and the memory cost.

The above techniques lead to static allocations and schedules that are well suited for applications whose behaviour can be accurately predicted at design time, with minimum run-time fluctuations. This is the case of signal processing and multimedia applications. Pipelining is one common workload allocation policy for increasing throughput of such applications, and this explains why research efforts have been devoted to extending mapping and scheduling techniques to pipelined task graphs. An overview of these techniques is presented in [31]. ILP formulations as well as heuristic algorithms have been traditionally employed. In [23] a retiming heuristic is used to implement pipelined scheduling, that optimizes the initiation interval, the number of pipeline stages and memory requirements of a particular design alternative. Pipelined execution of a set of periodic activities is also addressed in [40], for the case where tasks have deadlines larger than their periods. Palazzari et al. [102], focus on scheduling to sustain the throughput of a given periodic task set and to serve aperiodic requests associated with hard real-time constraints. Mapping of tasks to processors, pipelining of system specification and scheduling of each pipeline stage have been addressed in [5], aiming at satisfying throughput constraints at minimal hardware cost.

Also the voltage selection approaches have been extensively studied. They can be broadly classified into on-line and off-line techniques. In the following, we restrict ourselves to the off-line techniques since the presented approach falls into this category.

Yao et al. proposed in [126] the first DVS approach for single processor systems which can dynamically change the supply voltage over a continuous range. Ishihara and Yasuura [67] modelled the discrete voltage selection problem using an integer linear programming formulation. Xie et al. presented in [125] an algorithm for calculating the bounds on the power savings achievable through voltage selection, but is restricted to applications running on single processor systems. Jejurikar and Gupta [68] propose an algorithm that combines voltage scaling and shutdown in order to minimize dynamic and leakage energy in single processor systems.

Andrei et al. [2] proposed an approach that solves optimally the voltage scaling problem for multi-processor systems with imposed time constraints. Their solution explicitly takes into account the transition overheads implied by changing voltage levels. The continuous voltage scaling is solved using convex nonlinear programming with polynomial time complexity, while the discrete problem is proved strongly NP hard and is formulated as a MILP.

The previously mentioned approaches assume that the mapping and the schedule are given. However, the achievable energy savings of dynamic voltage scaling are greatly influenced by the mapping and the scheduling of the tasks on the target processors. Task mapping and scheduling are known NP complete problems [43] that have been previously addressed, without and with the objective of minimizing the energy. Both heuristic [117], [59] and exact solutions [13] have been proposed.

Assuming the mapping of the tasks on the processors is given as input, the authors from [51] present a scheduling technique that maximizes the available slack, which is then used to reduce the energy via voltage scaling. The allocation of the tasks on the processors (mapping) has a great influence on the energy consumption. Schmitz et al. [117] present a heuristic approach for mapping, scheduling and voltage scaling on multiprocessor architectures.

In the context of a network-on-chip platform, Hu and Marculescu [59] presented a mapping and scheduling algorithm for tasks and communications with the objective of minimizing the energy. They use a suboptimal heuristic and do not consider voltage-scalable cores.

The trend in deep-submicron CMOS technology to statically reduce the supply voltage levels and consequently the threshold voltages (in order to maintain peak performance) is resulting in the fact that a substantial portion of the overall power dissipation will be due to leakage currents [17, 72]. Martin et al. [93, 2] presented an approach for combined dynamic voltage selection and adaptive body-biasing and showed its effectiveness. At this point it is interesting to note that the approach presented in this dissertation can handle with minor changes the combined supply and body bias scaling problem. To each discrete frequency, instead of associating one supply voltage with the corresponding dynamic power, in the combined problem, we would associate to each frequency a supply and body bias voltage pair with the corresponding dynamic and leakage power. Moreover, the consideration of the body bias would not increase the computational complexity of the proposed approach.

The closest approach to the work presented in this dissertation is the one of Leung et al., [82]. They propose a MILP formulation for mapping, scheduling and voltage scaling of a given task graph to a target multiprocessor platform. They assume continuous voltages, so the overall result is suboptimal. Modelling the scheduling by means of Integer Programming, as opposed to Constraint Programming, is inefficient, resulting in an artificial explosion of the search space.

In general, even though ILP is used as a convenient modelling formalism, there is consensus on the fact that pure ILP formulations are suitable only for small problem instances (task graphs with a reduced number of nodes) because of their high computational cost. For this reason, heuristic approaches are widely used. A comparative study of well-known heuristic search techniques (genetic algorithms, simulated annealing and tabu search) is reported in [4]. Eles et al. [33] compare the use of simulated annealing and tabu search for partitioning a graph into hardware and software parts while trying to reduce communication and synchronization between parts. More scalable versions of these algorithms for large real-time systems are introduced in [74]. Many heuristic scheduling algorithms are variants and extensions of list scheduling [34].

Heuristic approaches provide no guarantees about the quality of the final solution. On the other hand, complete approaches which compute the optimum solution (possibly, with a high computational cost), can be attractive for statically scheduled systems, where the solution is computed once and applied throughout the entire lifetime of the system.

Constraint Programming is an alternative approach to Integer Programming for solving combinatorial optimization problems. The work in [78] is based on Constraint Programming to represent system synthesis problem, and leverages a set of finite domain variables and constraints imposed on these variables. Optimal solutions can be obtained for small problems, while large problems require use of heuristics. The proposed framework is able to create pipelined implementations in order to increase the design throughput. In [77] the embedded system is represented by a set of finite domain constraints defining different requirements on process timing, system resources and interprocess communication. The assignment of processes to processors and interprocess communications to buses as well as their scheduling are then defined as an optimization problem tackled by means of constraint solving techniques.

Both CP and IP techniques can claim individual successes but practical experience indicates that neither approach dominates the other in terms of computational performance. The development of a hybrid CP-IP solver that captures the best features of both would appear to offer scope for improved overall performance [96]. However, the issue of communication between different modelling paradigms arises. One method is inherited from the Operations Research and is known as Benders Decomposition [11]: it is proved to converge producing the optimal solution. Benders Decomposition (BD) technique has been extensively used to solve a large variety of problems. In [57] BD is applied to a numeric algorithm in order to solve the problem of verifying logic circuits: results show that, for some kind of circuits, the technique is an order of magnitude faster w.r.t. other state of the art algorithms. [35] embed BD in the CP environment ECLiPSe and show that it can be useful in practice. There are a number of papers using Benders Decomposition in a CP setting. [120] proposes the branch and check framework using Benders Decomposition. They applied this technique to the problem of scheduling orders on dissimilar parallel machines. Here, a set of tasks, linked by precedence constraints, must be performed on a set of parallel machine minimizing the total cost of the process. The machines are dissimilar, so the same task can be executed on a different machine with a different cost and processing time. [50] applied Benders decomposition to minimum cost planning and scheduling problems in a scenario similar to the one described in this paper, considering also release and due date constraints; in these two works, [120] and [50], the objective function involves only master problem variables, while the subproblem is simply a feasibility problem; costs depend only on the assignment of tasks to machines, differently from our problem, where contributes to the objective function depend on pairs of assignments. [21] applied BD to an allocation and scheduling problem; the master problem (allocation) is based on CP and the sub-problem (scheduling) is solved using a real-time scheduler. They considered a hardware architecture where the processors are connected via a network and communications are based on a token ring protocol: a task can communicate only when it holds the token, using all the network bandwidth and for a period of time large enough to send all the waiting messages. Tasks are scheduled with a fixed priority strategy. [55] uses Logic-Based BD for Planning and Scheduling problems. The paper explores two different planning and scheduling problems with different objective functions. In the first problem the main objective is to minimize the cost, that can be computed directly in terms of master problem variables, since the cost depends only on the allocation. The sub-problem becomes a feasibility problem and the cuts generated forbid the master problem to assign the same set of tasks to the same resource. In the second problem the objective is to minimize the makespan, thus the objective function depends also on the sub-problem variables, that becomes an optimization problem itself. The sub-problem provides lower bounds on the makespan of each processor and a Benders cut for the total makespan is derived and posted in the master problem. They do not consider tasks with precedence constraints, but with release and due date; communication between tasks are not addressed in the work.

Although a lot of work has been done applying BD to allocation and scheduling problems, we believe that our approach is not directly comparable with them, mainly because we take in consideration a real application where data must be exchanged between tasks and each task must read/write data (and thus must use the bus resource) during its execution.

9.6 Overview of the Part III

Part III is organized as follows: in Chapter 10 we describe the ASP and its model based on decomposition, discussing about design choices and simplifying assumptions. In Chapter 11 we will show experimental results to give evidence that our tool is efficient to solve the ASP and we will validate the executability of our solutions by simulating them on a virtual MPSoC platform, comparing the results. Chapters 12 and 13 are devoted to the DVSP analysis and have the same structure of Chapters 10 and 11.

Chapter 10

ASP model

Introduction

In this Chapter we will analyze the ASP, describing the model we used to solve it, as well as the modelling assumption we have done. In Section 10.1 we will formalize our problem, defining the problem decomposition in Section 10.2. In Section 10.3 we will present the complete model for the ASP.

10.1 Allocation and Scheduling Problem description

We consider the MPSoC platform introduced in Section 9.1. The target application to be executed on top of the hardware platform is input to our methodology, and for this purpose is represented as a task graph. This latter consists of a graph pointing out the parallel structure of the program. The application workload is therefore partitioned into computation sub-units denoted as tasks, which are the nodes of the graph. Graph edges connecting any two nodes indicate task communication dependencies, in the sense that the output data of a task are the input data for the subsequent task. Each task is annotated with computation, storage and communication requirements.

In detail, the worst case execution time (WCET) is specified for each task and plays a critical role whenever application real time constraints (expressed here in terms of minimum required throughput) are to be met. Each task also has 3 kinds of associated memory requirements:

- **Program Data**: storage locations are required for computation data and for processor instructions. They can be allocated either on the local scratchpad memory or on the remote on-chip memory.

- **Internal State**: when needed, an internal state of the task can be stored either locally or remotely.

- **Communication queues**: the task needs queues to transmit and receive messages to/from other tasks, eventually mapped on different processors. In the class of MPSoCs we are considering, such queues should be allocated only on local memories, in order to implement an efficient inter-processor communication mechanism.

Communication requirements of each task are automatically determined once computation data and internal state are physically allocated to scratchpad or remote memory, and obviously depend on the size of such data.

We have a real time constraint on the application throughput: each task, and in particular the last one, must generate the output data at most every time period RT.

The goal is to allocate and scheduling tasks to resources and memory requirements to storage device such that all the precedence, real time and resource capacity constraints are satisfied, minimizing the total amount of data transferred on the bus. We have a communication on the bus when:

- Program data are stored on the remote memory. These data are accessed during the task execution;

- Internal state is stored on the remote memory. Internal state is read immediately before the task execution and written immediately after the task execution;

- Two communicating tasks are allocated to different processors. If task A must communicate with task B, A writes the data in a buffer queue internal to its processor during the execution, and B reads these data using the bus before starting the execution.

The objective function minimizes the bus congestion. In the real platforms, the bus is usually a bottleneck resource and a congestion on the bus leads to an higher probability of collisions and an higher bus arbitrage mechanism time overhead. This causes an increase in the application total execution time and, in the worst case, the real time requirement could be violated.

We applied our methodology to task graphs extracted from a real video graphics application processing pixels of a digital image. Many real-life signal processing applications are subject to tight throughput constraints, therefore leverage a pipelined workload allocation policy. As a consequence, the input graph to our methodology consists of a pipeline of processing tasks, and can be easily extended to all pipelined applications. These applications process an unknown number of video frames and each task in the pipelined task graph must be executed once for each frame. We can not know in advance the number of repetitions of the pipeline, so we need to schedule an adequate number of repetitions of each task to analyze the system behaviour at full rate.

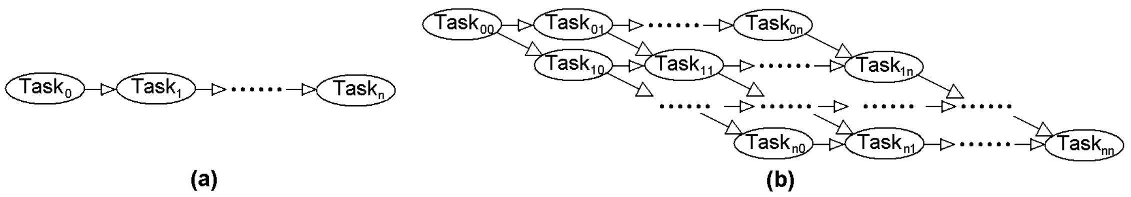


Figure 10.1: Pipelined task graph and pipeline repetition

Figure 10.1(a) depicts a pipeline of tasks, where each edge represents a precedence constraint due to communication. Figure 10.1(b) depicts several repetitions of the pipeline: here the horizontal edges represent communication constraints, while the diagonal ones represent precedence constraints due to the fact that repetitions must be executed in order; in other words, the $i - th$ repetition of each task must be executed before the $(i + 1) - th$ repetition. Analyzing the Figure 10.1(b) we can argue that, if n is the number of tasks to be scheduled, after the $n - th$ repetition the system is at full rate. Therefore, we need to schedule only n repetitions of the pipeline.

10.2 Motivation for problem decomposition

The problem described in the previous section has a very interesting structure. As a whole, it is a scheduling problem with alternative resources. In fact, each task should be allocated to one of the processors. In addition, each memory slot required for processing the task should be allocated to a memory device. Clearly, tasks should be scheduled in time subject to real time constraints, precedence constraints, and capacity constraints on all unary and cumulative resources. However, on a different perspective, the problem decomposes into two problems:

- the allocation of tasks to processors and the memory slots required by each task to the proper memory device;
- a scheduling problem with static resource allocation.

The objective function of the overall problem is the minimization of communication cost. This function involves only variables of the first problem. In particular, we have a communication cost each time two communicating tasks are allocated on different processors, and each time a memory slot is allocated on a remote memory device. Once we have optimally allocated tasks to resources, if the allocation is feasible for the scheduling problem, an optimal solution for the problem overall is found.

We used a Logic-Based Benders Decomposition approach, introduced in Section 9.4 to solve the problem. We solve the allocation master problem using IP and the scheduling sub-problem using CP. The mechanism the two solvers use to interact is the one described in Figure 9.2, depicting the case when the objective function depends only on master problem variables.

Now let us note the following: the assignment problem allocates tasks to processors, and memory requirements to storage devices. However, since real time constraints are not taken into account by the allocation module, the solution obtained tends to pack all tasks in the minimal number of processors. In other words, the only constraint that prevents to allocate all tasks to a single processors is the limited capacity of the tightly coupled memory devices. However, these trivial assignments do not consider throughput constraints which make them most probably infeasible for the overall problem. To avoid the generation of these (trivially infeasible) assignments, we should add to the master problem model a relaxation of the subproblem. In particular, we should state in the master problem that the sum of the durations of all the tasks allocated to a single processor does not exceed the real time requirement. In this case, the allocation is far more similar to the optimal one for the problem at hand. The use of a relaxation in the master problem is widely used in practice and helps in producing better solutions.

10.3 Modelling the Allocation and Scheduling Problem

As described in the last Section, the problem we are facing can be split into the resource allocation master problem and the scheduling sub-problem.

10.3.1 Allocation problem model

We start from the pipelined task graph presented in Figure 10.1(a). Each task $Task_i$ should be allocated to a processor. In addition each task needs a given amount of memory to store data, mem_i to store the program data, $state_i$ to store the internal state and $data_i$ to store the communication queues. Data can be allocated either in the local memory of the processor running the task (of dimension MEM_j) or in the remote one except for communication queues that are always mapped locally.

The allocation problem is the problem of allocating n tasks to m processors, such that the total amount of memory allocated to the tasks, for each processor, does not exceed the maximum available.

We assume the remote on-chip memory to be of unlimited size since it is able to meet the memory requirement of the application we are facing. The problem objective function is the minimization of the amount of data transferred on the bus.

We model the problem as an IP model. In the IP model we consider four decision variables:

- T_{ij}, taking value 1 if task i executes on processor j, 0 otherwise,
- Y_{ij}, taking value 1 if task i allocates the program data on the scratchpad memory of processor j, 0 otherwise,
- Z_{ij}, taking value 1 if task i allocates the internal state on the scratchpad memory of processor j, 0 otherwise,
- X_{ij}, taking value 1 if task i executes on processor j and task $i+1$ does not, 0 otherwise.

The constraints we introduced in the model are:

$$\sum_{j=1}^{m} T_{ij} = 1, \forall i \in 1 \dots n \tag{10.1}$$

$$X_{ij} = |(T_{ij} - T_{i+1j})|, \forall i \in 1 \dots n, \forall j \in 1 \dots m \tag{10.2}$$

$$\sum_{i=1}^{n} (Y_{ij} \times mem_i + Z_{ij} \times state_i + T_{ij} \times data_i) \leq MEM_j, \forall j \tag{10.3}$$

$$T_{ij} = 0 \Rightarrow Y_{ij} = 0, Z_{ij} = 0 \tag{10.4}$$

Constraints (10.1) state that each process can execute only on a processor, while constraints (10.2) state that X_{ij} must be equal to 1 iff $T_{ij} \neq T_{i+1j}$, that is, iff task i and task $i+1$ execute on different processors. Constraints (10.2) are not linear, thus we cannot use them in a IP model. If we consider that the sum $Xij + T_{ij} + T_{i+1j}$ must always equal either to 0 or 2, constraints (10.2) can be rewritten as:

$$T_{ij} + T_{i+1j} + X_{ij} - 2K_{ij} = 0 \ , \forall i \ , \forall j \tag{10.5}$$

where K_{ij} are integer binary variables that enforce the sum $T_{ij} + T_{i+1j} + X_{ij}$ to be equal either to 0 or 2.

Constraints (10.3) state that the total amount of tasks memory requirements allocated to each internal memory must not exceed the maximum capacity. Constraints (10.4) state that if a processor j is not assigned to a task i neither its program data nor the internal state can be stored in the local memory of processor j.

As explained in section 10.2, in order to prevent the master problem solver to produce trivially infeasible solutions, we need to add to the master problem model a relaxation of the subproblem. For this purpose, for each set of consecutive tasks whose execution times sum exceeds the real time requirement (RT), we impose constraints preventing the solver to allocate all the tasks in the group to the same processor. To generate this constraints, we find out all groups of consecutive tasks whose execution times sum exceeds RT. Constraints are the following:

$$\sum_{i \in S} Dur_i > RT \Rightarrow \sum_{i \in S} T_{ij} \leq |S| - 1 \ , \ \forall j \tag{10.6}$$

The objective function is the minimization of the total amount of data transferred on the bus for each pipeline. This amount consists of three contributions: when a task allocates its program data in the remote memory, it reads these data throughout the execution time; when a task allocates the internal state in the remote memory, it reads these data at the beginning of its execution and updates them at the end; if two consecutive tasks execute on different processors, their communication messages must be transferred through the bus from the communication queue of one processor to the other. Using the decision variables described above, we have a contribution respectively when: $T_{ij} = 1, Y_{ij} = 0$; $T_{ij} = 1, Z_{ij} = 0$; $X_{ij} = 1$. Therefore, the objective function is:

$$min \sum_{j=1}^{m} \sum_{i=1}^{n} (mem_i(T_{ij} - Y_{ij}) + state_i(T_{ij} - Z_{ij}) + (data_i X_{ij})/2) \tag{10.7}$$

The third contribution to the objective function is divided by 2 because the same communication is considered twice: in fact, from constraints (10.5) it follows that, if exactly one task among $Task_i$ and $Task_{i+1}$ executes on processor j, we have $X_{ij} = 1$ and also $X_{i+1j} = 1$.

10.3.2 Scheduling problem model

Once tasks and memory requirements have been allocated to the resources, we need to schedule the tasks execution. Since we are considering a pipeline of tasks, we need to analyze the system behavior at working rate, that is when all the tasks are running or ready to run. To do that, we need to consider several instantiations of the same task; as explained in Section 10.1, to achieve a working rate configuration, the number of repetitions of each task must be at least equal to the number of tasks n. So, to solve the scheduling problem, we must consider at least n^2 tasks (n iterations for each task), see Figure 10.1(b).

In the scheduling problem model, we split each task into the several activities it is composed of: for each task $Task_{ij}$ we introduce a variable A_{ij}, ($i = [0 \dots n-1]$, $j = [0 \dots n-1]$), representing the computation activity of the task. A_{ij} is the $j-th$ iteration of the computational activity of $Task_i$. Once the allocation problem is solved, we statically know if a task needs to use the bus to communicate with another task, or to read/write computation data and internal state in the remote memory. In particular, each activity A_{ij} must read the communication queue from the activity A_{i-1j}, or from the pipeline input if $i = 0$. To schedule these phases, we introduce in the model the activities In_{ij}. If a task requires an internal state, the state must be read just before the execution and written just after: we therefore introduce in the model the activities RS_{ij} and WS_{ij} for each $Task_i$ requiring an internal state.

The duration of the activities described so far depends on whether the data are stored in the local or the remote memory (data transfer through the bus needs more time than the transfer of the same amount of data using the local memory) but, after the allocation, these durations can be statically calculated. For each activity, we have a variable $Start$ representing the starting time, and a value Dur representing the duration.

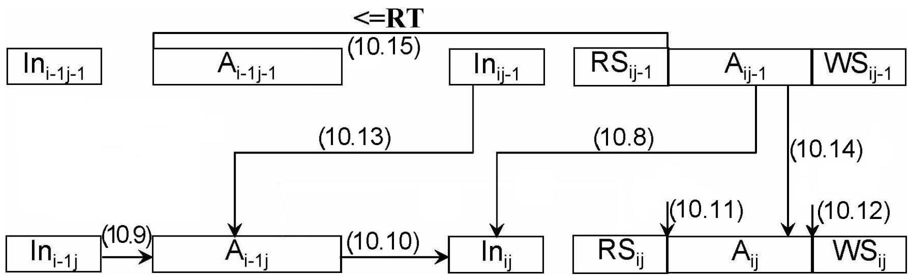


Figure 10.2: Precedence constraints among the activities

Figure 10.2 depicts the precedence constraints among the activities. Each task $Task_{ij}$ is represented by the activity A_{ij}, preceded by the input data reading activity In_{ij} and, possibly, preceded by the internal state reading activity RS_{ij} and followed by the internal state writing activity WS_{ij}.

The precedence constraints among the activities introduced in the model are the following (labels are used in Figure 10.2):

$$A_{i,j-1} \prec In_{ij} \ , \ \forall \ i,j \tag{10.8}$$

$$In_{ij} \prec A_{ij} \ , \ \forall \ i,j \tag{10.9}$$

$$A_{i-1,j} \prec In_{ij} \ , \ \forall \ i,j \tag{10.10}$$

$$RS_{ij} \preceq A_{ij} \ , \ \forall \ i,j \tag{10.11}$$

$$A_{ij} \preceq WS_{ij} \ , \ \forall \ i,j \tag{10.12}$$

$$In_{i+1,j-1} \prec A_{ij} \ , \ \forall \ i,j \tag{10.13}$$

$$A_{i,j-1} \prec A_{ij} \ , \ \forall \ i,j \tag{10.14}$$

$$Start\_A_{ij} - Start\_A_{i,j-1} \leq RT \ , \ \forall \ i,j \tag{10.15}$$

where the symbol $\prec$ means that the activity on the right should precede the activity on the left, and the symbol $\preceq$ means that the activity on the right must start as soon as the execution of the activity on the left ends: e.g., $In_{ij} \prec A_{ij}$ means $Start\_In_{ij} + Dur\_In_{ij} \leq Start\_A_{ij}$, and $RS_{ij} \preceq A_{ij}$ means $Start\_RS_{ij} + Dur\_RS_{ij} = Start\_A_{ij}$.

Constraints (10.8) state that each task iteration can start reading the communication queue only after the end of its previous iteration. Constraints (10.9) state that each task can start only when it has read the communication queue, while constraints (10.10) state that each task can read the data in the communication queue only when the previous task has generated them. Constraints (10.11) and (10.12) state that each task must read the internal state just before the execution and write it just after. Constraints (10.13) state that each task can execute only if the previous iteration of the following task has read the input data; in other words, it can start only when the memory allocated to the process for storing the communication queue has been freed. Constraints (10.14) state that tasks iterations must execute in order. Furthermore, we introduced the real time requirement constraints (10.15), whose relaxation is used in the allocation problem model. Each task must execute at most each time period RT.

10.3.3 Modelling the BUS

Each processor is modelled as a unary resource, that is a resource with capacity one. As far as the bus is concerned, we made a simplification. A real bus is a unary resource: if we model a bus as a unary resource, we should describe the problem at a finer grain with respect to the one we use, i.e., we have to model task execution using the clock cycle as unit of time. The resulting scheduling model would contain a huge number of variables. We therefore consider the bus as an additive resource, in the sense that more activities can share the bus using only a fraction of the total bandwidth available. We experimentally found that a good value for the fraction of bus granted to each activity is $1/m$ of the total bandwidth, where m is the number of processors.

Figure 10.3 depicts this assumption. The leftmost figure represents the bus allocation in a real processor, where the bus is assigned to different tasks at different times. Each task, when owning the bus, uses its total bandwidth (400 MByte/sec in the platform we consider). The rightmost figure, instead, represents how we model the bus. The bus arbitration mechanism will then transform the bus allocation into

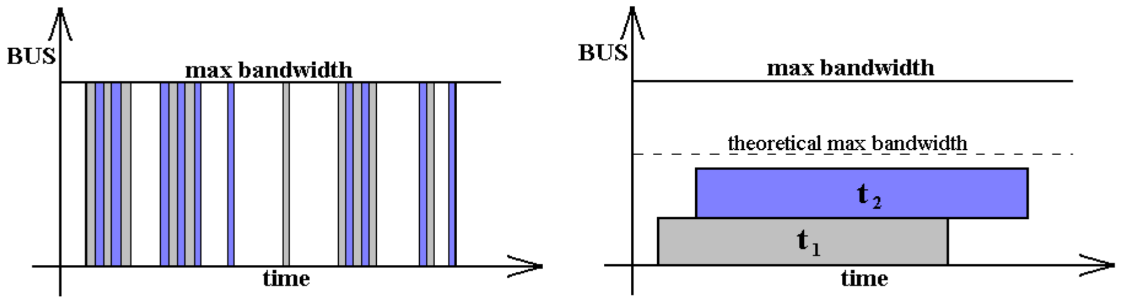


Figure 10.3: Bus allocation in a real processor (left) and in our model (right)

the interleaving of fine granularity bus transactions on the real platform. We experimentally found that this additive model is valid if the bus workload is under the 60% of its total bandwidth (240 MByte/sec). We therefore modelled the bus as an additive resource with a capacity equal to the 60% of the total. In Figure 10.3 this is depicted by the dotted line labelled *theoretical max bandwidth*. This value will be motivated in Section 11.3.1.

To define the communication requirements of each activity (the amount of computation data stored in the remote memory) we consider the amount of data they have to communicate and we spread it over its WCET. In this way we consume only a fraction of the overall bus bandwidth for the duration of the activity. In particular, the activities IN_{ij}, RS_{ij} and WS_{ij} use the whole fraction of the bus bandwidth they own and the execution time thus depends on the amount of data they have to transfer, while the activities A_{ij} spread the bus usage over all the execution. The latter activities thus consume only a little slice of the bus bandwidth. Figure 10.4 depicts these assumptions: the height of state reading/writing activities is the maximum fraction f available for an activity, thus their duration is $state_i/f$. On the contrary, we know the duration of the computational activities Dur_i, and the height of the bus requirement is $data_i/Dur_i$.

10.3.4 Generation of Logic-based Benders cut

When an allocation is provided, the minimal makespan schedule is computed if it exists. On the contrary, if no feasible schedule exists, we have to generate a Logic-Based Benders Cut, that in this case is a no-good, and pass it to the allocation module. Since the allocation module is a Integer Programming solver, the no-good should have the form of a linear constraint.

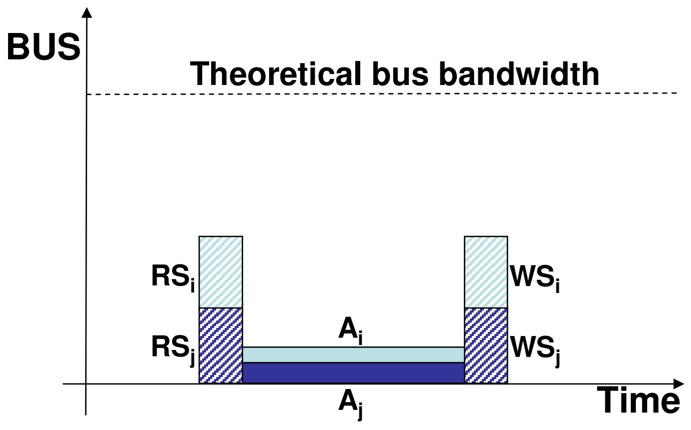


Figure 10.4: Activities bus usage

We investigated two kind of no-goods:

- The first kind of no-good should prevent from finding again an infeasible allocation. The resulting no-good is:

$$\sum_{j=1}^{m} \sum_{i \in S_j} X_{ij} < n \tag{10.16}$$

  where S_j is the set of tasks allocated to processor j. We also introduce no-goods to cut symmetric allocations.

- The cuts described above remove only complete solutions. It is possible to refine the analysis and to find tighter cuts that remove only the allocation of tasks to bottleneck resources. In particular, we select all the resources that provoke a failure, i.e. resources that lead to a violation of real time constraints. We call them *conflicting resources*, CR. Then, we impose that for each resource in $R \in CR$ the set of tasks ST_R allocated to R should not be reassigned to the same resource in the next iteration. The resulting no-goods are:

$$\sum_{i \in ST_R} T_{iR} \leq |ST_R| - 1 \ , \ \forall \ R \in CR \tag{10.17}$$

  This constraints prevent the same allocation to appear again on each conflicting resource. Identifying the conflicting resources requires to solve a one machine scheduling for each processor p considering constraints involving only tasks running on p. Finding these cuts is therefore a NP-Hard problem; in Section 11.2 we will experimentally show when it pays off.

10.4 Simplifying assumptions on the activities duration

In the last Section we have described the simplifying assumption we have done when modelling the bus. This is not the only simplification we did: in fact, in order to be able to generate a schedule off-line, we must estimate the activity durations. Each execution run of the same activity usually has a different duration, due to bus congestion or processor overhead to perform internal processes. We need to find a mean value for each activity. Here we consider different ways to deduce these values depending on the kind of the activity. Names introduced in subsection 10.3.2 are used. The following notation is used in the formulae for computing activities duration: B is the amount of data to be read/written; t_r, t_w, t_{rl}, t_{wl} are respectively the time for reading, writing, reading locally and writing locally one data; CM is the cache miss percentage and n is the available fraction of the bus, in the sense that each task owns $1/n$ of the total bandwidth.

- **Durations of activities (A_{ij})**: we characterize the tasks duration depending on where program data are stored. If they are in the local memory, we compute the task duration as the mean over 100 simulation runs of the execution time of the task alone on a processor with all its data stored in the local scratchpad and this value is Dur_i. If, instead, the task has the program data allocated remotely, it must access the bus several times to read these data, so its execution time must be increased by the time spent accessing remote data. Access efficiency to remote program data is typically enhanced by means of local caches, therefore the task has to actually access the bus only when a cache miss occurs. Using the notation introduced, in case program data are stored remotely, the total execution time is $Dur_i + B \times t_r \times CM \times n$.
- **Duration of communication queue reads (In_{ij})**: the duration of the reading activities depends on whether 2 communicating tasks are running on the same processor or not. In the former case, no bus transactions are needed since exchange data are produced and consumed directly to/from scratchpad memory, and activity duration is equal to $B \times t_{rl}$, while in the latter case the value

$B \times t_r \times n$ accounts for data transfers through the bus.

- **Internal state reads/writes** (RS_{ij}/WS_{ij}): the internal state can be stored either in the local scratchpad or remotely. Though, remote internal state data are efficiently accessed via cache memories. Depending on where data are stored, formulae for the durations of internal state reading are respectively $B \times t_{rl}$ or $B \times t_r \times CM \times n$, while formulae for the internal state writing are $B \times t_{wl}$ or $B \times t_w \times CM \times n$.

The problem that arises when considering the assumptions we did (activities duration and additive bus model) is that the durations might be inaccurate and the model might not exactly describe the considered problem. In the extreme case, the solutions found could even be not executable in the real platform, so we must check a posteriori if a schedule is feasible, executable and evaluate the mismatch between the real system behaviour and the theoretical results.

Chapter 11

ASP Experimental Results

Introduction

In this Chapter we will show the experimental results obtained when solving the ASP. In Section 11.1 we will solve the problem using the hybrid solver based on both the IP and CP models described in the last Chapter implementing the Logic-Based Benders Decomposition (LB-BD) methodology. We will show the experimental results and we will compare them with those obtained when solving the problem using a single technology, either IP or CP. In section 11.2 we will discuss about using different no-goods to make the two solvers interacting. Section 11.3 is devoted at measuring the accuracy of the solutions found and their executability on the real platform.

11.1 Computational Efficiency

To validate the strength of our approach, we now compare the results obtained using the hybrid model described in the last Chapter (**Hybrid** in the following) with results obtained using only a CP or IP model to solve the overall problem. Actually, since the first experiments showed that both CP and IP were not able to find a solution, except for the easiest instances, within 15 minutes, we simplified these models removing some variables and constraints. In CP, we fixed the activities execution time not considering the execution time variability due to remote memory accesses, therefore we do not consider the In_{ij}, RS_{ij} and WS_{ij} activities, including them statically in the activities A_{ij}. In IP, we do not consider all the variables and constraints involving the bus: we do not model the bus resource and we therefore suppose that each activity can access data whenever it is necessary.

In the Hybrid model we introduced the first kind of no-goods described in Section 10.3.4, those removing complete allocations. In Section 11.2 we will discuss about using different cuts.

We generate a large variety of problems, varying the number of tasks from 4 to 10 and the number of processors from 1 to 9. We considered only task graphs representing a pipeline. All the results presented are the mean over a set of 10 problems having the same number of tasks and processors. All problems considered have a solution. Experiments were performed on a 2GHz Pentium 4 with 512 Mb RAM. We used ILOG CPLEX 8.1 [63], ILOG Solver 5.3 [65] and ILOG Scheduler 5.3 [64] as solving tools.

In figures 11.1 and 11.2 we compare the algorithms search time for problems with a different number of, respectively, tasks and processors. Times are expressed in seconds and the y-axis has a logarithmic scale.

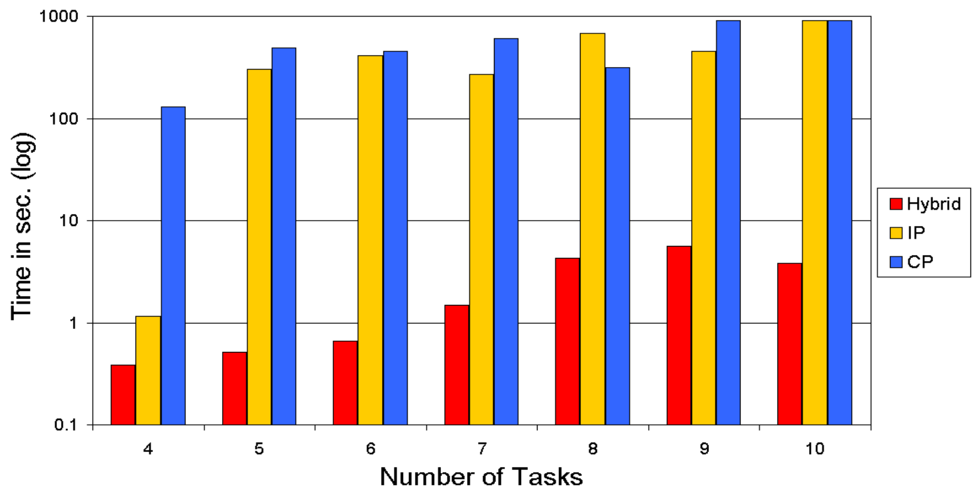


Figure 11.1: Comparison between algorithms search times for different task number

Although CP and IP deal with a simpler problem model, we can see that these algorithms are not comparable with Hybrid, except when the number of tasks and processors is very low; this is due to the fact that the problem instance is very easy to be solved, and Hybrid loses time creating and solving two models, the allocation and the scheduling. As soon as the number of tasks and/or processors grows, IP and CP performances worsen and their search times become order of magnitude higher

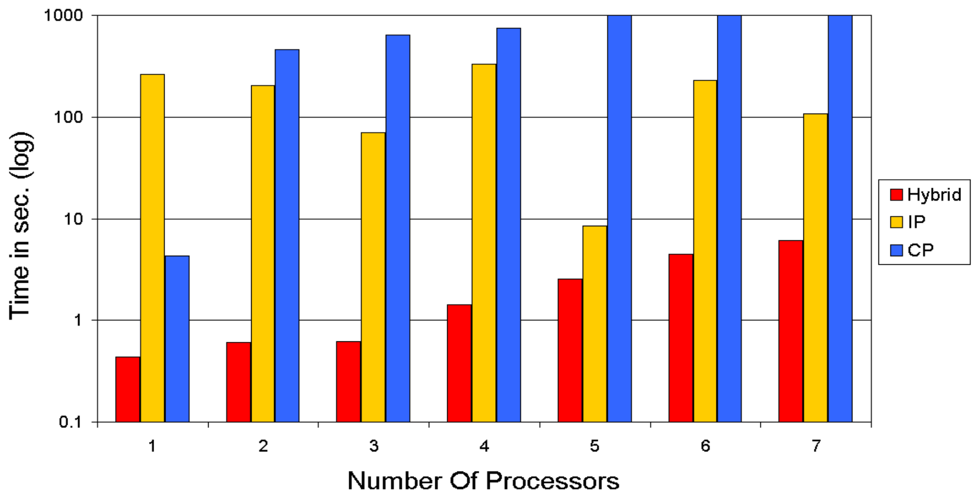


Figure 11.2: Comparison between algorithms search times for different processor number

w.r.t. Hybrid. Furthermore, we considered in the figures only instances where the algorithms are able to find the optimal solution within 15 minutes, and, for problems with 6 tasks or 3 processors and more, IP and CP can find the solution only in the 50% or less of the cases, while Hybrid always finds the optimal solution.

Alloc	Sched	Procs	Time(s)	Iters
4	32/64	3	0,42	1,01
4	32/64	4	0,41	1,05
5	50/100	4	0,5	1,01
5	50/100	5	0,57	1,07
6	72/144	4	0,6	1,06
6	72/144	5	0,85	1,09
6	72/144	6	1,26	1,10
7	98/196	5	2,84	1,08
7	98/196	6	6,14	1,09
8	128/256	5	0,98	1,03
8	128/256	6	9,53	1,07
8	128/256	7	14,37	1,12
9	162/324	6	7,71	1,11
9	162/324	7	9,25	1,02
10	200/400	4	3,85	1,03
10	200/400	7	27,85	1,06
10	200/400	9	46,69	1,11

Table 11.1: Search time and number of iterations for ASP instances

From now on we will solve the ASP instances only with the Hybrid solver. Table 11.1 shows the search time (in seconds) and the mean number of iterations between the master and the sub-problem when solving different ASP instances, with the number of allocated tasks and processors shown respectively in the first (*Alloc*) and third (*Procs*) column. We recall that, since each task is decomposed, for the scheduling subproblem, into two or four activities (data reading, execution and, if it is the case, state reading and writing) and we schedule n repetitions of each activity, where n is the number of tasks in the pipeline, the number of scheduled activities can vary from $2 \times n^2$ to $4 \times n^2$. Column *Sched* shows these two possible values. Each line represents the mean over 10 instances with the same number of tasks and processors.

We can see that the optimal solution can always be found within one minute and the mean number of iteration is very close to 1: this means that, in the most of the cases, the optimal solution can be found without iterations. In other words, the first optimal allocation found is also schedulable. This happens thanks to the relaxation of the sub-problem introduced in the master problem to take into account task durations. The only case to have an infeasibility in the sub-problem is when we find an allocation so close to the real time requirement that even a communication on the bus, usually shorter w.r.t. the execution, will cause a violation of the real time constraint. In Chapter 13 we will analyze in deep the importance of adding a relaxation in the master problem.

11.2 Effectiveness of the Benders Cuts

In Section 10.3.4 we have described two kinds of Benders Cut. The first kind (referred to as Base cut), very easy to be calculated and added to the model, has been used in all the experiments presented so far. To show the effectiveness of the second kind of cuts (referred to as Advanced cut), found solving a NP-Hard problem for each processor, we selected a hard ASP instance with 10 tasks allocated and 298 activities scheduled, and we solved it with different deadline values, starting from a very weak one to the tightest one. The deadline constraint values and the tasks characteristics have been selected to force an high number of iterations between the master and the sub-problem.

Table 11.2 shows the search time (in seconds) and the number of iterations when

	Number of Iterations		**Search time (sec.)**	
Deadline	**Base**	**Advanced**	**Base**	**Advanced**
1000000	3	3	1,23	0,609
647824	1	1	0,771	0,765
602457	1	1	0,562	0,592
487524	18	6	6,045	1,186
459334	185	16	198,452	9,546
405725	192	23	325,142	9,954
357491	79	17	60,747	6,144
345882	6	4	5,375	1,657
340218	4	3	3,347	1,046
315840	5	3	3,896	1,703
307465	2	2	2,153	0,188

Table 11.2: Number of iterations varying the deadline and with different Benders Cuts

solving these instances respectively without (row Base) and with (row Advanced) the second kind of cuts described in 10.3.4 for descending deadline values (row Deadline). We can see that, when the number of iterations is high, the Advanced cuts reduce them notably. These cuts are extremely tight, but we experimentally see that the time to generate them is one order of magnitude greater w.r.t. the time to generate the Base cuts. The mean time to find a Base cut is 7.4ms, while finding an Advanced cut needs 50ms on average, so finding the latter pays off only on hard instances where the two solvers iterates an high number of times.

11.3 Validation of the results

In Section 11.1 we have given evidence that our tool is efficient and scalable for solving to optimality the ASP problem. As introduced in Section 10.4, we have performed some simplifying assumption at design time. First of all, we modelled the bus as an additive resource, and secondly we statically calculated a fixed time for the activities execution time. We therefore need to validate our choices and to simulate our optimal solutions on the real platform, comparing the two executions.

We have performed four kinds of experiments, namely (i) validation and calibration of the bus additive model, (ii) measurement of deviations of simulated throughput from theoretically derived one for a large number of problem instances,

(iii) verification of the solutions executability on the real platform and (iv) showing the viability of the proposed approach by means of two real applications, namely the GSM codec and the MIMO processing.

11.3.1 Validation of the bus additive model

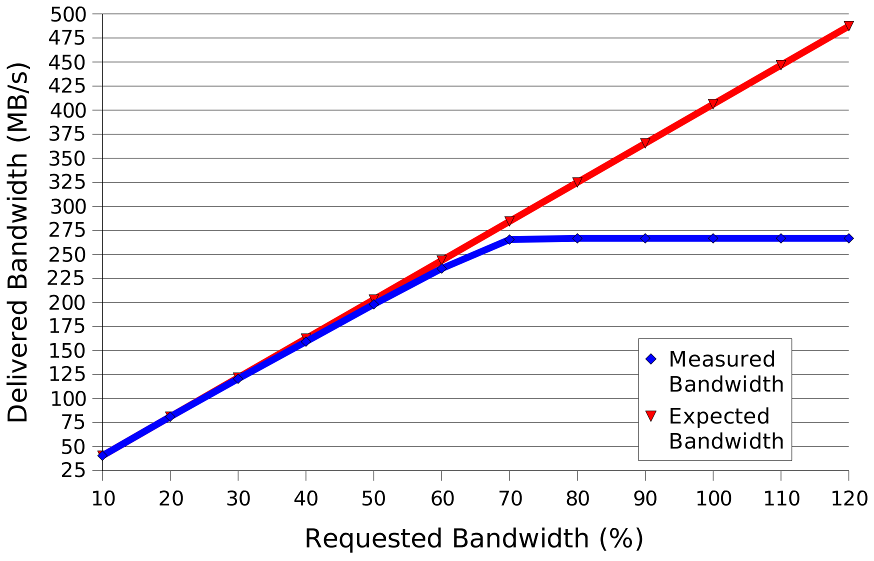


Figure 11.3: Implications of the bus additive model

The intuitive meaning of the bus additive model is illustrated by the experiment of Figure 11.3. An increasing number of uniform traffic generators, consuming each 10% of the maximum theoretical bandwidth (400 MByte/sec), have been connected to the bus, and the resulting real bandwidth provided by the bus measured in the virtual platform. It can be clearly observed that the delivered bandwidth keeps up with the requested one until the sum of the requirements amounts to 60% of the maximum theoretical bandwidth. This defines the real maximum bandwidth, notified to the optimizer, under which the bus works in a predictable way. If the communication requirements exceed the threshold, as a side effect we observe an increase of the execution times of running tasks with respect to those measured without bus contention, as depicted in Figure 11.4. For this experiment, synthetic tasks running on each processor have been employed. The 60% bandwidth threshold value corresponds to an execution time variation of about 2% due to longer bus transactions.

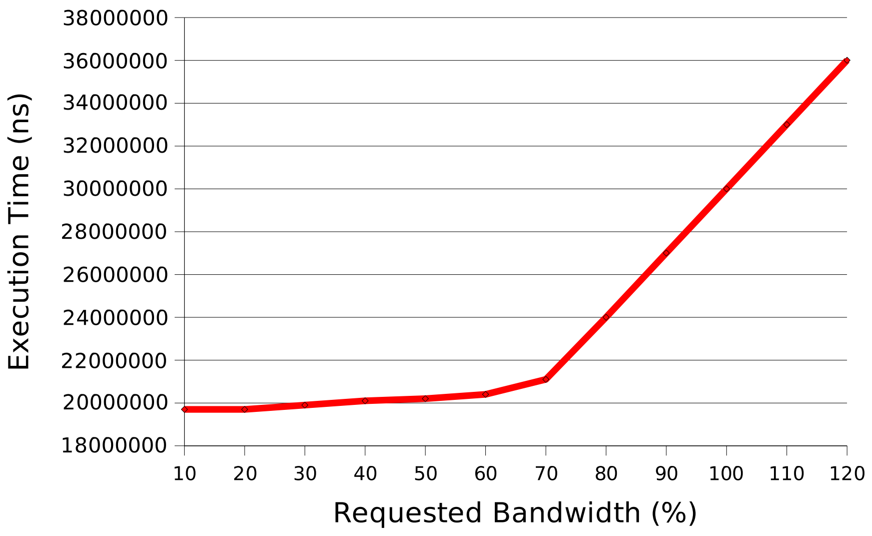


Figure 11.4: Execution time variation

However, the threshold value also depends on the ratio of bandwidth requirements of the tasks concurrently trying to access the bus. Contrarily to Figure 11.3, where each processor consumes the same fraction of bus bandwidth, Figure 11.5 shows the deviations of offered versus required bandwidth for competing tasks with different bus bandwidth requirements. Configurations with different number of processors are explored, and numbers on the x-axys show the percentage of maximum theoretical bandwidth required by each task. It can be observed that the most significant deviations arise when one task starts draining most of the bandwidth, thus creating a strong interference with all other access patterns. The presence of such communication hotspots suggests that the maximum cumulative bandwidth requirement which still stimulates an additive behaviour of the bus is lower than the one computed before, and amounts to about 50% of the theoretical maximum bandwidth.

The latter results do not discredit our assumption to set the theoretical maximum bandwidth to the 60% of the real value because each task owns only $1/m$ of the band, where m is the number of processors. Whit this assumption, in Figure 11.5 the configurations with the higher error are not possible. Furthermore, given that the real applications executed on the MPSoCs are typically CPU intensive, the BUS is seldom congested.

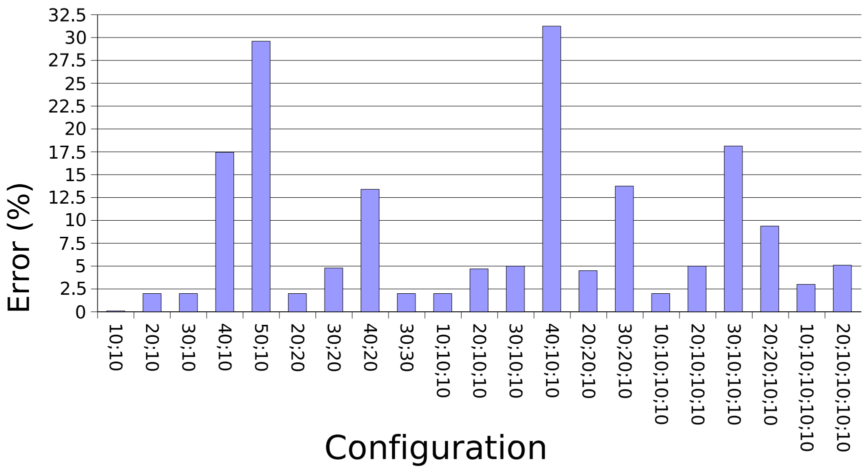


Figure 11.5: Bus additive model for different ratios of bandwidth requirements among competing tasks for bus access

11.3.2 Measuring accuracy on activity duration

To validate the accuracy of the pre-characterization and its impact on the computed schedule we compared the activity durations proposed by the scheduler and the simulator. To simulate the activity duration we used parameters from the simulation and we compute the average duration on 100 runs.

Activity	**Accuracy**
Processing	99.5%
Data read	99.5%
State read/write	96%
Throughput	95%

Table 11.3: Activity duration accuracy

Table 11.3 shows the percentage of accuracy (ratio of the durations) for each kind of activity and for the throughput. As we can see, activities accuracy is very high and this leads to an high throughput accuracy, that is the most important parameter to be taken into consideration, since we are working in scenario with RT constraints. Clearly, if the accuracy were low, the should be a feed back of the pre-characterization phase, in order to compute more realistic activity durations.

11.3.3 Validation of allocation and scheduling solutions

We have deployed the virtual platform to implement the allocations and schedules generated by the optimizer, and we have measured deviations of the simulated throughput from the predicted one for 50 problem instances. A synthetic benchmark has been used for this experiment, allowing to change system and application parameters (local memory size, execution times, data size, etc.). We want to make sure that modelling approximations are not such to significantly impact the accuracy of optimizer results with respect to real-life systems.

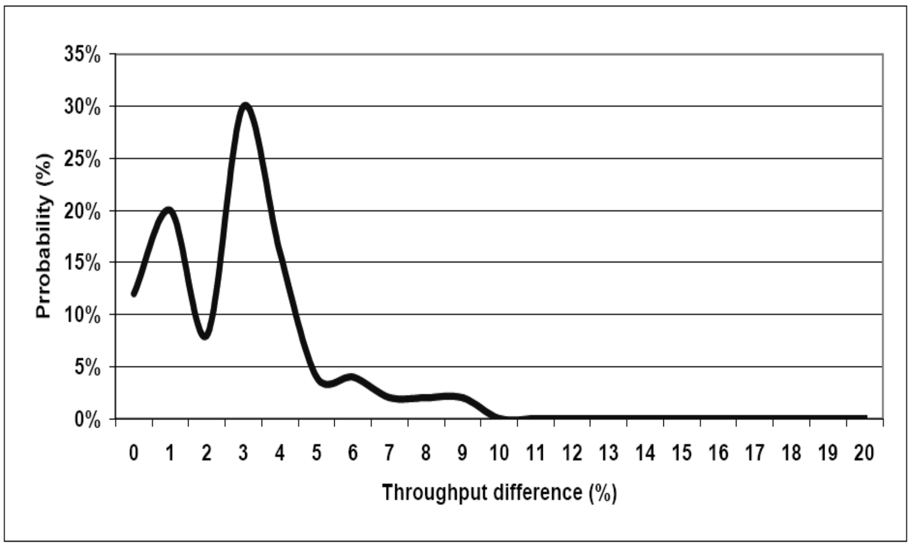


Figure 11.6: Probability of throughput differences

The results of this validation phase are reported in Figure 11.6, which shows the probability for throughput differences. The average difference between measured and predicted values is 4.7%, with 0.08 standard deviation. This confirms the high level of accuracy achieved by the developed optimization framework, thanks to the calibration of system model parameters against functional timing-accurate simulation and to the control of system working conditions.

Figure 11.7 shows that our optimizer is not only accurate within acceptable limits, but also conservative in predicting system performance, and this is very important for meeting real-time requirements. For a given problem instance, the plot compares the throughput provided by the optimizer with the simulated one for the same allocations and schedules. The range of throughputs has been spanned by progressively making the real-time constraint of the solver tighter. This latter provides an al-

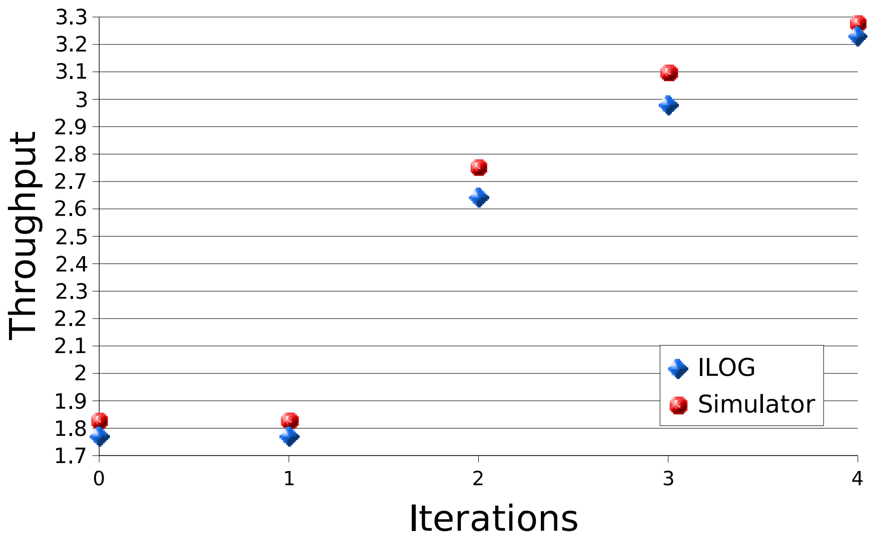


Figure 11.7: Conservative performance predictions of the optimizer

location and a schedule that are able to guarantee an entire range of throughput constraints. If a lower throughput is required, than the configuration found by the solver changes. Moving from one configuration to another corresponds to increasing steps on the x-axys. At each new point, the simulated throughput is reported as well, and it is showed to provide a conservative throughput with respect to the predicted one, within the accuracy limits found above.

11.3.4 Verifying executability

Once we have verified the correspondence between the scheduler and the simulator activity durations, we can focus our analysis on the executability of the optimal schedule, checking if the off-line schedule found by the solver can be really executed by the MPSoC platform. A schedule of tasks to be repetitively performed on a data stream of unknown length is executable only if it is periodic. A periodic schedule is defined by a priority table of finite dimension. If the table has dimension one, it is called a priority list: in this case we say that the schedule is periodic with a period of length one. In other words, called $Succ_{ij}$ the task to be executed after $Task_{ij}$, a schedule is periodic of length one if, at full rate, it is $Succ_{ij} = Succ_{ij+1}$, $\forall i = 1 \ldots n$, $\forall j \in \mathbb{N}$.

At the state of the art, we can provide our simulator only with priority lists. So, if the optimal solution is periodic with a period of length one, providing the simulator with a task priority list derived from the off-line sequence ensures that all constraints and RT requirements will be satisfied also by the on-line schedule, given that the accuracy on execution time estimation is very high, as shown in subsections 11.3.2 and 11.3.3.

We can demonstrate (see Appendix 1 at the end of this Chapter) that the optimal solutions found are always periodic, but in general the period can be longer than one. We experimentally found that, in over 90% of the cases out of a set of 200 problems, after the initial set-up stage, the first off-line schedule found by our tool is periodic with period of length one, thus executable. Concerning the remaining cases, we solved again the instances for which a periodic solution was not found inserting executability constraints in the model, in order to find an executable schedule. We measured the difference between the throughputs of the two schedules.

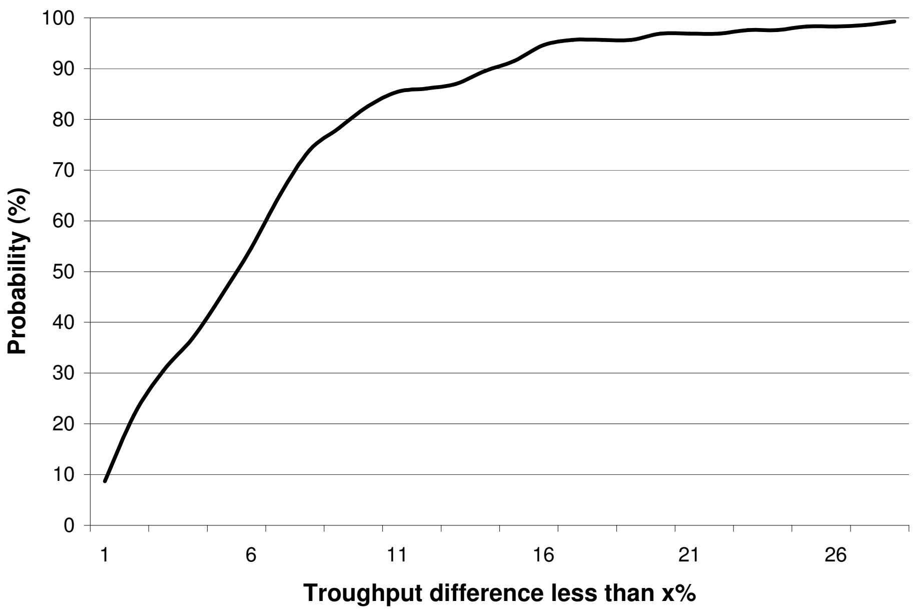


Figure 11.8: Probability of throughput difference

Figure 11.8 depicts the probability (y-axis) for the difference between the throughputs of the optimal (but not periodic with period one) solution and the periodic one to be equal or less than the corresponding value in the x-axis (in %). As an example, the probability for the difference to be less than 15% is 90%. We can see that, for most of the cases, the difference is within 10%. We recall that an optimal but not executable solution is found only in the 10% of the cases analyzed.

11.3.5 Validation on real applications

To prove the viability of our approach, we solved two ASP problems, namely the GMS Codec and the MIMO processing, and we verify the compliance of our optimal solutions with the application requirements.

GSM Codec

We first proved the viability of our approach with a GSM encoder\decoder application. Most state-of-the-art cell-phone chip-sets include dual-processor architectures. Therefore GSM encoding/decoding have been among the first target applications to be mapped onto parallel multi-processor architectures.

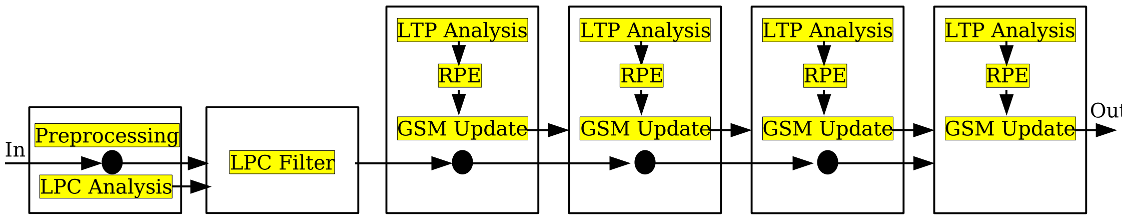


Figure 11.9: GSM codec task graph

Figure 11.9 depicts the GSM codec task graph. We can see that the source code can be parallelized into 6 pipeline stages. Each stage is grouped in a task pre-characterized by the virtual platform to provide parameters of task models to the optimizer. Such information, together with the results of the optimization run, are reported in Table 11.4. Each column reports information on the tasks in the pipeline. The second row reports the duration of the computational activity and the third the overhead to read the program data from the remote memory. This two values represent the duration of the activities A_i (the overhead considered only if program data are stored in the remote memory). The fourth and fifth lines represent the duration of the activities In_i when reading data respectively from the same processor or from another one. We do not have any task internal state in this case study. The sixth, seventh and eighth lines report respectively the dimension of the program data, input data and output data. The last two lines report the optimal allocation found: the processor where we execute the task and the memory where we allocate the program data.

The MPSoC platform we considered has 4 processors with 2KB of internal scratch-

	$Task_1$	$Tasks_2$	$Task_3$	$Task_4$	$Task_5$	$Task_6$
Computation Time (ns)	281639	437038	317032	308899	306213	306470
Remote Data Overhead (ns)	3978	1620	1099	2243	1916	1707
Local communication (ns)		4754	6675	5810	6020	5810
Remote Communication (ns)		8621	12266	10773	10609	10576
Program data (Byte)	420	420	560	560	560	560
Communication Data In (Byte)	0	340	444	444	444	444
Communication Data Out (Byte)	340	444	444	444	444	0
Processor	1	1	2	2	3	3
Data Location	Local	Local	Remote	Remote	Remote	Local

Table 11.4: GSM case study allocation

pad memory. Note that the optimizer makes use of 3 out of the 4 available processors, since it tries to minimize the cost of communication while meeting hardware and software constraints. The required throughput in this case is 1 frame/10ms, compliant with the GSM minimum requirements. The obtained throughput was 1.35 frames/ms, even more conservative. As already seen, the simulation gave a better throughput than the predicted one, with a difference of 4.1%. The table also shows that program data has been allocated in scratch-pad memory for Tasks 1,2 and 6 since they have smaller communication queues. The time taken by the optimizer to come to a solution was 0.1 seconds.

MIMO processing

One major technological breakthrough that will make an increase in data rate possible in wireless communication is the use of multiple antennas at the transmitters and receivers (Multiple-input Multiple-output systems - MIMO). MIMO technology is expected to be a cornerstone of many next-generation wireless communication systems. The scalable computation power provided by MPSoCs is progressively making the implementation of MIMO systems and associated signal processing algorithms feasible, therefore we applied our optimization framework to spatial multiplexing-based MIMO processing [98].

The MIMO computation kernel was partitioned into 5 pipeline stages. The MPSoC platform we considered has 6 processors with 4KB of internal scratchpad memory. Optimal allocation and scheduling results for a MPSoC system of 6 processors are reported in Fig.11.5. The meaning of the lines is the same of the GSM codec

	$Task_1$	$Tasks_2$	$Task_3$	$Task_4$	$Task_5$
Computation Time (ns)	526737	1633286	66385	324883	5253632
Remote Data Overhead (ns)	8683	13734	749	2279	62899
Local communication (ns)		3639	12052	5373	10215
Remote Communication (ns)		6037	17605	10615	16960
Program data (Byte)	676	2500	256	4	3136
Communication Data In (Byte)	0	256	784	400	784
Communication Data Out (Byte)	256	784	400	784	0
Processor	1	1	1	1	2
Data Location	Remote	Remote	Local	Local	Local

Table 11.5: MIMO processing allocation

case study.

The reported mapping configuration is referred to the case where the tightest feasible real-time constraint was applied to the system (about 1.26Mbit/sec). In this benchmark, Task 5 has the heaviest computation requirements, and requires a large amount of program data for its computation. In order to meet the timing requirements and to be able to allocate program data locally, this task has been allocated on a separate processor. As can be observed, the optimizer has not mapped each remaining task on a different processor, since this would have been a waste of resources providing sub-optimal results. In other words, the throughput would have been guaranteed just at the same, but at a higher communication cost. Instead, Tasks 1-4 have been mapped to the same processor. Interestingly, the sum of the local memory requirements related to communication queues leaves a very small remaining space in scratchpad memory, which allows the optimizer to map locally only the small program data of Tasks 3 and 4. The overall mapping solution was therefore not trivial to devise without the support of the combined CP-IP solver, which provides the optimal allocation and scheduling in about 600 ms. The derived configuration was then simulated onto the virtual platform, and throughput accuracy was found to be (conservatively) within 1%.

Appendix 1: Proof of schedule periodicity

In this appendix we prove that despite our algorithm considers an unbounded number j of iterations of a pipeline with n tasks $Task_{ij}$, $i = 1..n$, our final schedule is always periodic. The proof assumes single token communication queues (i.e. length one queues), but it can be easily extended to any finite length.

Tasks are partitioned by the allocation module on m processors. So let us consider m partitions: $Task_{ij}$, $\forall i \in Sp_k$, $\forall j$, where $k = 1..m$ and Sp_k is the set of tasks assigned to processor k. Our aim is to show that our (time discrete) scheduling algorithm that minimizes the makespan produces a periodic solution even if we have a (theoretical) infinite number of pipeline iterations.

The proof is based on the following idea: if we identify in the solution a state of the system that assumes a finite number of configurations, than the solution is periodic. In fact, after a given state S the algorithm performs optimal choices; as soon as we encounter S again, the same choices are performed.

For each iteration j, the state we consider is the following: the slack of each task in S_k to its deadline. The state of the system is the following: For each processor $k = 1..m$ we have $\langle Slack^k_{1j}, \dots, Slack^k_{lj}\rangle$, where $Slack^k_{ij}$ is the difference between the deadline of $Task_{ij}$ running on processor k and its completion time. Therefore, if we prove that the number of possible state configurations is finite (i.e., it does not depend on the iteration number j), being the transitions between two states deterministic, even if we have an infinite number of repetition of the pipeline, the solution is periodic.

After the pipeline starts up, the deadline of each task $Task_{ij}$ is defined by the first iteration of $Task_i$. i.e., $Task_{i1}$. In fact, the real time (throughput) constraint states that every RT time points each task should be repeated. Therefore, if the first iteration of a $Task_i$ is performed at time t_i, the second iteration of $Task_i$ should be performed at time $t_i + P$, and the j-th iteration at time $t_i + (j - 1) * P - Dur_{Task_{ij}}$.

Now, let us consider two cases:

- if the tasks in S_k are consecutive in the pipeline, then their repetition cannot change. For example, if tasks $Task_{1j}$, $Task_{2j}$ and $Task_{3j}$ are allocated to the same processor (for all j), having length one queues, they can be repeated only in this order. Indeed, one can repeat $Task_{1j}$ after $Task_{2j}$, but minimizing the makespan it is not the right decision.

- if instead the tasks in S_k are not consecutive, then there could be repetitions in between that could break the periodicity. Therefore, we should concentrate on this case.

For the sake of readability, we now omit the index representing the iteration since we concentrate on the maximum slack a task can assume. Let us consider two non consecutive tasks $T_A \in S_k$ and $T_B \in S_k$. Suppose that between T_A and T_B there are v tasks allocated on other processors different from k. Let us call them $T_{A1}, T_{A2}, \ldots T_{Av}$ ordered by precedence constraints. If we have communication queues of length one, between T_A and T_B there are AT MOST v iterations of T_A. In fact, T_A can be repeated as soon as T_{A1} starts on another processor. Also, it can be repeated as soon as another iteration of T_{A1} starts, that can happen as soon as T_{A2} starts and so on. Clearly, v iterations are possible only if

$$m * Dur_{T_A} \leq \sum_{i=1}^{m} Dur_{T_{Ai}}$$

but if this relation does not hold, there can be only less iterations of T_A. Therefore, v is an upper bound on the number of iterations of T_A between the first T_A and T_B. If t_A is the time where the first repetition of T_A is performed, the v^{th} iteration of T_A has a deadline of $t_A + (v-1) * P$. Its slack is clearly bounded to the maximum deadline minus its duration, $t_A + (v-1) * P - Dur_{T_A}$.

The upper bound for v is $n-2$. In fact, in a pipeline of n tasks the maximum number of repetitions of a task happen if only the first and the last task are allocated on the same processor. They have $n-2$ tasks in between allocated on different processors. Therefore, the maximum number of repetitions of T_1 between T_1 and T_n is $n-2$.

Therefore if the first iteration of T_1 is executed at time t_1 its $(n-2)^{th}$ iteration has a max deadline $t_1 + (n-3) * P - Dur_{T_1}$.

Being the max deadline of a task finite, also its max slack is finite despite the number of iteration of the pipeline.

Therefore, whatever the state is, each task belonging to the state has a finite slack. The combination of slacks are finite, and therefore, after a finite number of repetition, the system finds a state already found and becomes periodic.

Chapter 12

DVSP Model

Introduction

In this Chapter we will analyze the DVSP, describing the model we used to solve it, as well as the modelling assumption we have done. In Section 12.1 we will formalize our problem and in Section 12.2 we will present the complete model for the DVSP.

12.1 Dynamic Voltage Scaling Problem description

The new MPSoC paradigm for hardware platform design is pushing the parallelization of applications, so that instead of running them at a high frequency on a single monolithic core, they can be partitioned into a set of parallel tasks, which are mapped and executed on top of a set of parallel processor cores operating at lower frequencies. Power minimization is a key design objective for MPSoCs to be used in portable, battery-operated devices. This goal can be pursued by means of low power design techniques at each level of the design process, from physical-level techniques (e.g., low swing signaling) up to application optimization for low power. Here we focus on system-level design, where the main knobs for tuning power dissipation of an MPSoC are: allocation and scheduling of a multi-task application onto the available parallel processor cores, voltage and frequency setting of the individual processor cores. For those systems where the workload is largely predictable and not subject to run-time fluctuations (e.g., signal processing or some multimedia applications), the above design parameters can be statically set at design time. Traditional ways to tackle the mapping and configuration problem either incur overly large computation times already for medium-size task sets, or are inaccurate (e.g., use of heuristics and problem modelling with highly simplifying assumptions on system operation). Therefore, design technology for MPSoCs strongly needs accurate, scalable and composable modelling and solving frameworks.

We consider the energy-aware MPSoC platform described in Section 9.1. In real-life MPSoC platforms, switching voltage and frequency of a processor core is not immediate nor costless, therefore the switching overhead in terms of switching delay (referred to as setup times) and energy overhead (referred to as setup costs) must be carefully considered when selecting the optimal configuration of a system. In practice, interesting trade-offs have to be studied. On one hand, tasks can be spread across a large number of processor cores, so that these cores can operate at lower frequencies, but more communication arises and the energy cost of many running cores has to be compensated by a more energy-efficient execution of tasks. On the other hand, tasks have to be grouped onto the processor cores and scheduled taking care of minimizing the number of frequency switchings. It must be observed that application real-time requirements play a dominant role in determining solutions for the MPSoC mapping and configuration problem. A good methodology should be conservative with respect to task deadlines, so to minimize the probability of timing violations in the real system.

Similarly to the ASP, the application we should allocate and schedule are represented by a directed acyclic task graph G whose nodes represent a set of T tasks, are annotated with their deadline dl_t and with the worst case number of clock cycles WCN_t (the execution time depends on the working frequency). Arcs represent dependencies/communications among tasks. Each arc is annotated with the amount of data two dependent tasks should exchange, and therefore the number of clock cycles for exchanging (reading and writing) these data WCN_R and WCN_W. Tasks are running on a set of processors P. Each processor can run with M energy/speed modes and has a maximum load constraint dl_p. Each task spends energy both in computing and in communicating. In addition, when a processor switches between two modes it spends time and energy. We have both energy overhead E_{ij} and time overhead T_{ij} for switching from frequency i to frequency j.

The Dynamic Voltage Scaling Problem (DVSP) is the problem of allocating tasks to processors, define the running speed of each task and schedule each of them minimizing the total energy consumed.

Similarly to the ASP, the method we use for handling the DVSP is based on the Logic-Based Benders Decomposition technique. The problem is decomposed into master and sub-problem: the former is the allocation of processors and frequencies to tasks and the latter is the scheduling of tasks given the static allocation and frequency assignments provided by the master. Note that the frequency assignment could be done in the subproblem. However, the scheduling part becomes extremely slow and performances highly decrease because we have to deal with a scheduling problem with variable durations. In addition, we will see in Section 12.2.5 that the relaxation of the subproblem introduced in master problem becomes extremely loose. Differently from the ASP, the objective function depends both on master and subproblem variables. In fact, the master problem minimizes the communication and execution energy, while only during the scheduling phase we could minimize the energy overhead for frequency switching.

The master problem is tackled by an Integer Programming solver (through a traditional Branch and Bound) while the subproblem through a Constraint Programming solver. As described in Section 9.4, the two solvers interact via no-good and cutting planes generation. The solution of the master problem is passed to the subproblem. We have two possible cases: (1) there is no feasible schedule: we have to compute a no-good avoiding the same allocation to be found again; (2) there is a feasible and optimal schedule minimizing the second component of the objective function: here we cannot simply stop the iteration since we are not sure to have the optimal solution overall. We have to generate a cut saying that this is the optimal solution unless a better one can be computed with a different allocation.

The procedure converges when the master problem produces a solution with the same objective function of the previous one.

12.2 Modelling the Dynamic Voltage Scaling Problem

In this Section we will describe the master and the sub-problem models for the DVSP, as well as the Benders cut and relaxations we used. Before introducing the models, we will give an example of a DVSP instance.

12.2.1 DVSP example

As an example, let consider 5 tasks and 5 communications, with the precedence constraints as described in Figure 12.1. Table 12.1 shows the duration (in clock cycles) of execution and communication tasks (the durations of the reading and the writing phase R_i and W_i of each communication Com_i are the half of these values). We have 2 processors, that can run at 2 different frequencies, 200MHz and 100MHz (so, e.g. $Task_1$ will last 500ns if runs at 200MHz and 1μs if runs at 100MHz). The processors waste 10mW when running at 200MHz and 3mW when running at 100MHz. Switching from the higher frequency to the lower needs 2ns and wastes 2pJ, while the contrary needs 3ns and wastes 3pJ. The realtime requirement settles the processor deadline at 2μs.

Nome	Task1	Task2	Task3	Task4	Task5	Com1	Com2	Com3	Com4	Com5
Clock	100	54	134	24	10	20	10	8	8	8

Table 12.1: Activities durations for the example

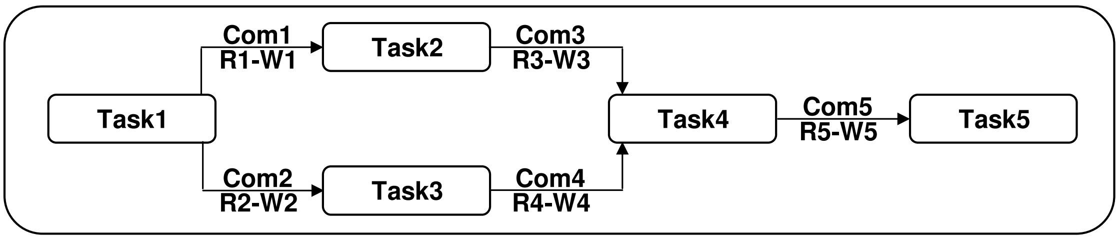


Figure 12.1: Task graph for the example in Table 12.1

The first allocation found minimizing the power consumption tries to assign the lower frequency to the third task, being the longest one and thus the most power consuming one; this solution is however not schedulable due to the deadline constraint. The second allocation found is schedulable and is also the optimal one w.r.t. the power consumption minimization. The first two tasks are allocated on the first processor at the higher frequency and the other three tasks on the second processor: here only $Task_5$ runs at the higher frequency. The total power consumption is 13502mW. The Gantt chart in Figure 12.2 shows the schedule of this solution.

We have seen that, in this simple example, the master and sub-problem solvers must iterate two times to find the optimal solution. This depends on the fact that the task graph is generic and the task graph can contain several communication

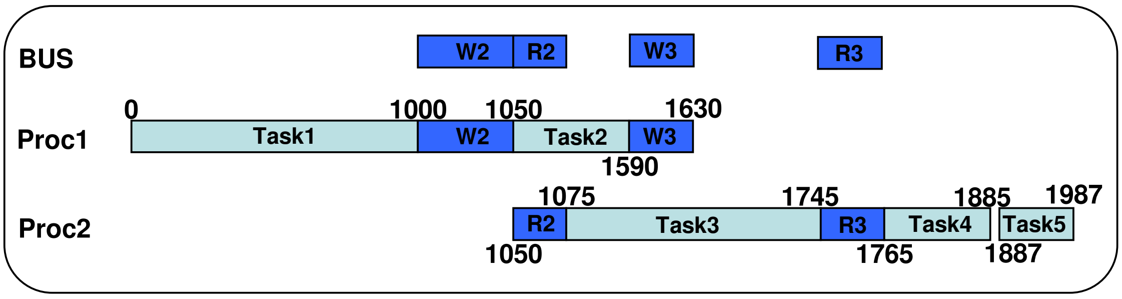


Figure 12.2: Schedule for the example in Table 12.1

chains introducing overheads considered only while solving the subproblem; furthermore, the time overhead for frequency switching is higher w.r.t. the time overhead for communication considered in the ASP and most probably this can cause an infeasibility. In addition, the objective function depends on both the problems, and this complicates the model. In the following Chapter we will show experimental results when considering both pipelined and generic tasks graphs and we will see that the number of iterations is typically fairly higher than one, a quite typical value for the ASP.

12.2.2 Allocation and voltage selection problem model

We model the allocation problem with binary variables X_{ptm} taking value 1 if task t is mapped on the processor p and runs at mode m, 0 otherwise. Since we also take into account communications, we assume that two communicating tasks running on the same processor do not consume any energy and do not spend any time (indeed the communication time and energy spent are included in the execution time and energy), while if they are allocated on two different processors, they both consume energy and spend time. The first task spends time and energy for writing data on a shared memory. This operation makes the duration of the task becoming longer: it increases of a quantity WCN_W/f_m where WCN_W is the number of clock cycles for writing data (it depends on the amount of data we should write), and f_m is the clock frequency when task t is performed. The second task should read data from the shared memory. Again its duration increases of a quantity WCN_R/f_m where WCN_R is the number of clock cycles for reading data (it depends on the amount of data we should read), and f_m is the clock frequency when task t is performed.

Both the read and write activities are performed at the same speed of the task and use the bus (which instead works at the maximum speed). For modelling this aspect, we introduce in the model two variables $R_{pt_1t_2m}$ and $W_{pt_1t_2m}$ taking value 1 if the task t_1 running on processor p reads (resp. writes) data at mode m from (resp. for) task t_2 not running on p.

Any task can be mapped on only one processor and can run at only one speed. This translates in the following constraints:

$$\sum_{p=1}^{P}\sum_{m=1}^{M} X_{ptm} = 1 \ , \ \forall t \tag{12.1}$$

Also the communication between two tasks happens at most once:

$$\sum_{p=1}^{P}\sum_{m=1}^{M} R_{pt_1t_2m} \leq 1 \ , \ \forall t_1, t_2 \tag{12.2}$$

$$\sum_{p=1}^{P}\sum_{m=1}^{M} W_{pt_1t_2m} \leq 1 \ , \ \forall t_1, t_2 \tag{12.3}$$

The objective function is to minimize the energy consumption of the task execution, and of the task communication (read and write)

$$E_{comp} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t=1}^{T} X_{ptm} WCN_t t_{clock_m} P_{tm} \tag{12.4}$$

$$E_{Read} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t,t_1=1}^{T} R_{ptt_1m} WCN_{Rtt_1} t_{clock_m} P_{tm} \tag{12.5}$$

$$E_{Write} = \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{t,t_1=1}^{T} W_{ptt_1m} WCN_{Wtt_1} t_{clock_m} P_{tm} \tag{12.6}$$

where P_{tm} is the power consumed in a clock cycle (lasting t_{clock_m}) by the task t at mode m.

$$OF = E_{comp} + E_{Read} + E_{Write} \tag{12.7}$$

The objective function defined up to now depends only on master problem variables. However, switching from one speed to another introduces transition costs, but their value can be computed only at scheduling time. In fact, they are not constrained in the master problem original model. They are constrained by Benders Cuts instead, after the first iteration. We will present Benders Cuts in section 12.2.4. Therefore, in the master problem the objective function is:

$$OF_{Master} = OF + Setup \tag{12.8}$$

$$Setup = \sum_{p=1}^{P} Setup_p \tag{12.9}$$

It is worth noting that this contribution should be added to the master problem objective function, but, being the $Setup_p$ variables not constrained at the first iteration in the master problem, they are all forced to be 0. From the second iteration, instead, cuts are produced constraining variables $Setup_p$ and this contribution could be no longer 0.

This formulation will result in tasks that are potentially running initially with lower frequencies on the same processor (thus avoiding communication). A measure of control is provided by constraints on deadlines in order to prevent the blind selection of the lowest frequencies and the allocation of all tasks on the same processor. The timing is not yet known in this phase, but we can introduce some constraints that represent a relaxation of the subproblem and will reduce the solution space. For each processor, only a certain load is allowed. Therefore, on each processor the sum of the time spent for computation, plus the time spent for communication (read and write) should be less than or equal to the processor deadline dl_p:

$$T^p_{comp} = \sum_{t=1}^{T} \sum_{m=1}^{M} X_{ptm} \frac{WCN_t}{f_m} \tag{12.10}$$

$$T^p_{read} = \sum_{t=1}^{T} \sum_{m=1}^{M} \sum_{t_1=1}^{T} R_{ptt_1m} \frac{WCN_{Rtt_1}}{f_m} \tag{12.11}$$

$$T^p_{write} = \sum_{t=1}^{T} \sum_{m=1}^{M} \sum_{t_1=1}^{T} W_{ptt_1m} \frac{WCN_{Wtt_1}}{f_m} \tag{12.12}$$

$$T^p_{comp} + T^p_{read} + T^p_{write} \leq dl_p \ , \ \forall p \tag{12.13}$$

These relaxations can be tightened by considering chains of tasks in the task graph instead of groups of tasks running on the same processor. For example, let us consider four tasks t_1, t_2, t_3, t_4 linked by precedence constraints so that $t_1 \rightarrow t_2$,

$t_2 \rightarrow t_3$ and $t_3 \rightarrow t_4$. Now suppose that t_1 and t_4 are allocated on processor 1 and t_2 and t_3 on other processors. Instead of summing only the durations of t_1 and t_4 that should be less than or equal to the processor deadline, one could add also the duration of t_2 and t_3 since they should be executed before t_4. The chains in a graph can be many, we added only some of them.

Finally, task deadlines can be captured:

$$\sum_{p=1}^{P}\sum_{m=1}^{M}\left[X_{ptm}\frac{WCN_t}{f_m}+\sum_{t1=1}^{T}\left(R_{ptt_1m}\frac{WCN_{Rtt_1}}{f_m}+W_{ptt_1m}\frac{WCN_{Wtt_1}}{f_m}\right)\right]\leq dl_t \ ,\forall t \tag{12.14}$$

There are several improvements we have introduced in the master problem model. In particular we have removed many symmetries leading the solver to explore the same configurations several times.

12.2.3 Scheduling problem model

Once allocation and voltage selection have been solved optimally, for the scheduling part each task t has an associated variable representing its starting time $Start_i$. The duration is fixed since the frequency is decided, i.e., $duration_i = WCN_i/f_i$. In addition, if two communicating tasks t_i and t_j are allocated on two different processors, we should introduce two additional activities (one for writing data on the shared memory and one for reading data from the shared memory). We model the starting time of these activities $StartWrite_{ij}$ and $StartRead_{ji}$. These activities are carried on at the same frequency of the corresponding task. If t_i writes and t_j reads data, the writing activity is performed at the same frequency of t_i and its duration $dWrite_{ij}$ depends on the frequency and on the amount of data t_i writes, i.e., WCN_{Wij}/f_i. Analogously, the reading activity is performed at the same frequency of t_j and its duration $dRead_{ji}$ depends on the frequency and on the amount of data t_j reads, i.e., $WCN_{R_{ji}}/f_j$. Clearly the read and write activities are linked together and to the corresponding task:

$$StartWrite_{ij} + dWrite_{ij} \leq StartRead_{ji}, \forall i,j \text{ s.t. i communicates with j} \tag{12.15}$$

$$Start_i + duration_i \leq StartWrite_{ij}, \forall i,j \text{ s.t. i communicates with j} \tag{12.16}$$

$$StartRead_{ji} + dRead_{ji} \leq Start_j, \forall i,j \text{ s.t. i communicates with j} \tag{12.17}$$

In the subproblem, we model precedence constraints in the following way: if task t_i should precede task t_j and they run on the same processor at the same frequency the precedence constraint is simply:

$$Start_i + duration_i \leq Start_j \tag{12.18}$$

If two tasks run on different processors and should communicate we should add the time for communicating.

$$Start_i + duration_i + dWrite_{ij} + dRead_{ji} \leq Start_j \tag{12.19}$$

Deadline constraints are captured stating that each task must end its execution before its deadline and, on each processor, all the tasks (and in particular the last one) running on it must end before the processor deadline.

$$Start_i + duration_i \leq dl_{t_i} \ , \ \forall i \tag{12.20}$$

$$Start_i + duration_i \leq dl_p \ , \ \forall i \in p, \ \forall p \tag{12.21}$$

Resources are modelled as follows. We have a unary resource constraint for each processor, modelled through a cumulative constraint having as parameters a list of all the variables representing the starting time of the activities (tasks, readings, writings) sharing the same resource p, their durations, their resource consumption (which is a list of 1) and the capacity of the processor which is 1.

$$cumulative(StartList_p, DurationList_p, [1], 1) \ , \ \forall p \tag{12.22}$$

We model the bus through the additive model we have presented in Chapter 10 and validated in Chapter 11. We have an activity on the bus each time a task writes or reads data to or from the shared memory. The bus is modelled as an additive resource and several activities can share the bus, each one consuming a fraction of it until the total bandwidth is reached. The cumulative constraint used to model the bus is:

$$cumulative(StartReadWriteList, DurationList, Fraction, TotBWidth) \tag{12.23}$$

where *StartReadWriteList* and *DurationList* are lists of the starting times and durations of all read and write activities needing the bus, $Fraction$ is the amount of bandwidth granted to an activity when accessing the bus[1] and $TotBWidth$ is the total bandwidth available of the bus.

[1]This value was experimentally tuned to 1/4 of the total bus bandwidth.

To model the setup time and cost for frequency switching we take advantage of the classes defined by ILOG Scheduler [64] to manage transitions between activities. It is possible to associate a label to each activity and to define a transition matrix that specifies, for each couple of labels l_i and l_j, a setup time and a setup cost that must be paid to schedule, on the same resource, an activity having the label l_i just before an activity having the label l_j. When, during the search for a solution, two activities with labels l_i and l_j are scheduled one just after the other on the same resource, the solver will satisfy the additional constraint:

$$Start_{l_i} + duration_{l_j} + TransTime_{l_i l_j} \leq Start_{l_j} \tag{12.24}$$

where $TransTime_{l_i l_j}$ is the setup time specified in the transition matrix. Likewise, the solver introduces $TransCost_{l_i l_j}$ in the objective function. If S_p is the set of all the tasks scheduled on processor p, the objective function we want to minimize is:

$$OF = \sum_{p=1}^{P} \sum_{(i,j)\in S_p | next(i)=j} TransCost_{l_i l_j} \tag{12.25}$$

12.2.4 Generation of Logic-based Benders Cuts

Once the subproblem has been solved, we generate Benders Cuts. The cuts are of two types:

- if there is no feasible schedule given an allocation, the cuts are the same we computed for the ASP and depend on variables X_{ptm}.
- if the schedule exists, we cannot simply stop the iteration since the objective function depends also on subproblem variables. Therefore, we have to produce cuts saying that the one just computed is the optimal solution unless a better one exists with a different allocation. These cuts produce a lower bound on the setup of single processors.

The first type of cuts are no-good: we call J_p the set of couples (Task, Frequency) allocated to processor p. As shown in Section 10.3.4, we can impose cuts avoiding to find the same allocation again:

$$\sum_{p=1}^{P} \sum_{(t,m)\in J_p} X_{ptm} \leq T - 1 \tag{12.26}$$

or, solving a one-machine scheduling for each processor to find the set of conflicting resources CR where the infeasibility happens, we can impose the tighter cuts:

$$\sum_{(t,m)\in J_p} X_{ptm} \leq |J_p| - 1 \ , \ \forall p \in CR \tag{12.27}$$

Let us concentrate on the second type of cuts. The cuts we produce in this case are bounds on the variable *Setup* defined in the Master Problem and introduced in equation (12.9).

Suppose the schedule we find for a given allocation has an optimal setup cost $Setup^*$. It is formed by independent setups, one for each processor $Setup^* = \sum_{p=1}^{P} Setup_p^*$.

We have a bound on the setup LB_{Setup_p} on each processor and therefore a bound on the overall setup $LB_{Setup} = \sum_{p=1}^{P} LB_{Setup_p}$.

$$Setup_p \geq 0 \ , \ \forall p \tag{12.28}$$

$$Setup_p \geq LB_{Setup_p} \ , \ \forall p \tag{12.29}$$

$$LB_{Setup_p} = Setup_p^* - Setup_p^* \sum_{(t,m)\in J_p} (1 - X_{ptm}) \ , \ \forall p \tag{12.30}$$

These cuts remove only one allocation. Indeed, we have also produced cuts that remove some symmetric solutions.

We have devised tighter cuts removing more solutions. Intuitively, each time we consider a solution of the problem overall, we generate an optimal setup cost $Setup^*$ for the given allocation. In the current solution, we know the number of frequency switches producing $Setup^*$. We can consider each processor independently since the frequency switches on one processor are independent from the other. We can impose cuts saying that $Setup^*$ is bound for all solutions with the same set of frequency switches of the last one found or a superset of it. To do that we have to introduce in the model variables $Next_{t_1 t_2 f_1 f_2 p}$, taking value 1 if, on processor p, task t_1, running at frequency f_1, executes just before t_2, running at frequency f_2. This variables complicate the model too much. In fact, our experimental results show that these cuts, even if tighter, do not lead to any advantage in terms of computational time.

12.2.5 Relaxation of the subproblem

The iterative procedure presented so far can be improved by adding a bound on the setup cost and setup time in the master problem based only on information derived from the allocation.

Suppose we have five tasks running on the same processor using three different frequencies. So for instance, tasks t_1, t_3 and t_5 run at frequency f_1, t_2 runs at frequency f_2 and t_4 runs at frequency f_3. Since we have to compute a bound, we suppose that all tasks running at the same speed go one after the other. We can have six possible orders of these frequencies leading to different couples of frequency switches. A bound on the sum of the energy spent during the frequency switches is the minimal sum between two switches, i.e., the sum of all possible switches minus the costliest one. This bound is extremely easy to compute and does not enlarge the allocation problem model.

Let us introduce in the model variables Z_{pf} taking value 1 if the frequency f is allocated at least once on the processor p, 0 otherwise. Let us call E_f the minimum energy for switching to frequency f, i.e. $E_f = min_{i,i \neq f}\{E_{if}\}$.

$$Setup_p \geq \sum_{f=1}^{M} (Z_{pf}E_f - max_f\{E_f | Z_{pf} = 1\}) \ , \ \forall p \tag{12.31}$$

This bound (referred to as Energy relaxation in the following) helps in reducing the number of iterations between the master and the subproblem.

Similarly, we can compute the bound on the setup time given an allocation. Let us consider $T_f = min_{i,i \neq f}\{T_{if}\}$. Therefore, we can compute the following bound.

$$SetupTime_p \geq \sum_{f=1}^{M} (Z_{pf}T_f - max_f\{T_f | Z_{pf} = 1\}) \ , \ \forall p \tag{12.32}$$

This bound (referred to as Time relaxation in the following) can be used to tighten the constraint 12.13 in the following way.

$$T^p_{comp} + T^p_{read} + T^p_{write} + SetupTime_p \leq dl_p \ , \ \forall p \tag{12.33}$$

so that solutions provided by the master problem are more likely to be feasible for the subproblem.

A tighter bound on the setup time and cost could be achieved by introducing in the allocation problem model variables $Next_{t_1 t_2 f_1 f_2 p}$, but as explained in section 12.2.4 they complicate the model too much and are not worth using.

Chapter 13

DVSP Experimental Results

Introduction

In this Chapter we will show the experimental results obtained when solving the DVSP. In Section 13.1 we will show the experimental results obtained when solving the problem using a hybrid solver based on both the IP and CP models described in the last Chapter implementing the Logic-Based Benders Decomposition (LB-BD) methodology. In section 13.2 we will discuss about the effectiveness of the sub-problem relaxations introduced in the master problem showing some results obtained when solving the problem using different relaxations. Section 13.3 is devoted at validating our approach by measuring the accuracy of the solutions found and their executability on the real platform.

13.1 Experimental Results

In this Section we will show the experimental results obtained when solving DVSP instances. We will first validate our approach comparing the decomposition-based solver with pure solver based only on IP or CP. For the DVSP, we will analyze both pipelined (as for the ASP) and generic task graphs.

13.1.1 Data set

We have generated 500 realistic instances, with the number of tasks varying from 4 to 10 and the number of processors from 2 to 10. We assume that each processor can run at three different frequencies. We consider, similarly to the ASP, applications with a pipeline workload. Therefore we refer to the number of tasks to be allocated and we schedule a larger number of tasks corresponding to many iterations of the pipeline. We also have generated 27 realistic instances with the number of tasks varying from 8 to 14 and the number of processors from 2 to 6, with generic task graphs. The generic task graph complicates the problem since it increases the parallelism degree. We assume that each processor can run at six different frequencies. All the considered instances are solvable and we found the proved optimal solution for each of them. Experiments were performed on a 2.4GHz Pentium 4 with 512 Mb RAM. We used ILOG CPLEX 8.1 [63], ILOG Solver 5.3 [65] and ILOG Scheduler 5.3 [64] as solving tools.

13.1.2 Comparison with pure approaches

In Chapter 13, we compared our Hybrid solving tool for the ASP with pure CP or IP based solving tools. Results shown that the pure approaches were not comparable with the Hybrid one, being the search times for finding a solution to a relaxed (thus easier) problem orders of magnitude higher. The DVSP is much more complex then the ASP, since we consider also frequency switching. We developed a CP and an IP-based approach to solve allocation, scheduling and voltage selection, but not even a single (feasible) solution was found within 15 minutes, while the Hybrid approach, within 4 minutes, finds the optimal solution and proves optimality for all the pipelined instances considered. This results validate to a greater extent the decomposition-based approach already validated for the ASP.

13.1.3 Experimental results

In this section we show the results obtained solving DVSP instances using the model described in section 12.2. We consider first instances with the task graphs representing a pipeline workflow. Note that here, since we are considering applications with pipeline workload, if n is the number of tasks to be allocated, the number of scheduled tasks is n^2. Results are summarized in Table 13.1. The first three columns contain the number of allocated and scheduled activities (execution+communication data writes and reads) and the number of processors considered in the instances (we remind that each processor can run at three different frequencies). The last two columns represent respectively the search time and the number of iterations. Each value is the mean over 10 instances with the same number of tasks and processors. We can see that for all the instances the optimal solution can be found within four minutes. The number of iterations is typically low, but nevertheless higher than the value 1 as for the ASP. Table 13.2 shows the percentage of occurrence of a given number of iterations. We can see that the optimal solution can be found at the first step in one half of the cases and the number of iterations is at most 5 in almost the 90% of cases. This result is due to the tight relaxations added to the master problem model. In Section 13.2 we will show the importance of the relaxations used.

Activities					**Activities**				
Alloc	**Sched**	**Procs**	**Time (s)**	**Iters**	**Alloc**	**Sched**	**Procs**	**Time (s)**	**Iters**
4+6	16+24	2	1,73	1,98	7+12	49+84	7	34,53	6,34
4+6	16+24	3	1,43	2,91	8+14	64+112	2	4,09	3,28
4+6	16+24	4	2,24	3,47	8+14	64+112	3	10,99	1,83
5+8	25+40	2	2,91	2,36	8+14	64+112	4	12,34	4,45
5+8	25+40	3	4,19	4,12	8+14	64+112	5	22,65	10,53
5+8	25+40	4	5,65	4,80	8+14	64+112	7	51,07	6,98
5+8	25+40	5	6,69	3,41	9+16	81+144	2	1,79	1,12
6+10	36+60	2	3,84	2,90	9+16	81+144	5	60,07	7,15
6+10	36+60	3	10,76	2,17	9+16	81+144	6	70,40	9,20
6+10	36+60	4	15,25	4,66	10+18	100+180	2	5,52	1,83
6+10	36+60	5	23,17	4,50	10+18	100+180	3	3,07	1,96
6+10	36+60	6	26,14	3,66	10+18	100+180	6	120,02	6,23
7+12	49+84	2	4,67	1,75	10+18	100+180	10	209,35	10,65
7+12	49+84	3	5,90	1,90					

Table 13.1: Search time and number of iterations for instances with pipelined task graphs

Iter	1	2	3	4	5	6	7	8	9	10	11+
%	50,27	18,51	7,11	4,52	4,81	2,88	2,46	2,05	1,64	1,64	4,11

Table 13.2: Number of iterations distribution ratio

We extended our analysis to instances where the task graph is a generic one, so an activity can possibly read data from more than one preceding activity and possibly write data that will be read by more than one following activity. The number of reading and writing activities can become considerably higher, being higher the number of edges in the task graph. We consider here processors that can run at six different frequencies, so the number of alternative resources a task can use is six times the number of processors. Differently from the pipelined instances, here we schedule a single repetition of each task. Table 13.3 summarizes the results. Each line represents an instance that has been solved to optimality. Columns have the same meaning as those already described in Table 13.1. The number of communications in this case in not equal to $2 \times (n-1)$ as for the pipelined instances, but depends on the instance task graph. We can see that typically the behaviors are similar to those found when solving the pipelined instances, but we can note some instances where the number of iterations or the search time is notably higher. For example, in the last but two line the number of iterations is very high: this is due to the particular structure of the task graph; in fact it can happens that a high degree of parallelism between the tasks, that is a high number of tasks that can execute only after a single task, leads to a number allocations that are not schedulable. The master problem solver thus looses time proposing to the scheduler a high number of unfeasible allocation. Introducing in the master problem model some relaxations coming from an analysis of the task graph structure, and in particular from the precedence constraints, can lead to better results.

On the contrary, in the last line the number of iterations is low but the search time is extremely high: this is due to the tasks characteristics that make the scheduling problem very hard to be solved.

13.2 Effectiveness of the sub-problem relaxations

To show the effectiveness of the relaxations used for the DVSP we solved the instances considering either both or only one of the two relaxations (Energy and Time) described in 12.2.5. Table 13.4 shows the percentage of occurrence of a given number of iterations when solving the pipelined DVSP instances with different relaxations. As already shown in Section 13.1.3, using both of the relaxations (row Both), we found the optimal solution at the first step in one half of the cases and the number of iterations is at most 5 in almost the 90% of cases. We tried to solve the same instances using only one relaxation; rows Time and Energy show the results when considering only the relaxation on the deadlines and on the sub-problem objective function lower bound respectively. We can see that, for most of the cases, the num-

Activities					Activities				
Alloc	**Sched**	**Procs**	**Time(s)**	**Iters**	**Alloc**	**Sched**	**Procs**	**Time(s)**	**Iters**
8+16	8+16	2	1,57	1	9+16	9+16	4	29,59	26
8+12	8+12	3	1,48	2	9+16	9+16	4	4,84	6
8+16	8+16	3	0,81	1	9+20	9+20	6	158,43	39
8+12	8+12	3	4,26	6	10+18	10+18	2	5,90	1
8+16	8+16	4	0,86	1	10+18	10+18	3	2,12	1
9+24	9+24	2	2,51	1	10+16	10+16	3	12,81	3
9+12	9+12	2	1,11	1	10+12	10+12	4	0,37	1
9+8	9+8	2	2,73	3	10+16	10+16	4	13,92	14
9+16	9+16	3	35,95	43	10+24	10+24	4	4,18	5
9+20	9+20	3	2,51	1	10+12	10+12	4	11,50	27
9+22	9+22	3	6,62	2	12+20	12+20	5	551,92	213
9+12	9+12	4	1,40	3	14+22	14+22	2	14,11	1
9+12	9+12	4	2,14	5	14+62	14+62	6	3624,81	2
9+10	9+10	4	2,60	4					

Table 13.3: Search time and number of iterations for instances with generic task graphs

ber of iterations is higher than 10. In addiction, experimental results showed that, on average, the search time rises up to 1 order of magnitude and, in the worst cases, the solution cannot be found within two hours.

Iter	1	2	3	4	5	6	7	8	9	10	11+
Both	50,27	18,51	7,11	4,52	4,81	2,88	2,46	2,05	1,64	1,64	4,11
Time	35,23	10,32	3,47	4,76	3,12	2,84	2,13	2,06	1,04	1,11	33,92
Energy	28,6	10,12	5,64	3,78	4,35	2,91	1,29	1,48	1,12	0,84	39,87

Table 13.4: Number of iterations distribution ratio with different relaxations

13.3 Validation of the results

In Section 13.1 we have given evidence that our tool is efficient and scalable for solving to optimality the DVSP problem. In this Section we will validate our results simulating them on the cycle accurate MPSoC simulator MP-ARM [1].

For each task in the input graph we need to extract the task execution time, the time required for writing and for reading input data from local memory and the overhead due for writing and reading input data if queues are allocated onto remote shared memory. This information are collected running simulations on MP-ARM.

The optimizer assumes that mapping two tasks onto the same processor is always more energy-efficient than having them on separate processors, since message exchange through scratchpad memory is less power-hungry than shared memory communication [91]. In the optimization problem, it is enough to model the cost for communication in the objective function of the master problem as the additional energy incurred by a producer/consumer pair for the message exchange time. As a result, during the validation step we have to compare the deviation of predicted values of the objective function (i.e., the energy dissipation of processor cores) with respect to the simulation statistics.

We have performed two types of experiments, namely (i) measurement of deviations of simulated energy consumption and application throughput from the values obtained by the optimizer for 200 synthetic problem instances, (ii) showing the viability of the proposed approach by means of real life demonstrators (GSM, JPEG).

13.3.1 Validation of optimizer solutions

We have deployed the virtual platform to implement the allocations, schedules and frequency assignments generated by the optimizer. A tunable multi-task application has been used for this experiment, allowing to change system and application parameters (local memory size, execution times, data size, real-time requirements, etc.) and generate 200 problem instances we used for validation. In Section 11.3 we have already validated the bus additive model and we have verified the throughput accuracy of our tool. Here we measure the difference between the energy consumption found by the optimizer and the simulator, the most important parameter for the DVSP. The results are reported in Figure 13.1, which shows the distribution of the energy consumption difference. The average difference between measured and predicted energy values is 2.9%, with 1.72 standard deviation. This confirms the high level of accuracy achieved by the developed optimization framework in modelling real-life MPSoC systems with the assumed architectural template.

13.3.2 Validation on real applications

To prove the viability of our approach, we solved two DVSP problems, namely the GMS Codec and the JPEG decoder, and we verify the compliance of our optimal

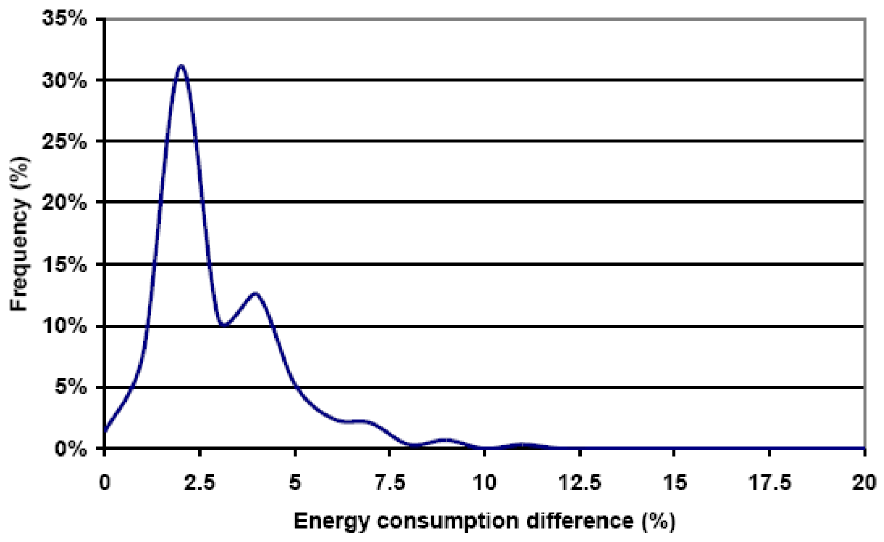


Figure 13.1: Distribution of energy consumption differences

solutions with the application requirements.

GSM Codec

We consider again the GSM Codec application parallelized into 6 pipeline stages depicted in Figure 11.9. Each task has been pre-characterized by the virtual platform to provide parameters of task models to the optimizer. After the optimization stage, the validation process on the virtual platform showed an accuracy on the processor energy dissipation, as predicted by the optimizer, by 2%.

We deployed the GSM demonstrator to explore how the optimizer minimizes energy dissipation of the processor cores with varying real-time requirements. The behaviour of the optimizer is not specific for the GSM case study, but can be extended to all applications featuring timing constraints. Results for the GSM case study are reported in Table 13.5, where allocations and frequency assignments are given for different values of the deadline constraint. The allocation is given as an array indicating the processor ID on which each task is mapped. Similarly, the frequency of each task is expressed in terms of the integer divider of the baseline frequency. Only 3 dividers are used for this example, i.e. the processors can run at only 3 different speeds.

When the deadline is loose, all tasks are allocated to one single processor at the

Deadline (ns)	Number of processors	Allocation	Frequency Assignment	Energy consumption (nJ)
6000	1	1,1,1,1,1,1	3,3,3,3,3,3	5840
5500	2	2,1,1,1,1,1	3,3,3,3,3,3	5910
5000	2	1,1,1,1,1,2	3,3,3,3,3,3	5938
4500	2	1,1,1,1,2,2	3,3,3,3,3,3	5938
4000	2	1,1,1,2,2,2	3,3,3,3,3,3	5938
3500	2	1,1,1,2,2,2	3,3,3,3,3,3	5938
3000	3	1,2,2,3,3,3	3,3,3,3,3,3	6008
2500	3	1,2,3,3,4,4	3,3,3,3,3,3	6039
2000	4	1,2,3,4,5,6	3,3,3,3,3,3	6109
1500	6	1,2,3,4,5,6	3,3,3,3,3,3	6304
1000	6	1,2,3,4,5,6	3,2,2,2,3,2	6807
900	6	1,2,3,4,5,6	3,1,2,2,2,2	9834
750	6	1,2,3,4,5,6	2,1,2,2,2,2	9934
730	6	1,2,3,4,5,6	2,1,1,2,2,2	12102
710	6	1,2,3,4,5,6	2,1,1,1,2,2	14193

Table 13.5: GSM case study allocation and frequency assignment

minimum frequency (66 MHz, corresponding to a divisor of 3). As the deadline gets tighter, the optimizer prefers to employ a second processor and to progressively balance the load, instead of increasing task frequencies. This procedure is repeated every time a new processor has to be allocated to meet the timing constraints. Only under very tight deadlines, the optimizer leverages increased task frequencies to speed-up system performance. When the system is pushed to the limit, its configuration consists of 1 task for each processor, although they are not all running at the maximum frequency. In fact, the GSM pipeline turns out to be unbalanced, therefore it would be energy inefficient to run the shorter tasks at maximum speed, and would not even provide performance benefits. As a result, the optimizer determines the most energy-efficient configuration that provides the best performance. The problem becomes infeasible if more stringent deadlines than 710 ns are required.

JPEG decoder

In this section we will show that this optimizer behaviour is a function of the computation-communication ratio. We analyze a JPEG decoder case study. A

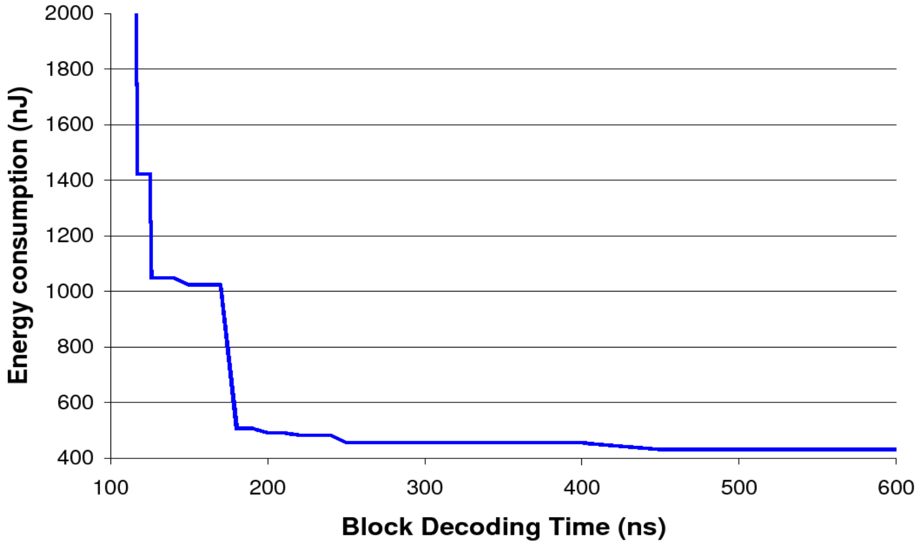


Figure 13.2: JPEG case study: Pareto-optimal frontier in the performance-energy design space

JPEG decoder is partitioned into 4 pipeline stages: Huffman DC decoding, Huffman AC decoding, inverse quantization, inverse DCT. Each stage processes an 8x8 block, amounting to an exchange of 1024 bit among pipeline stages. The accuracy of the energy estimation given by the optimizer was found to be 3.1% from functional simulation. In contrast to GSM, user requirements on a JPEG decoding usually consist of the minimization of the execution time and not of a deadline to be met. Therefore, two approaches to allocation and scheduling of a JPEG decoder task graph are feasible. On one hand, the designer could be primarily interested in reducing execution time at the cost of increased energy. On the other hand, the primary objective function could be the minimization of energy dissipation, whatever the decoding performance. This trade-off has been investigated with the optimizer and the Pareto-optimal frontier in the performance-energy space is illustrated in Figure 13.2. The constraint on the execution time on the x-axis is translated into a constraint on the block decoding time. The curve is not linear since there is a discrete number of voltage-frequency pairs, which makes the problem for the optimizer much more complex.

As we can observe, for a large range of deadlines, the optimizer is good at improving system performance without significantly changing processor energy dissipation. This is done by using one or two processors, changing the allocations and using high frequency dividers. Beyond 200 ns, the optimizer is forced to use low frequency dividers, thus causing the energy to skyrocket. Interestingly, the increase of task frequency is preferred to an increase of the number of processors, since the communication energy would involve even higher total energy consumption. This behaviour is different from the one seen for the GSM, since this time the computation-communication ratio is lower than for GSM due to a larger size of exchanged messages.

13.3.3 Sensitivity to initial mapping

To show the effectiveness of our approach, we have compared our optimal solutions (with proof of optimality) with those found by an heuristic approach. We have in fact devised a heuristic algorithm for the mapping on top of which we have computed a scheduling and a frequency assignment. The devised heuristic balances the workload on different processors. We have tested the optimal and heuristic solutions both on GSM and on JPEG demonstrators for decreasing values of deadlines.

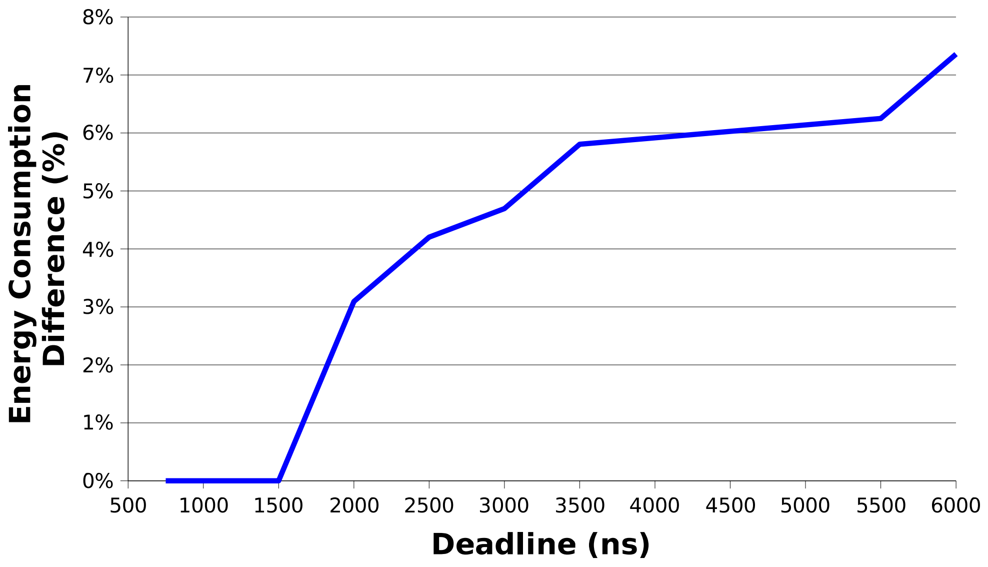


Figure 13.3: Energy consumption difference between different approaches on GSM

As shown in Fig.13.3, experiments on GSM show that for tight deadlines the heuristic approach provides an energy consumption equal to the optimal algorithm. This is because also the optimal allocator tends to spread tasks on different processors so as to meet deadline constraints. When the deadline is loose, instead, the optimal allocation provides solutions which save up to the 8% of energy w.r.t. the sub-optimal algorithm.

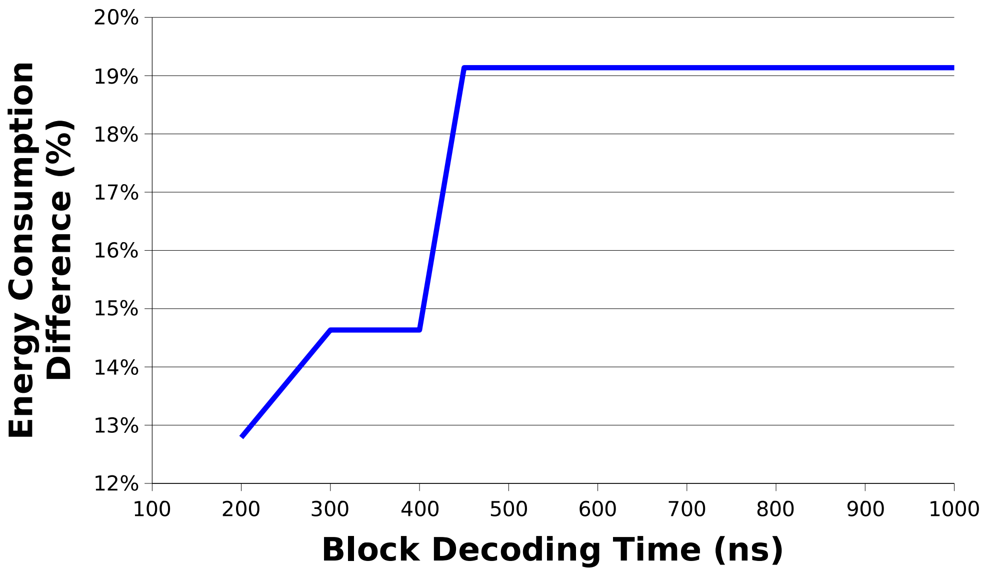


Figure 13.4: Energy consumption difference between different approaches on JPEG

The curve for the JPEG demonstrator, reported in Figure 13.4, has a similar trend, but with significantly different extreme values: in fact, even for tight deadlines the optimal solution saves 13% of the energy. This is due to the increased communication occurring when load balancing is applied. The optimal solution for loose deadlines saves up to the 19% which is a significant gap overall. In addition, for a very tight deadline (150ns) the heuristic approach fails to find even a feasible solution while the optimal solver finds the optimal solution and proves optimality.

Chapter 14

Conclusions and future works

In this dissertation we have given evidence, to support our thesis, that Constraint Programming and Integer Programming are effective paradigms to solve hard combinatorial problems and, for some classes of problems, the hybridization of the two solving techniques can lead to great advantages.

In this last Chapter we will draw some conclusions, summarizing in Section 14.1 the results achieved in this thesis, evidencing in Section 14.2 the lessons learnt, the strong points and the limitations of our approach, and concluding with future lines of research.

14.1 Contribution

In the previous Chapters we have analyzed two classes of problems with particular structures that suggest to develop solvers based on two techniques. Our choice is fallen on Constraint Programming (CP) and Integer Programming (IP), mainly for their efficiency in solving NP-Hard problems and for the fact that their characteristics are somehow orthogonal. In fact, a strong point of an approach is a weakness for the other, and viceversa.

The Part II of this dissertation has been devoted to the Bid Evaluation Problem (BEP), a NP-Hard combinatorial problem rising in the context of electronic commerce. In Chapter 5 we have described the problem, finding that the structure can be very different from instance to instance. What we believed could be effective to solve the BEP is to build several solving tools, based on IP, CP or both. In Chapter 6 we have described the CP and IP models for the BEP, and the implemented algorithms based on these models.

In Chapter 7, experimental results have shown the effectiveness of our approaches, being order of magnitude better with respect to MAGNET, a commercial tool based on IP, able to solve the BEP. By further analyzing the experimental results, we have found that the solvers behaviours are strongly influenced by the instance structure. Starting from these observation, in Chapter 8 we have introduced an algorithm portfolio to solve the BEP and an automatic algorithm selection tool, based on a Machine Learning approach, the Decision Trees, able to suggest the best algorithm on the basis of few structural parameters extracted from the constraint graph associated to each BEP instance. Experimental results have shown that we are able to predict the best algorithm in over the 90% of the cases analyzed, with a time saving of orders of magnitude w.r.t. a single solving strategy and over than 12% w.r.t. informed selection techniques based on the service-for-bid parameter, which has a negligible extraction time.

The Part III of this dissertation has been devoted to two problems, rising in the context of embedded systems design, namely the Allocation and Scheduling Problem (ASP) and the Dynamic Voltage Scaling Problem (DVSP) of coordinated tasks in a Multi-Processor System-on-Chip (MPSoC) platform. As for the BEP, in Chapter 9 we have analyzed in deep the problem structure and we have shown that, differently from the BEP case, here CP and IP must be integrated in a single solver, having recognized in the problem structure two distinct sub-problems, the allocation and the scheduling, with such characteristics that, taken separately, the former would be best solved by IP while the latter by CP.

In Chapters 10 and 12 we have introduced the models to solve respectively the ASP and the DVSP, and in particular the IP models to solve the allocation part and the CP models to schedule the allocations found. We have also exploited the Logic-Based Benders Decomposition technique to make the two solvers interacting, focussing our attention on the relaxations and Benders cuts used.

In Chapters 11 and 13 we have shown the experimental results obtained when solving respectively ASP and DVSP problem instances. Our experimental analysis was two-fold. First of all, our aim was to verify the efficiency of the hybrid approach proposed and, being these problems strongly related on real applications in the system design scenario, we also need to verify the accuracy and the actual executability of the solutions found.

We have shown that our hybrid approach is incomparably better than pure approaches based only on CP or IP on realistic instances, and we have shown the importance of smart relaxations and tight Benders cut in reducing the iterations between the two solvers.

To verify the accuracy of our tool, we have simulated our problem instances on a MPSoC platform simulator, measuring the difference of some important features, such as the activities durations, the application throughput, the power consumption, for real applications such as JPEG encoding, GSM coding/decoding, MIMO applications. We found that the relative difference always lies within the 5% for all the features measured, thus our tool is extremely accurate in modelling and solving real world applications.

14.2 Final considerations

14.2.1 Lessons learnt

As in nature the evolution of the species usually advances by maintaining the better examples of each species and by hybridizing different individuals of the same species to "keep the best" of each individual, here we have shown that a similar technique can be successfully exploited to solve NP-Hard combinatorial problems. Of course, it is well known that, for a large number of classes of problems there is no space for hybridization, being the problem structure very clear, so that a single approach exists and is particularly well suited for that kind of "structural skeleton". We believe that, being the real world rarely simple and clear, to model a real problem in a realistic way, it is often the case that the various facets of the problem can suggest to model and solve it in a hybrid way. We have chosen two examples in this dissertation, very different one another, but we believe, and the literature supports this claim, that an hybrid approach can be applied to a wide variety of problems.

When choosing the hybridization framework, a large amount of time must be spent in analyzing the problem characteristics. In fact, besides having shown that hybrid approaches are preferable w.r.t. pure ones for the problems considered, in this dissertation we have also given evidence that the way the different solving paradigms are hybridized strongly influences the solver performance.

We have learnt therefore that hybridization needs an accurate project phase where the way the different paradigms will interact must be deeply studied. In particular, considering the ASP and DVSP problems, we have seen that the sub-problems definition has a strong reflection on the overall performances. What would happen if some of the choices done in the allocation phase would have been postponed to the scheduler? We were often asked to answer this question by the AI community. We can see that, splitting a problem in a different way, we would have a scheduling dealing with a number of alternative resources: if, for example, in the ASP we allocate the memory during the scheduling phase, we would schedule reading/writing activities with a variable duration. The same happens in the DVSP if the frequency assignment would be postponed to the scheduling. On the other way round, deciding the successor of an activity in the allocation phase would affect the model size: as seen in Section 12.2.4, to model the successors in the allocation IP model, we need decision variables $Next_{t_1 t_2 f_1 f_2 p}$, taking value 1 if, on processor p, task t_1, running at frequency f_1, executes just before t_2, running at frequency f_2. These variables complicate the model too much.

We have also verified, in the context of the Logic-Based Benders Decomposition, the importance of the sub-problem relaxations and Benders cuts, and in particular we have learnt the importance of finding the best tradeoff between cut tightness and computation hardness to find it.

The need to find the best tradeoff is also evident for the BEP, when developing a portfolio and an automatic selection tool. The CC parameter is extremely accurate in predicting the best algorithm, but is too time-expensive in the 50% of the cases. On the other extreme, the S/B is costless but inaccurate on the 28% of the cases and, worst of all, inaccurate for those cases where predicting the wrong algorithm leads to the impossibility of finding the solution. As seen in Chapter 8, the ED and ND parameters are, in this case, an optimal tradeoff.

14.2.2 Limitations and future works

In this dissertation we have combined together only CP and IP. We believe this is not a limitation because our tools are extensible and flexible, being based on general integration frameworks, as Benders Decomposition and algorithm portfolios. We can extend our tools including other solving techniques or we can reuse them adapting to different problems presenting similar characteristics.

We can easily add other algorithms to the BEP portfolio. The selection strategy does not depend on the solver but on the instance, and it is therefore reasonable to believe that the same structural characteristics will remain a valid way to discern the best among an higher number of algorithms.

Considering the MPSoC-related problems, we believe that the Benders framework will remain a valid choice when changing the sub-problems solver. For example, we can schedule using an heuristic approach, so finding sub-optimal (or not provably optimal) solutions for the scheduling sub-problem, and we are still able to generate the cuts for the master problem.

We have seen a limit for the Benders Decomposition approach when the problem is loosely constrained, that is when the solution space is large and sparse. It is hard to find cuts such that the number of solutions removed justifies the time spent in generating them. One of our work in progress is aimed at improving the Benders Decomposition framework to overcome the limitations, analyzing the structure of the task graph to have an estimation of the number of iterations between master and sub-problem, thus to search for tighter cuts only when their presence in the model will considerably reduce the number of such iterations.

Another case where it is difficult to find the Benders Cuts is when the task graph has a high degree of parallelism or when considering conditional task graphs, where each edge outcoming from each node, thus each successor of each activity, is annotated with the probability for the activity to be executed. We believe that finding a methodology to derive smart cuts from conditional task graphs will allow us to embed our allocation and scheduling module in a code compiler for optimizing the code execution on parallel processors systems. In fact, a program can be seen as an application where each code instruction is an activity: code dependencies and jumps defines a task graph with conditional precedences and cycles. It is therefore extremely difficult to find cuts, valid for all the possible execution paths in the task graph, able to effectively reduce the solution space.

Bibliography

[1] *The MPARM Project Homepage.* http://www-micrel.deis.unibo.it/sitonew/research/mparm.html.

[2] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage power reduction of time-constraint systems. In *Proceedings of the Conference and Exposition in Design, Automation and Test in Europe (DATE2004)*, pages 518–523, Paris, France, February 2004.

[3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of travelling salesman problems. *Documenta Mathematica*, Extra Volume Proceedings ICM III (1998):645–656, August 1998.

[4] J. Axelsson. Architecture synthesis and partitioning of real-time synthesis: a comparison of 3 heuristic search strategies. In *Proceedings of the 5th International Workshop on Hardware/Software Codesign (CODES/CASHE97)*, pages 161–166, Braunschweig, Germany, March 1997.

[5] S. Bakshi and D. D. Gajski. A scheduling and pipelining algorithm for hardware/software systems. In *Proceedings of the 10th international symposium on System synthesis (ISSS97)*, pages 113–118, Antwerp, Belgium, September 1997.

[6] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling.* Springer, Berlin, Germany, 2001.

[7] J. Beck and L. Perron. Discrepancy-bounded depth first search. In *Proceedings of the 2nd international workshop on Integration of AI and OR Techniques in*

Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2000), pages 7–17, Paderborn, Germany, March 2000.

[8] J. C. Beck and E. C. Freuder. Simple rules for low-knowledge algorithm selection. In *Proceedings of the 1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, pages 50–64, Nice, France, May 2004.

[9] N. Beldiceanu and E. Contejean. Introducing global constraint in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, December 1994.

[10] A. Bender. MILP based task mapping for heterogeneous multiprocessor systems. In *Proceedings of the conference on European design automation (EURO-DAC96/EURO-VHDL96)*, pages 190–197, Geneva, Switzerland, September 1996.

[11] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, February 1962.

[12] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. MPARM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing Systems*, 41(2):169–182, September 2005.

[13] L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI05)*, pages 1517–1518, Edinburgh, Scotland, August 2005.

[14] L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP2005)*, pages 107–121, Sitges, Spain, September 2005.

[15] L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation, scheduling and voltage scaling on energy aware MPSoCs. In *3rd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2006)*, pages 44–58, Cork, Ireland, June 2006.

[16] L. Benini, D. Bertozzi, A. Guerri, M. Milano, and M. Ruggiero. A fast and accurate technique for mapping parallel applications on stream-oriented MPSoCs platforms with communication-awareness. *International Journal of Parallel Computing*, page to appear.

[17] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.

[18] R. Borndorfer. *Aspects of Set Packing, Partitioning, and Covering.* Shaker Verlag, Aachen, Germany, 1998.

[19] C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for the allocation of resources with complementarities. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 527–534, Stockholm, Sweden, August 1999.

[20] C. Boutilier and H. H. Hoos. Solving combinatorial auctions using stochastic local search. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI2000)*, pages 22–29, Austin, TX, USA, August 2000.

[21] H. Cambazard, A. M. Déplanche, P. E. Hladik, N. Jussien, and Y. Trinquet. Decomposition and learning for a hard real time task allocation problem. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pages 153–167, Toronto, Canada, September 2004.

[22] T. Carchrae and J. C. Beck. Applying machine learning to low knowledge control of optimization algorithms. *Computational Intelligence*, 21(4):373–387, November 2005.

[23] K. S. Chatha and R. Vemuri. Hardware-software partitioning and pipelined scheduling of transformative applications. 10(3):193–208, March 2002.

[24] J. Collins, R. Sundareswara, M. L. Gini, and B. Mobasher. Bid selection strategies for multi-agent contracting in the presence of scheduling constraints. In *Agent Mediated Electronic Commerce (IJCAI Workshop)*, pages 113–130, Stockholm, Sweden, August 1999.

[25] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 1999.

[26] S. Craw, N. Wiratunga, and R. Rowe. Case-based design for tablet formulation. In *4th European Workshop on Advances in Case-Based Reasoning (EWCBR-98)*, pages 358–369, Dublin, Ireland, September 1998.

[27] D. A. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, 1999.

[28] G. B. Dantzig. Programming in a linear structure. USAF, Washington D.C., 1948.

[29] G. B. Dantzig. *Linear programming and extensions.* Princeton University Press, 1963.

[30] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programming. *Econometrica*, 29(4):767–778, October 1961.

[31] G. De Micheli. *Synthesis and optimization of digital circuits.* McGraw Hill, 1994.

[32] A. G. Doig and A. H. Land. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, January 1960.

[33] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Design Automation for Embedded Systems*, 2(1):5–32, January 1997.

[34] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, and P. Pop. Scheduling of conditional process graphs for the synthesis of embedded systems. pages 132–139, Paris, France, February 1998.

[35] A. Eremin and M. Wallace. Hybrid Benders decomposition algorithms in constraint logic programming. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*, pages 1–15, Paphos, Cyprus, November 2001.

[36] W. Ertel. Performance of competitive OR-parallelism. In *Proceedings of the ICLP Workshop on Parallel Execution of Logic Programs*, pages 132–145, Paris, France, June 1991.

[37] E. Fink. How to solve it automatically: Selection among problem solving methods. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS98)*, pages 128–136, Pittsburgh, PA, USA, June 1998.

[38] F. Focacci, A. Lodi, and M. Milano. Solving tsp with time windows with constraints. In *Proceedings of the 16th International Conference on Logic Programming (ICLP99)*, pages 515–529, Las Cruces, NM, USA, December 1999.

[39] F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the tsp-tw. *INFORMS Journal on Computing*, 14(4):403–417, Fall 2002.

[40] G. Fohler and K. Ramamritham. Static scheduling of pipelined periodic tasks in distributed real-time systems. In *Proceedings of the 9th EUROMICRO Workshop on Real-Time Systems (EUROMICRO-RTS97)*, pages 128–135, Toledo, Spain, June 1997.

[41] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: optimal and approximate approaches. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 548–553, Stockholm, Sweden, August 1999.

[42] M. Gagliolo and J. Schmidhuber. Dynamic algorithm portfolios. In *9th International Symposium on Artificial Intelligence and Mathematics (AIMATH-2006)*, Fort Lauderdale, FL, USA, January 2006. http://anytime.cs.umass.edu/aimath06/proceedings/P37.pdf.

[43] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

[44] C. Gebruers and A. Guerri. Machine learning for portfolio selection using structure at the instance level. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, page 794, Toronto, Canada, September 2004.

[45] C. Gebruers, A. Guerri, B. Hnich, and M. Milano. Making choices using structure at the instance level within a case based reasoning framework. In *Proceedings of the 1st International Conference on Integration of AI and OR*

Techniques in Constraint Programming for Combinatorial Optimization Problems, (CPAIOR 2004), pages 380–386, Nice, France, May 2004.

[46] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, October 1972.

[47] C. Gini. Measurement of inequality and incomes. *The Economic Journal*, 31:124–126, 1921.

[48] M. L. Ginsberg and W. D. Harvey. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 607–613, Montrèal, Quèbec, Canada, August 1995.

[49] C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI97)*, pages 190–197, Providence, RI, USA, August 1997.

[50] I. E. Grossmann and V. Jain. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4):258–276, Fall 2001.

[51] F. Gruian and K. Kuchcinski. LEneS: Task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of 2001 on Asia and South Pacific Design Automation Conference (ASP-DAC2001)*, pages 449–455, Yokohama, Japan, January 2001.

[52] A. Guerri and M. Milano. CP-IP techniques for the bid evaluation in combinatorial auctions. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP2003)*, pages 863–867, Kinsale, Ireland, September 2003.

[53] A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the 16th European Conference on Artificial Intelligence, (ECAI 2004)*, pages 475–479, Valencia, Spain, August 2004.

[54] H. Guo and W. H. Hsu. A machine learning approach to algorithm selection for NP-hard optimization problems: A case study on the MPE problem. *Annals of Operations Research, Special Issue on Stochastic Search Algorithm*, 2006. to appear.

[55] J. N. Hooker. A hybrid method for planning and scheduling. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pages 305–316, Toronto, Canada, September 2004.

[56] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, April 2003.

[57] J. N. Hooker and H. Yan. *Principles and practice of constraint programming : the Newport papers*, chapter "Logic circuit verification by Benders decomposition". V. Saraswat and P. Van Hentenryck (eds), MIT Press, 1995.

[58] E. N. Houstis, A. C. Catlin, J. R. Rice, V S. Verykios, N. Ramakrishnan, and C. E. Houstis. PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, 26(2):227–253, June 2000.

[59] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceedings of the Conference and Exposition in Design, Automation and Test in Europe (DATE2004)*, pages 234–239, Paris, France, February 2004.

[60] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, January 1997.

[61] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP2006)*, pages 213–228, Nantes, France, September 2006.

[62] R. Ihaka and R. Gentlemen. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, September 1996.

[63] ILOG SA. *ILOG CPLEX 8.1, Reference Manual.* 2002.

[64] ILOG SA. *ILOG Scheduler 5.3, Reference Manual.* 2002.

[65] ILOG SA. *ILOG Solver 5.3, Reference Manual.* 2002.

[66] K. Imai, G. King, and O. Lau. Toward a common framework for statistical analysis and development. Technical report, October 2006. http://gking.harvard.edu/files/z.pdf.

[67] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 international symposium on Low power electronics and design (ISPLED98)*, pages 197–202, Monterey, CA, USA, August 1998.

[68] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *In Proceedings of the 42nd Design and Automation Conference (DAC2005)*, pages 111–116, San Diego, CA, USA, June 2005.

[69] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, November 1984.

[70] R. M. Karp. *Complexity of Computer Computation*, chapter "Reducibility among combinatorial problems", pages 85–103. R. Miller and J. Thatcher (eds), Plenum Press, 1972.

[71] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[72] C. Kim and K. Roy. Dynamic VTH scaling scheme for active leakage power reduction. In *Proceedings of the Conference and Exposition in Design, Automation and Test in Europe (DATE2002)*, pages 163–167, Paris, France, March 2002.

[73] D. E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29:121–139, January 1975.

[74] S. Kodase, S. Wang, Z. Gu, and K. Shin. Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS2003)*, pages 181–188, Toronto, Canada, May 2003.

[75] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, San Mateo, CA, USA, 1993.

[76] R. E. Korf. Improved limited discrepancy search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI96)*, pages 288–691, Portland, OR, USA, August 1996.

[77] K. Kuchcinski. Embedded system synthesis by timing constraint solving. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(5):537–551, May 1994.

[78] K. Kuchcinski and R. Szymanek. A constructive algorithm for memory-aware task assignment and scheduling. In *Procs of the 9th International Symposium on Hardware/Software Codesign (CODES2001)*, pages 147–152, Copenhagen, Denmark, April 2001.

[79] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML2000)*, pages 511–518, Standford, CA, USA, June 2000.

[80] C. Lee, M. Potkonjak, and W. Wolf. System-level synthesis of application-specific systems using A* search and generalized force-directed heuristics. pages 2–7, San Diego, CA, USA, November 1996.

[81] M. Lenz, H. D. Burkhard, P. Pirk, E. Auriol, and M. Manago. CBR for diagnosis and decision support. *AI Communications*, 9(3):138–146, September 1996.

[82] L-F. Leung, C-Y. Tsui, and W-H. Ki. Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment. In *Proceedings of 2004 on Asia and South Pacific Design Automation Conference (ASP-DAC2004)*, pages 647–652, Yokohama, Japan, January 2004.

[83] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP2003)*, pages 899–903, Kinsale, Ireland, September 2003.

[84] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1542–1543, Acapulco, Mexico, August 2003.

[85] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the 8th International Conference Principles and Practice of Constraint Programming (CP2002)*, pages 556–572, Ithaca, NY, USA, September 2002.

[86] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. Technical report, 2005. http://web.tepper.cmu.edu/jnh/planning.pdf.

[87] K. Leyton-Brown, E. Nudelman, and Y. Shoham. *Combinatorial Auctions*, chapter 19, “Empirical Hardness Models for Combinatorial Auctions”, pages 479–504. P. Cramton and Y. Shoham and R Steinberg (eds.), MIT Press, 2006.

[88] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce (EC00)*, pages 66–76, Minneapolis, MN, USA, October 2000.

[89] Y. Li and W. H. Wolf. Hardware/software co-synthesis with memory hierarchies. 18(10):1405–1417, October 1999.

[90] L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI98-IAAI98)*, pages 353–358, Madison, WI, USA, July 1998.

[91] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a MPSoC environment. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE2004)*, pages 752–757, Paris, France, February 2004.

[92] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, September 1993.

[93] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design (ICCAD2002)*, pages 721–725, San Jose, CA, USA, November 2002.

[94] S. Meftali, F. Gharsalli, A. A. Jerraya, and F. Rousseau. An optimal memory allocation for application-specific multiprocessor system-on-chip. In *Proceedings of the 14th international symposium on Systems synthesis (ISSS01)*, pages 19–24, Montrèal, Quèbec, Canada, September 2001.

[95] R. Menke and R. Dechter. An implementation of the combinatorial auction problem in ECL$^{\text{i}}$PS$^{\text{e}}$. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI2000)*, page 1084, Austin, TX, USA, August 2000.

[96] M. Milano. *Constraint and Integer Programming: toward a unified methodology.* Kluwer Academic Publisher, 2004.

[97] N. Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce (EC00)*, pages 1–12, Minneapolis, MN, USA, October 2000.

[98] D. Novo, W. Moffat, V. Derudder, and B. Bougard. Mapping a multiple antenna SDM-OFDM receiver on the ADRES coarse-grained reconfigurable processor. In *IEEE Workshop on Signal Processing Systems Design and Implementation 2005*, pages 473–478, Athens, Greece, November 2005.

[99] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP2004)*, pages 438–452, Toronto, Canada, September 2004.

[100] Optimal tour of Sweden. 2004. http://www.tsp.gatech.edu/sweden/index.html.

[101] R. Steinberg P. Cramton, Y. Shoham. *Combinatorial Auctions.* The MIT Press, Cambridge, MA, USA, 2006.

[102] P. Palazzari, L. Baldini, and M. Coli. Synthesis of pipelined systems for the contemporaneous execution of periodic and aperiodic tasks with hard real-time constraints. In *18th International Parallel and Distributed Processing Symposium (IPDPS04)*, pages 121–128, Santa Fe, NM, USA, April 2004.

[103] G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5(3):255–279, October 1999.

[104] F. Poletti, A. Poggiali, and P. Marchal. Flexible hardware/software support for message passing on a distributed shared memory architecture. In *2005 Design, Automation and Test in Europe Conference and Exposition DATE2005*, pages 736–741, Munich, Germany, March 2005.

[105] S. Prakash and A. Parker. SOS: Synthesis of application-specific heterogeneous multiprocessor systems. *Journal of Parallel and Distributed Computing*, 16(4):338–351, December 1992.

[106] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, USA, 1993.

[107] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A combinatorial mechanism for airport time slot allocation. *Bell Journal of Economics*, 13:402–417, Autumn 1982.

[108] J. C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI94)*, volume 1, pages 362–367, Seattle, WA, USA, August 1994.

[109] J. C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI96)*, volume 1, pages 209–215, Portland, OR, USA, August 1996.

[110] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, September 1976.

[111] G. Rockafellar. *Convex Analysis.* Princeton University Press, 1970.

[112] M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini. Application-specific power-aware workload allocation for voltage scalable MPSoC platforms. In

23rd International Conference on Computer Design (ICCD2005), pages 87–93, San Jose, CA, USA, October 2005.

[113] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *Proceedings of the conference on Design, automation and test in Europe (DATE06)*, pages 3–8, Munich, Germany, March 2006.

[114] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 542–547, Stockholm, Sweden, August 1999.

[115] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, February 2002.

[116] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

[117] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Iterative schedule optimization for voltage scalable distributed embedded systems. *ACM Transactions on Embedded Computing Systems*, 3(1):182–217, February 2004.

[118] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming (CP98)*, pages 417–431, Pisa, Italy, October 1998.

[119] H. Theil. A multinomial extension of the linear Logit model. *International Economics Review*, 10(3):251–259, October 1969.

[120] E. S. Thorsteinsson. A hybrid framework integrating mixed integer programming and constraint programming. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP2001)*, pages 16–30, Paphos, Cyprus, November 2001.

[121] S. De Vries and R. V. Vohra. Combinatorial auctions: a survey. *INFORMS Journal on Computing*, 15(3):284–309, Summer 2003.

[122] T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 1388–1395, Nagoya, Japan, August 1997.

[123] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005. http://www.cs.waikato.ac.nz/ml/weka/.

[124] W. Wolf. The future of multiprocessor systems-on-chips. In *Proceedings of the 41st Design and Automation Conference (DAC2004)*, pages 681–685, San Diego, CA, USA, June 2004.

[125] F. Xie, M. Martonosi, and S. Malik. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Proceedings of the 2005 international symposium on Low power electronics and design (ISPLED05)*, pages 287–292, San Diego, CA, USA, August 2005.

[126] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS95)*, pages 374–382, Milwaukee, WI, USA, October 1995.

Index