Integrated Methods and Systems for Optimization and Decision Support

Stathis Plitsos

Department of Management Science and Technology

Athens University of Economics and Business

A thesis submitted for the degree of

Doctor of Philosophy

September 2017

This thesis is dedicated to
my parents, Giorgos and Margarita,
for supporting me in many different ways
during the hard period of my research.

Acknowledgements

I would like to personally thank my supervisor, Dr. Ioannis Mourtos, for his patience, understanding and chances he offered me during my research. Additionally, I would like to thank Prof. Dimitrios Magos and Prof. Christos Tarantillis for their support on this work. I could not let aside Dr. Pavlos Eirinakis who helped me in many different ways during the period of my research.

Furthermore, this research has been funded by

- a European research project, namely “ARTISAN” (GA no. 287993);
- the National Research Fund under the umbrella of Archimedes III sub-project 28, namely “Assignment problem and all-different Systems” (MIS: 379389);
- the National Research Fund under the umbrella of Thales 85418, namely “From Graph Theory to Matroids: Algorithmic Issues and Applications” (MIS: 379437).

Finally, I feel honoured working with such brilliant colleagues and professors in the E-Business Research Centre (ELTRUN) the past 5 years. In my opinion, ELTRUN remains one of the top places for a PhD candidate in Greece, given the research projects and ideas worked and put in action along with the academic results achieved.

Abstract

This thesis falls within the scope of Combinatorial Optimization and Decision Support Systems (DSS). Its purpose is to introduce algorithmic components for different optimization problems along with a DSS for each problem as further result. Motivated by the so-called *integrated* methods for optimization, we study three different optimization problems, and present new algorithms that combine the complementary strengths of the three major types of optimization methods, i.e., Mathematical Programming, Constraint Programming and Heuristics.

The first problem we focus on, is the multi-index assignment. In this part of work we propose several components that can be employed across different types of assignment, i.e, a constraint propagation mechanism, a tabu-search meta-heuristic, a new variant of the *Feasibility Pump* heuristic that employs cutting planes, along with a new Branch & Cut method for the problem at hand. The computational experimentation shows that indeed these components when employed together reduce the time to optimality or the integrality gap for large instances where a competitive commercial solver runs out of memory. An important aspect of this approach is its versatility, for example in terms of including a subset of the selected components or an alternative FP variant as a primal heuristic. Furthermore, the existence of these components in terms of code paves the way towards the development of a DSS for the problem at hand, which can facilitate several types of use, given its general-purpose design and the various applications of the multi-index assignment problem.

The second problem is the energy-aware production scheduling. Here, we present an energy-aware production scheduling DSS as designed, implemented and evaluated in a real context. In short, this work contributes to decision support for energy-efficient manufacturing by a metaheuristic algorithm that hierarchically optimizes flexible job-shop scheduling problems, a set of data requirements, the integrated deployment of this DSS as

a web-service and the evaluation of the DSS in real settings. The adopted scheduling framework incorporates various operational issues, while the data entities accompanying it meet generic energy-related requirements, as obtained from the literature and the textile industry. Apart from examining theoretical aspects regarding the design of energy-aware DSS, this work presents the significant tangible benefits obtained from the use of such systems within the textile manufacturing industry. Hence, the applicability of the proposed DSS, as deployed in two significantly different users and production environments, is shown to be both feasible and effective.

Last, we focus on the the binary multi-dimensional knapsack problem. Motivated by our research on the multi-index assignment problem and the new variant of the Feasibility pump heuristic that has been designed and tested, we broaden our focus on the multi-dimensional knapsack problem, where its structure is general enough to encompass all binary optimization problems. Here, we describe a new primal-dual method for this problem, which is a strongly NP-hard combinatorial optimization problem with many applications. Current exact approaches and commercial solvers run into difficulties even for a small-to-medium number of constraints and variables. The proposed primal-dual method employes the linear relaxation of the problem at hand, enhanced by global lifted cover inequalities to improve the upper bound and a new version of the Feasibility Pump heuristic that uses this family of inequalities in the pumping procedure to obtain better and feasible lower bounds. Since this is only preliminary work, this new variant of feasibility-pump is tested on literature instances, for a good portion of which there are still no optimal solutions available. The results of the heuristic are interesting enough to trigger further research and development for the proposed primal-dual method.

The contribution of this effort, given that we focus on two such large research areas, is multi-fold. Intuitively, an optimization algorithm on its own is not a DSS, while a DSS without an efficient algorithm cannot support efficiently decision making. The examination of different optimization problems in terms of structure and applications, has motivated the generation of quite diverse integrated optimization methods, while for the two of the problems two DSS have been developed and tested, with one of these in a real and demanding context. Given that there are various choices of algorithmic components towards optimization, this thesis could shed some light on the design of efficient optimization algorithms given the structure and the applications of the problem, while also demonstrating the use and study of such algorithms not only from a computational or analytical perspective, but from a DSS viewpoint.

Contents

List of Figures

List of Algorithms

Chapter 1

Motivation and outline

This thesis falls within the scope of Combinatorial Optimization (also referred to as Discrete Optimization) and Decision Support Systems (DSS). The purpose of this effort is to introduce algorithmic components for different optimization problems, not only from a Combinatorial Optimization point of view, i.e., mathematical analysis and algorithmic results, but also from a DSS perspective. This means that, apart from the design and implementation of efficient optimization methods, we focus also on the requirements generated by the application of the corresponding optimization problems and the design, implementation and evaluation of DSS that address these requirements by incorporating these optimization methods.

1.1 Background

Over the past decade, there is a broad interest and motivation for Integrated Optimization Methods [55]. Integration refers to three major types of optimization methods, namely, Mathematical Programming (MP), Constraint Programming (CP) and Heuristic methods. That is, instead of using methods of a single optimization type, one could exploit their complementary strengths by combining them in a meaningful and effective manner. Each optimization type offers different methods and advantages that can be selected and integrated appropriately considering the structure and the difficulties of the problem at hand. For example, Mathematical Programming offers the so-called linear relaxation, plus problem-specific or general-purpose polyhedral analysis, which reveals valid inequalities for the derivation of cutting planes. Constraint Programming offers inference techniques and modelling flexibility, i.e., a higher level mathematical formulation for the problem as opposed to Mathematical Programming where the problem formulation can become quite challenging. Heuristic methods and algorithms offer a clever search strategy of the solution space. Although these algorithms are highly problem-specific, they provide very good results.

Still, optimization calls for a multifaceted approach. Over the past 15 years, research has shown that there is much to be gained by exploiting the complementary strengths of different approaches to optimization instead of using each one individually. The result of such combinations of optimization methods is often encompassed under the recent domain of *integrated methods for optimization* [55]. There is a growing literature arguing in favour of several combinations of the methods presented above, displaying better computational performance compared to the use of a single optimization method. Nevertheless, a rising problem seems to be that if there are many solution methods, there are even more ways to combine them. Hence, there is a field of research on the 'appropriate' combination of solution methods.

Apparently, such optimization methods, given an application context modelled after the optimization problem they tackle, can facilitate decision making whether it is a minimization or a maximization of an objective. Decision support systems showed up early in the academic literature [77]. The inclusion of any algorithm, not necessarily an optimization one, in a system with appropriate architectural design, generates a Decision Support System (DSS) for the problem or decision at hand [101], thus enhancing decision making. In general, a DSS can be part of an organization emphasizing on flexibility, adaptability, quick response and support for the personal decision making styles of individual users, while it is user initiated and controlled [101]. Intuitively, an optimization algorithm on its own is not a DSS, while a DSS without an efficient algorithm cannot support efficiently decision making.

Since this thesis combines two major in size research areas, and considering that it discusses different optimization problems with different DSS as further results, the structure and discussion of each topic is modular. Rather than introducing all optimization problems and DSS with the research background at once, this thesis presents and discusses thoroughly each topic separately. That is, each optimization problem and DSS is introduced with the associated literature review, algorithmic components and experimental analysis. This helps the reader to navigate through this work without loss of focus, while keeping each chapter more self-contained and concise.

1.2 Structure of the thesis

Let us now provide an outline of this thesis; Chapter 2 introduces the multi-index assignment problem. This includes a description of the problem, the motivation deriving from the various applications of the axial and planar families of this problem and an Integer Programming mathematical model along with a Constraint Programming model using the *all-different* system.

Chapter 3 describes in detail a general-purpose Integer Programming heuristic, namely *Feasibility Pump* (FP), which is extensively studied and used as shown in the current literature. We focus on this heuristic because its general-purpose nature can tackle the structural differences of the two families of multi-index assignments, namely, axial and planar ones. Additionally, Chapter 3 presents a new variant of this heuristic that employs problem-specific cutting planes for multi-index assignment, followed by computational experimentation of this new FP variant along with the existing ones.

Chapter 4 presents an integrated method, i.e., an exact algorithm for multi-index assignment that incorporates different optimization components. These are: problem-specific cutting planes, as described in the literature, the aforementioned new FP variant, a tabu-search meta-heuristic and a mechanism that performs constraint propagation for the problem at hand. Note here, that these components remain functional for all families of assignment problems, while as discussed, the versatility of combinations of these components towards new integrated methods is quite interesting.

Chapter 5 presents a DSS for multi-index assignment that includes some of the aforementioned components. This DSS is thoroughly discussed, starting from the elicitation of user requirements, to the implementation of the system and the demonstration of the workflow, in terms of system screens. Let us highlight that integrated solvers coupled by such a DSS are not reported in the literature. Additionally, we consider such interfaces an important step for the adoption of such methods in practical situations, while the versatility of integrated methods yields a breadth of parameterization for which such a DSS could be of help. Therefore the work presented here could be of both practical and academic use beyond the scope of the optimization problem underlying it.

In Chapter 6 we shift our focus to a different optimization problem, for which this thesis contributes to DSS design and not to the optimization method employed. This is the energy-aware production scheduling problem, for which a DSS is described, i.e., user requirements, algorithmic components, data requirements and model followed by the system architecture. The proposed DSS has been used by two industrial users in the textile manufacturing sector under the framework of a European research project, namely ARTISAN [1]. Results of this research include the evaluation of the proposed DSS in terms of use and energy savings for the two industrial users.

Chapter 7 follows a different track by focusing on a third optimization problem for which we propose an optimization method but not a DSS. The problem is the multi-dimensional knapsack which is generic enough to include all binary optimization problems. Hence, in this chapter we focus on the design and development of a primal-dual algorithm that uses a family of general-purpose cutting planes appropriately chosen for this problem, along with the aforementioned FP variant that employs these cuts so as to obtain better feasible solutions. This last chapter is rather exploratory and attempts to transfer the optimization ideas developed in Chapters 3 and 4 to a far more generic class of problems. Therefore it presents only preliminary findings and mostly shapes the path for future research.

We complete our exposition with a short concluding chapter.

1.3 Contribution of the thesis

Considering that we examine three different optimization problems and propose two different DSS, the contribution of this work is multi-fold. Regarding the multi-index assignment problem, in this thesis we propose several components that can be employed across different types of assignment, i.e, a constraint propagation mechanism, a tabu-search meta-heuristic, a new variant of the *Feasibility Pump* heuristic that employs cutting planes, along with a new Branch & Cut method for the problem at hand. The computational experimentation shows that indeed these components when employed together reduce the time to optimality or the integrality gap for large instances where a competitive commercial solver runs out of memory. An important aspect of this approach is its versatility, for example in terms of including a subset of the selected components or an alternative FP variant as a primal heuristic. Furthermore, the existence of these components in terms of code paves the way towards the development of a DSS for the problem at hand, which can facilitate several types of use, given its general-purpose design and the various applications of the multi-dimensional assignment problem.

Regarding the energy-aware production scheduling problem, we present an energy-aware production scheduling DSS as designed, implemented and evaluated in a real context. In short, this work contributes to decision support for energy-efficient manufacturing by a metaheuristic algorithm that hierarchically optimizes flexible job-shop scheduling problems, a set of data requirements, the integrated deployment of this DSS as a web-service and the evaluation of the DSS in real settings. The adopted scheduling framework incorporates various operational issues, while the data entities accompanying it meet generic energy-related requirements, as obtained from the literature and the textile industry. Apart from examining theoretical aspects regarding the design of energy-aware DSS, this work presents the significant tangible benefits obtained from the use of such systems within the textile manufacturing industry. Hence, the applicability of the proposed DSS, as deployed in two significantly different users and production environments, is shown to be both feasible and effective.

Regarding the binary multi-dimensional knapsack problem, we describe a new primal-dual method for this problem, which is a well known (and strongly NP-hard) combinatorial optimization problem with many applications. Current exact approaches and commercial solvers run into difficulties even for a small-to-medium number of constraints and variables. The proposed primal-dual method employs the linear relaxation of the problem at hand, enhanced by global lifted cover inequalities to improve the upper bound and a new version of the Feasibility Pump heuristic that uses this family of inequalities in the pumping procedure to obtain better and feasible lower bounds. Since this is only preliminary work, this new variant of feasibility-pump is tested on literature instances, for a good portion of which there are still no optimal solutions available. The results of the heuristic are interesting enough to trigger further research and development for the proposed primal-dual method.

Overall, the examination of different optimization problems in terms of structure and applications, has motivated the generation of quite diverse integrated optimization methods, while for the two of the problems two DSS have been developed and tested, with one of these in a real and demanding context. Given that there are various choices of algorithmic components towards optimization, this thesis could shed some light on the design of efficient optimization algorithms given the structure and the applications of the problem, while also demonstrating the use and study of such algorithms not only from a computational or analytical perspective, but from a DSS viewpoint.

Chapter 2

The multi-index assignment problem

This section describes the multi-index assignment problem, i.e., a combinatorial problem that is well-studied across the literature. Indeed, several optimization problems include an assignment structure, hence the problem at hand encompasses numerous applications. In this section, we formally define the multi-index assignment problem, i.e., we provide its mathematical formulation and also discuss its application. Furthermore, since different approaches are followed towards its study, we present an additional Constraint Programming model using the *all-different* system.

The remainder of this chapter goes as follows; in Section 2.1 we provide the motivation and describe the mathematical model. Section 2.2 presents the research background of the multi-index assignment, i.e., approximation studies, exact algorithms, polyhedral analysis and heuristics that have showed up in the literature. Finally, Section 2.3 presents the Constraint Programming model for the problem at hand using the *all-different* system.

2.1 Problem definition and motivation

Several optimization problems include an assignment, i.e., two disjoint sets that must comply to an one-to-one relation [75]. Formally, the 2-index assignment problem is defined on two sets I and J, where $|I| = |J| = n$, plus a weight per pair $(i, j) \in I \times J$ and asks for a minimum-weight collection of n such pairs with the property that each element of $I \cup J$ appears in exactly one of them.

In a similar manner, the 3-index assignment problem [87] considers a third set K also of cardinality n plus a weight per triple $(i, j, k) \in I \times J \times K$ and asks for a collection of n such triples with the property that each element of $I \cup J \cup K$ appears in exactly one triple. This problem is called the *axial* 3-index problem because there is a second assignment problem defined on 3 sets, namely the *planar* 3-index problem [44]. The latter differs by asking for a collection of n^2 triples with the property that each pair of elements in $(I \times J) \cup (I \times K) \cup (J \times K)$ appears in exactly one triple (thus, each element in $I \cup J \cup K$ appears in n triples). Both problems can easily be modelled via (linear) Integer Programming (IP), using a binary variable per triple, as presented in Table 2.1.

Table 2.1: Integer Programming models for 3-index assignment problems

Axial assignment	Planar assignment
$\sum_{i\in I}\sum_{j\in J} x_{ijl} = 1,\ l \in L,$	$\sum_{i\in I} x_{ijl} = 1,\ j \in J,\ l \in L,$
$\sum_{i\in I}\sum_{l\in L} x_{ijl} = 1,\ j \in J,$	$\sum_{j\in J} x_{ijl} = 1,\ i \in I,\ l \in L,$
$\sum_{j\in J}\sum_{l\in L} x_{ijl} = 1,\ i \in I,$	$\sum_{l\in L} x_{ijl} = 1,\ i \in I,\ j \in J,$
$x_{ijl} \in \{0,1\},\ i \in I,\ j \in J,\ l \in L.$	$x_{ijl} \in \{0,1\},\ i \in I,\ j \in J,\ l \in L.$

In fact, axial assignment is originally defined on k disjoint n-sets [87], while planar assignment is an alternative representation of a long-standing combinatorial structure called *mutually orthogonal Latin squares* (MOLS) [36]. A Latin square of order n is an $n \times n$ matrix in which values $1, 2, \ldots, n$ appear once in each row and column; hence if sets I, J and K index the rows, columns and values of an $n \times n$ matrix, it becomes evident that each 3-index planar assignment is a Latin square of order n and vice-versa. For the definition of MOLS, see [65]. IP models can be defined also for k-index axial and planar assignment problems, using a binary variable for each of the n^k k-tuples.

A more general IP model that also encompasses other assignment types has been introduced by Appa et al. [6], by defining the (k, s) assignment problem, denoted as $(k, s)AP_n$. The parameter s determines the assignment type, with $s = 1$ and $s = 2$ yielding axial and planar assignment respectively. Formally, the $(k, s)AP_n$ assumes k disjoint n-sets and asks for a collection of n^s k-tuples with the property that each s-tuple of elements appears in exactly one k-tuple. Hence, Table 2.1 presents the IP models for $(3, 1)AP_n$ and $(3, 2)AP_n$. The mathematical formulation and complete definition lies in [8]; Here, is provided a short description and the mathematical model as obtained by that study.

The problem is formulated as follows:

$$min \sum \{w_{m^K} \cdot x_{m^K} : m^K \in M^K\},$$

$$\text{s.t. } \sum \{x_{m^K} : m^{K\setminus S}\} = 1 \text{ for every } m^S \in M^S, S \in Q_{k,s},$$

$x_{m^K} \in \{0,1\}^{n^k}$,for every $m^K \in M^K$

where,

K a set of indices, $K = \{1, ..., k\}$,

$S \subseteq K$, subset of indices,

$Q_{k,s}$ the collection of all distinct S, i.e. $Qk, s = \{S \subseteq K : |S| = s\}$, with $|Q_{k,s}| = \binom{k}{s}$,

k disjoint n-sets $M_1, M_2, ..., M_k$ and let $m^i \in M_i$, for $i \in K$,

$S = \{i_1, i_2, ...i_s\}$ such that $i_1 < i_2 < ... < i_s$,

$M^S = M_{i_1} \times M_{i_2} \times ... \times M_{i_s}$ and $m^S \in M^S$,

x_{m^K} binary variables and the mapping $w : M^K \longrightarrow \mathbb{R}$

There are exactly s fixed indices in each constraint. $M^{K \backslash S}$ is the set of indices appearing in the sum, whereas M^S is the set of indices common to all variables in an equality constraint. The (0,1) matrix of the constraints $A_n^{(k,s)}$ has n^k columns and $\binom{k}{s} \cdot n^s$ rows, i.e. n^s constraints for each of the $\binom{k}{s}$ distinct $S \in Q_{k,s}$ and each constraint includes n^{k-s} variables.

Apart from their nice combinatorial structure, assignment problems enjoy a broad range of applications, thus requiring an effective optimization approach. For example, axial assignment applies to data-association problems [91], to the classification and pairing of human chromosomes [17] and to wafer-to-wafer yield optimization in 3D electronic circuit printing [103]. Planar assignment shares the diverse applications of Latin squares [65] from the statistical design of experiments [54] to error correcting codes [86]. The size of these real problems varies significantly but normally assumes $k \geq 3$ and $n \geq 50$.

Despite the extensive literature on assignment problems, discussed in the next section, there is a gap in exact optimization methods that tackle large-scale instances. In addition, although assignment problems for different values of k and s share a common structure, most existing computational methods focus on a specific s and frequently on a specific k. This appears reasonable given that the $(k, s)AP_n$ becomes $\mathcal{NP}$-complete already for $k = 3$: for $s = 1$ this follows from an early result on 3-dimensional matching [61], while for $s = 2$ it is shown by Frieze [44]. Moreover, the fast-growing size of the $(k, s)AP_n$'s Linear Programming (LP) relaxation made it memory-wise intractable even for small n, thus discouraging the design of appropriate 'Branch & Cut' (B&C) methods. This is indicated by the fact that existing exact methods normally rely on 'Branch & Bound' (B&B) and subgradient algorithms for solving a Langrangean relaxation, as in [13] or [71], with limited (if any) use of cutting planes, e.g., at the top node of the B&B tree by [94] or [72]. Notably, all existing approaches focus on small instances (e.g., for $s = 1, k = 3$ and $n \leq 30$), although related applications ask for much larger ones [33] and B&C approaches can cope with large instances on other optimization problems, e.g., Lysgaard et al. [69].

Several non-exact methods have also been proposed for the 3-index axial problem in the form of either approximation methods for special (or even polynomially solvable) cases as in [26] or meta-heuristics (e.g., [59]) that indeed deal with large instances. A metaheuristic for the 3-index planar problem has been presented by Magos [70]. Let us note here that these approaches focus on a specific type of assignment, thus it is doubtful whether they could become applicable for different values of k and s.

The remainder of this chapter goes as follows; Section 2.2 presents the research background of the Multi-index assignment, i.e., approximation studies, exact algorithms, polyhedral analysis and heuristics that showed up in the literature the past years. Finally, Section 2.3 presents the Constraint Programming model for the problem at hand using the *all-different* system.

2.2 Research background

As already mentioned, Pierskalla [87] introduces the $(k,1)AP_n$ (i.e., the axial k-index assignment problem) and the first B&B method, which is a primal-dual scheme for upper and lower bound acquisition, with an analogous scheme appearing in [52] for $k = 3$; the branching strategy in both cases fixes single variables to zero or one. Also for the $(3,1)AP_n$, Balas and Saltzman [13] present an elegant B&B method that uses dual heuristics to obtain lower bounds, based on a Lagrangian relaxation tightened with facet-defining inequalities, and primal heuristics for obtaining upper bound; Qi et al. [94] present a B&B algorithm in which these inequalities are added to the LP-relaxation but only at the top node. Kim et al. [63] use the Hungarian algorithm and a Lagrangian relaxation to obtain tighter lower bounds for the $(3,1)AP_n$. Pustaszeri et al. [93] use a B&B algorithm for the $(5,1)AP_n$, using the LP-relaxation and branching over the variables with the smallest extrapolation error, while also using some preprocessing techniques.

Andrijich and Cacceta [4] focus on the general case of the $(k, 1)AP_n$, using the primal heuristics and the Lagrangian relaxation presented by Balas and Saltzman [13]. Pasiliao et al. [84] propose and compare two B&B algorithms for the $(k, 1)AP_n$, where the first one uses the standard formulation of Pierskalla [87] and branching over variables sharing the same index, while the second relies on a permutation-based formulation and branching over permutations in a subset of indices. The last known exact approach, proposed by Walteros et al. [107] for the $(k, 1)AP_n$ but for the special case of the so-called 'star' costs, is a 'Branch & Price' algorithm. Hence, no B&C or integrated method has been proposed for axial assignment problems.

Non-exact methods include the approximation algorithms of Crama and Spieksma [26] for the $(3, 1)AP_n$ with triangle inequalities, of Bandelt et al. [15] for the $(k, 1)AP_n$ with cost coefficients of four special forms and of Burkard et al. [22] for the $(3, 1)AP_n$ with decomposable cost coefficients (which is a polynomially-solvable case). Several meta-heuristics, such as variable depth interchange, variable neighborhood search and greedy randomized adaptive search are examined by [59].

Concerning the planar multi-index assignment problem, the first exact approach is the B&B algorithm of Vlach [106] for the $(3, 2)AP_n$. A more elaborate B&B algorithm also for $k = 3$ appears in [71], where a relaxation heuristic and a local-improvement tackle the upper bound and a dual heuristic solving a Lagrangian relaxation of the problem obtains lower bounds. A B&C algorithm for the the $(4, 2)AP_n$, which also implements constraint propagation during branching, appears in [7] but for finding only a single (and not a minimum-weight) assignment.

Approximation algorithms for planar problems focus only on the $(3, 2)AP_n$, e.g., Dichkoskaya and Kravtsov[32]. A tabu-search algorithm that improves a given feasible solution has been proposed by Magos [70]. Recent literature for the $(3, 2)AP_n$ focuses mainly on the completion of partial Latin squares, e.g., see Soicher et al. [100].

Despite the great literature size of the problem at hand, there exists no reference for a DSS that encorporates different algorithmic components and assists in a general manner the solution of a $(k, s)AP_n$ instance.

2.3 The *all-different* system

When modelling in Constraint Programming (CP), it is convenient to have at one's disposal some constraints corresponding to a set of constraints, i.e. *global constraints*. These constraints can be associated with more powerful filtering algorithms because

Table 2.2: Planar and Axial 3-index assignment CP models

Axial assignment ($s = 1$)	Planar Assignment ($s = 2$)
$all\_different(x_i^1 : i \in \{0, ..., n-1\}),$	$all\_different(x_{ij} :$ $i \in \{0, ..., n-1\}, \forall j \in \{0, ..., n-1\}),$
$all\_different(x_i^2 : i \in \{0, ..., n-1\}),$	$all\_different(x_{ij} :$ $j \in \{0, ..., n-1\}, \forall i \in \{0, ..., n-1\}),$
$x_i^1, x_i^2 \in \{0, ..., n-1\}, \forall i \in \{0, ..., n-1\}.$	$x_{ij} \in \{0, ..., n-1\},$ $\forall i, j \in \{0, ..., n-1\}.$

they can take into account the simultaneous presence of simple constraints, to further reduce the domains of the variables.

The *all-different* system is a *global* CP constraint; it is a generalization of the '$\neq$' operator and imposes to the variables it involves that they receive pairwise different values. For example, having three variables,

$x_1, x_2, x_3 \in \{1, 4, 5, 7\},$

$all\_different(x_1, x_2, x_3)$ implies that $x_1 \neq x_2, x_1 \neq x_3$, and $x_2 \neq x_3$

hence a feasible solution could be $x_1 = 4, x_2 = 5, x_3 = 1$.

As examined in [5], the *all-different* system describes and generalizes assignment structures. Table 2.2 gives an example of how the axial and planar assignment problem may be modelled after the *all-different* system.

Chapter 3

Feasibility-pump heuristics

This chapter describes a general-purpose heuristic for mixed integer programs (MIP), namely *feasibility-pump*, along with two new variants that include constraint propagation and cutting planes in order to provide better feasible solutions. This is applied to the multi-index assignment problem, as defined in Chapter 2, yet it becomes easy to see that the approach is generic enough to be applied to any MIP; this line of work will be further developed in Chapter 7.

The remainder of this chapter goes as follows; Section 3.1 presents the research background, i.e., the existing feasibility-pump variants. In Section 3.2 we present the integration of constraint propagation and cutting planes in this heuristic, while in Section 3.3 presents the computational results of all feasibility-pump variants, including the new ones, when employed on literature and generated instances for the 3-index axial and planar assignment problem.

3.1 Background

The feasibility-pump (FP) is a rather recent [40], yet extensively studied Linear Programming (LP) based heuristic for mixed integer programs. At each iteration (*pumping cycle*), FP rounds the current LP solution and sets the distance from the derived integer vector as the objective to be minimized at the next iteration, thus guiding the LP towards feasibility. To better display how the basic version of FP, hereafter denoted as FP1, consider that we wish to find a feasible solution for a generic MIP problem of the form

$$(MIP) \quad min\ c^T x \tag{1}$$

$$Ax \geq b \tag{2}$$

$$x_j \in \mathrm{Z},\ \forall j \in J \tag{3}$$

where A is a $n \times m$ matrix. Let $P := \{x \ : \ Ax \geq b\}$ denote the polyhedron of the LP relaxation of the given MIP. We represent with $[\cdot]$ the scalar rounding to the nearest integer and define as $\widetilde{x}$ the rounding of a given x obtained by setting $\widetilde{x}_j := [x_j]$ if $j \in J$ and $\widetilde{x}_j := x_j$ otherwise.

Following this we will consider the L_1-norm distance between a generic point $x \in P$ and a given integer point $\widetilde{x}$ defined as,

$$\Delta(x, \widetilde{x}) = \sum_{j \in J} |x - \widetilde{x}|$$

Given an integer point $\widetilde{x}$, the closest point $x^* \in P$ can be determined by solving the LP

$$min\{\Delta(x, \widetilde{x}) \ : \ Ax \geq b\}$$

If the L_1-norm distance is equal to zero, then $x^*_j (= \widetilde{x}_j)$ is integer for all $j \in J$, so x^* is a feasible solution. Conversely, given a point $x^* \in P$, the integer point $\widetilde{x}$ can be obtained by rounding x^*. If this minimization of the distance is performed iteratively, two trajectories of integer and LP-feasible points are generated, the distance of which is consecutively reduced, hence leading to a feasible integer solution. Algorithm 1 is the Pseudocode of FP1

A variant of FP, proposed by Achterberg and Berthold [3], guiding the LP not only towards feasibility but also towards optimality is the 'Objective FP' (OFP1). This variant uses a convex combination of the distance $\Delta(x, \widetilde{x})$ and the original objective function vector c (assuming $c \neq \mathbf{0}$), i.e.,

$$\Delta^S_\alpha(x, \widetilde{x}) = (1 - a)\Delta(x, \widetilde{x}) + \alpha \frac{\sqrt{|S|}}{||c||} c^T x, a \in [0, 1],$$

where $|| \cdot ||$ is the Euclidean norm of a vector and $\sqrt{|S|}$ is the Euclidean norm of the vector of coefficients in $\Delta(x, \widetilde{x})$. In each cycle, α is geometrically decreased by a fixed factor $\phi \in (0, 1)$, i.e. $\alpha_{t+1} = \alpha_t \phi$ and $a_0 \in [0, 1]$. Algorithm 2 is the pseudocode of OFP1 with the α_t and ϕ parameters set to values as in [3].

Algorithm 1 Pseudocode of the basic version of feasibility-pump

$nIT := 0$
$distance = \infty$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
    return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
    $nIT = nIT + 1$
    $x^* = argmin\{\Delta(x, \widetilde{x}) \ : Ax \geq b\}$
    $distance = \Delta(x, \widetilde{x})$
    if x^* is integer **then**
        return x^*
    end if
    if $\exists\, j \in J \ : \ [x_j^*] \neq \widetilde{x}_j$ **then**
        $\widetilde{x} = [x^*]$
        if cycle detected **then**
            $\rho_j = rand(-0.3, 0.7)$
            for $i = 0$ **to** n **do**
                if $|x_j^* - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
                    flip $\widetilde{x}_j$ //Random restart
                end if
            end for
            empty list l
        end if
        keep the hash of $\widetilde{x}$ in list l
    else
        flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x_j^* - \widetilde{x}_j|$
    end if
end while

Algorithm 2 Pseudocode of the Objective feasibility-pump

$t := 0$
$distance = \infty$
$a_t = 1$
$\phi = 0.9$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
  return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
  $x^* = argmin\{\Delta_\alpha^S(x, \widetilde{x}) \ : Ax \geq b\}$
  $distance = \Delta_\alpha^S(x, \widetilde{x})$
  if x^* is integer **then**
    return x^*
  end if
  if $\exists\, j \in J \ : \ [x_j^*] \neq \widetilde{x}_j$ **then**
    $\widetilde{x} = [x^*]$
    if cycle detected **then**
      $\rho_j = rand(-0.3, 0.7)$
      for $i = 0$ **to** n **do**
        if $|x_j^* - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
          flip $\widetilde{x}_j$ //Random restart
        end if
      end for
      empty list l
    end if
    keep the hash of $\widetilde{x}$ and α_t in list l
  else
    flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x_j^* - \widetilde{x}_j|$
  end if
  $t = t + 1$
  $\alpha_t = \alpha_t \phi$
end while

Furthermore, Fischetti et al. [41] successfully combine FP with Constraint Programming (CP) methods as shown by the improved computational behavior of 'feasibility-pump v2.0' (FP2, OFP2); at this variant, each rounding decision is followed by propagation on the linear constraints. In that regard, FP2 defines another interesting combination of CP and Integer Programming (IP) methods. Algorithms 3 and 4 are the pseudocode of FP2 and OFP2 respectively.

Algorithm 3 Pseudocode of the FP2 heuristic

$nIT := 0$
$distance = \infty$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
$nIT = nIT + 1$
$x^* = argmin\{\Delta(x, \widetilde{x}) \ : Ax \geq b\}$
$distance = \Delta(x, \widetilde{x})$
if x^* is integer **then**
return x^*
end if
if $\exists\, j \in J \ : \ [x_j^*] \neq \widetilde{x}_j$ **then**
round x^* and propagate
if cycle detected **then**
$\rho_j = rand(-0.3, 0.7)$
for $i = 0$ **to** n **do**
if $|x_j^* - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
flip $\widetilde{x}_j$ //Random restart
end if
end for
empty list l
end if
keep the hash of $\widetilde{x}$ in list l
else
flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x_j^* - \widetilde{x}_j|$
end if
end while

Following these variants of FP, several other emerged in the current literature. In this study we focus on the variants that deal with linear optimization problems. De Santis et al. [30], by interpreting the feasibility-pump as a Frank-Wolfe method applied to a non-smooth concave merit function. Following this, De Santis et al.

Algorithm 4 Pseudocode of the OFP2 heuristic

$t := 0$
$distance = \infty$
$a_t = 1$
$\phi = 0.9$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
  return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
  $x^* = argmin\{\Delta^S_\alpha(x, \widetilde{x}) \ : Ax \geq b\}$
  $distance = \Delta^S_\alpha(x, \widetilde{x})$
  if x^* is integer **then**
    return x^*
  end if
  if $\exists\, j \in J \ : \ [x^*_j] \neq \widetilde{x}_j$ **then**
    round x^* and propagate
    if cycle detected **then**
      $\rho_j = rand(-0.3, 0.7)$
      for $i = 0$ **to** n **do**
        if $|x^*_j - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
          flip $\widetilde{x}_j$ //Random restart
        end if
      end for
      empty list l
    end if
    keep the hash of $\widetilde{x}$ and α_t in list l
  else
    flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x^*_j - \widetilde{x}_j|$
  end if
  $t = t + 1$
  $\alpha_t = \alpha_t \phi$
end while

[29] extend their previous results and propose new concave non-differentiable penalty functions for measuring solution integrality and define another FP variant, namely objective re-weighted FP (hereafter denoted as ORFP1) that uses a general class of functions for measuring solution integrality. In this variant the L_1-norm is replaced with a weighted one of the form

$$\Delta_{W,\theta}(x,\widetilde{x}) = \frac{1-\theta}{||\Delta||}\Delta_W(x,\widetilde{x}) + \frac{\theta}{||c||}c^T x$$

with

$$\begin{aligned}&\Delta_W(x,\widetilde{x}) = \sum_{j\in J} w_j|x_j - \widetilde{x}_j|,\\ &J = 1,...,n\end{aligned}$$

where, $||\Delta|| = \sqrt{|J|}$, $\theta \in [0,1]$ decreased at each iteration k by a factor ν (i.e., $\theta^{k+1} = \nu\theta^k$) and $w_j, \forall j \in J$, are positive weights depending on the merit function ϕ chosen. The results of this study concluded that a combination of two such merit functions

$$\phi(x) = \lambda\phi_1(x) + (1-\lambda)\phi_2(x)$$

where $\lambda \in [0,1]$ and modifying the λ parameter at each iteration k as soon as the algorithm stalls, with

$$\begin{aligned}&\phi_1(x) = min\{1 - exp(-\alpha x), 1 - exp(-\alpha(1-x))\},\\ &\phi_2(x) = min\{[1 + exp(-\alpha x)]^{-1}, [1 + exp(-\alpha(1-x))]^{-1}\},\\ &\alpha > 0,\\ &w_j^k = \lambda^k|g_j^k| + (1-\lambda^k)|h_j^k|, j = 1,...,n,\\ &g_j^k \in \partial\phi_1(\widetilde{x}^k), h_j^k \in \partial\phi_2(\widetilde{x}^k),\end{aligned}$$

provides the best possible results. De Santis et al. [29] suggest that constraint propagation could be used in the rounding phase, however, this has not been tested. Employing such a tool in ORFP1, implies a new variant of this heuristic which is hereafter denoted as ORFP2. Algorithms 5 and 6 are the pseudocode for ORFP1 and ORFP2 respectively.

Finally, Boland et al. [19] investigate the benefits of enhancing the rounding procedure with an integer line search that efficiently explores a large set of integer points.

Algorithm 5 Pseudocode of the OFRP1 heuristic

$k := 0$
$distance = \infty$
$\theta^k = 1$
$\nu = 0.9$
$\lambda^k = 0.5$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
  return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
  $x^* = argmin\{\Delta_{W,\theta}(x, \widetilde{x}) = \frac{1-\theta}{||\Delta||}\Delta_W(x, \widetilde{x}) + \frac{\theta}{||c||}c^T x) \ : Ax \geq b\}$
  $distance = \Delta^k_{W,\theta}(x, \widetilde{x})$
  if x^* is integer **then**
    return x^*
  end if
  if $\exists\, j \in J \ : \ [x^*_j] \neq \widetilde{x}_j$ **then**
    round x^*
    if cycle detected **then**
      $\rho_j = rand(-0.3, 0.7)$
      $\lambda^k = 0.5\lambda^k$
      for $i = 0$ **to** n **do**
        if $|x^*_j - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
          flip $\widetilde{x}_j$ //Random restart
        end if
      end for
      empty list l
    end if
    keep the hash of $\widetilde{x}$ and α_t in list l
  else
    flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x^*_j - \widetilde{x}_j|$
  end if
  $k = k + 1$
  $\theta^k = \theta^k \nu$
end while

Algorithm 6 Pseudocode of the OFRP2 heuristic

$k := 0$
$distance = \infty$
$\theta^k = 1$
$\nu = 0.9$
$\lambda^k = 0.5$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
    return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
    $x^* = argmin\{\Delta_{W,\theta}(x,\widetilde{x}) = \frac{1-\theta}{||\Delta||}\Delta_W(x,\widetilde{x}) + \frac{\theta}{||c||}c^T x) \ : Ax \geq b\}$
    $distance = \Delta^k_{W,\theta}(x,\widetilde{x})$
    if x^* is integer **then**
        return x^*
    end if
    if $\exists\, j \in J \ : \ [x^*_j] \neq \widetilde{x}_j$ **then**
        round x^* and propagate
        if cycle detected **then**
            $\rho_j = rand(-0.3, 0.7)$
            $\lambda^k = 0.5\lambda^k$
            for $i = 0$ **to** n **do**
                if $|x^*_j - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
                    flip $\widetilde{x}_j$ //Random restart
                end if
            end for
            empty list l
        end if
        keep the hash of $\widetilde{x}$ and α_t in list l
    else
        flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x^*_j - \widetilde{x}_j|$
    end if
    $k = k + 1$
    $\theta^k = \theta^k \nu$
end while

Such methods are the common theme of several papers over the last decade and have been applied successfully to several combinatorial optimization problems. In that direction, the concept of 'Integrated Methods for Optimization' [55] anticipates further integration of exact and heuristic optimization methods, thus also motivating this work.

3.2 Embedding cutting planes in feasibility-pump

The nature of this heuristic seems intriguing considering that it allows the use of different optimization methods, such as constraint propagation. Therefore this study proposes the inclusion of cutting planes in each pumping cycle; note that this differs substantially from the use of cuts in the restart-phase to avoid FP 'cycling' [18]. The intuition is that cuts added before the pumping phase tighten the LP thus offering FP an improved starting point, while cuts added in each pumping cycle enforce feasibility and possibly drive FP faster to a good integer feasible vector (thus reducing the number of cycles). To test this assumption, we focus on the multi-index assignment problem, i.e., $(k, s)AP_n$, as defined in Chapter 2, particularly the 3-index axial ($s = 1$) and planar ($s = 2$) assignment problem, thus problem-specific cuts are used for this purpose.

For the $(3, 1)AP_n$, we use the two families of inequalities induced by cliques of the intersection graph, which are not included in the IP formulation. As shown by [12], these inequalities are facet-defining (i.e., the strongest possible in polyhedral terms) and are the only such inequalities arising from cliques of the intersection graph. Both families can be separated in $O(n^3)$ steps through the algorithms of [11]. Inequalities arising from odd-holes of size 5 and also separable in $O(n^3)$ steps appear in [94] but not used here, since computationally less effective; i.e., these inequalities are rarely violated if clique inequalities are separated first and their addition does not improve the lower bound. Families of clique inequalities for the $(k, 1)AP_n$ appear in [72], generalizing for all k the ones of [12] and also separable in polynomial time.

For the $(3, 2)AP_n$, there are no clique inequalities other than the ones used (as equalities) in its IP formulation. Inequalities arising from odd-holes are presented in [37], while a much broader class is identified in [73], accompanied by a standard separation routine that runs in $O(n^8)$ steps. This procedure separates *all* odd-hole inequalities hence employed also here.

Since we treat this optimization problem as a minimization one, this novel FP variant implies that cut-addition is used to improve the upper-bound, whereas typical

B&C utilizes cuts only for improving the lower bound. This new variant of FP is denoted as ORFP3. Algorithms 7 and 8 are the pseudocodes for OFP3 and ORFP3.

3.3 Computational results

All components and methods are coded in ANSI C, using the IBM-ILOG CPLEX 12.5 callable library. The experiments are conducted under Linux Ubuntu 14.04, on a quad-core machine (Intel i7, 3.6GHz CPU speed, 16GB RAM). Each experiment includes 5 instances of the same size and the same range of (randomly generated) cost coefficients, thus the average results over each such set of 5 instances are reported per experiment.

All FP variants are allowed up to 2000 pumping cycles, except when employed into a B&C algorithm where the maximum number of pumping cycles is reduced to 20. Parameters α and ϕ of the OFP1, OFP2 and OFP3 are set as in [3], while parameters θ, ν and λ of the ORFP1, ORFP2 and ORFP3 variants are set as in [19].

The main performance metric is the (average) integrality gap, defined as $IG = [(z^* - z_{LP})/z_{LP}] \cdot 100$, where z^* is the value of the solution found by the FP variant and z_{LP} the value of the LP-relaxation. The CPU time required is also reported (in seconds). Additionally, we report the success ratio if less than 1, i.e., the percentage in each set of 5 instances for which a solution is found. Last, the average number of pumping cycles needed by each variant to find a feasible solution is also presented.

We test these algorithms on four classes of non-polynomially solvable instances in the literature, denoted as *bsn* [13], *gpn* [47], *clustern* and *quadn* [45], n being the instance size (the classes in [22, 26] are polynomially solvable hence excluded).

- The class *bsn* uses integer cost coefficients sampled uniformly from the interval $[1, 100]$.

- The class *gpn* uses cost coefficients sampled uniformly from the interval $[1, 300]$; in addition, each instacne is generated via an algorithm ensuring that the uniqueness of the optimal solution.

- The class *clustern* [45] uses coefficients sampled uniformly from three intervals $[0, 49], [450, 499], [950, 999]$, where each interval is selected with a probability equal to 1/3.

- The class *quadn* [45] uses cost coefficients with a value $10000 \cdot z^2$, where z is uniformly distributed in the interval $[0, 1]$.

Algorithm 7 Pseudocode of the OFP3 heuristic

$t := 0$
$distance = \infty$
$a_t = 1$
$\phi = 0.9$
initialize list l
$x^* = argmin\{c^T x\ : Ax \geq b\}$
if x^* is integer **then**
    return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
    add cuts //clique or odd-hole cuts for axial and planar assignment respectively
    $x^* = argmin\{\Delta^S_\alpha(x, \widetilde{x})\ : Ax \geq b\}$
    $distance = \Delta^S_\alpha(x, \widetilde{x})$
    if x^* is integer **then**
        return x^*
    end if
    if $\exists\, j \in J\ :\ [x^*_j] \neq \widetilde{x}_j$ **then**
        round x^* and propagate
        if cycle detected **then**
            $\rho_j = rand(-0.3, 0.7)$
            for $i = 0$ **to** n **do**
                if $|x^*_j - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
                    flip $\widetilde{x}_j$ //Random restart
                end if
            end for
            empty list l
        end if
        keep the hash of $\widetilde{x}$ and α_t in list l
    else
        flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x^*_j - \widetilde{x}_j|$
    end if
    $t = t + 1$
    $\alpha_t = \alpha_t \phi$
end while

Algorithm 8 Pseudocode of the ORFP3 heuristic

$k := 0$
$distance = \infty$
$\theta^k = 1$
$\nu = 0.9$
$\lambda^k = 0.5$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
    return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
    $x^* = argmin\{\Delta_{W,\theta}(x, \widetilde{x}) = \frac{1-\theta}{||\Delta||}\Delta_W(x, \widetilde{x}) + \frac{\theta}{||c||}c^T x) \ : Ax \geq b\}$
    $distance = \Delta^k_{W,\theta}(x, \widetilde{x})$
    if x^* is integer **then**
        return x^*
    end if
    if $\exists\, j \in J \ : \ [x^*_j] \neq \widetilde{x}_j$ **then**
        round x^* and propagate
        if cycle detected **then**
            $\rho_j = rand(-0.3, 0.7)$
            $\lambda^k = 0.5\lambda^k$
            for $i = 0$ **to** n **do**
                if $|x^*_j - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
                    flip $\widetilde{x}_j$ //Random restart
                end if
            end for
            empty list l
        end if
        keep the hash of $\widetilde{x}$ and α_t in list l
    else
        flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x^*_j - \widetilde{x}_j|$
    end if
    $k = k + 1$
    $\theta^k = \theta^k \nu$
end while

Recall from Karapetyan and Gutin [59] that, with cost coefficients sampled in a range $[a, b]$, the optimal solution as n increases tends to an, i.e., the minimum possible assignment weight. Notice that all the above classes sample from an interval that does not increase with n, hence the optimal solution in each instance tends to a constant or n; in addition, their cost coefficients always follow a uniform distribution. Therefore, to enrich this computational study, we generated two further classes of instances:

- the class *axialn* whose coefficients are integer numbers sampled from $U[1, n^k]$, and
- The class *normaln* whose coefficients are integer numbers sampled from normal distribution with $\mu = 1000$ and $\sigma = 200$.

Focus is given on instances for $n \in \{25, 54, 66, 80, 100\}$. For each instance 5 different objective functions are generated hence the results reported per instance are averages over 5 objective functions.

Table 3.3 shows the results of the FP variants when tested on these instances. In general, OFP1 is the fastest variant, but sometimes fails to find a feasible solution. When constraint propagation and cuts are incorporated (in variants OFP2 and OFP3), the quality of solution improves in most cases while feasibility is reached in all instances and the number of pumping cycles is reduced; as expected, this comes at the expense of time. ORFP1 is comparable to OFP1, however constraint propagation and cuts (ORFP3) indeed improve the solution quality and reduce the number of pumping cycles in the majority of the instances, while the computational time remains comparable to ORFP1. That is, the best upper bound is reached only if cuts and constraint propagation are integrated into the reweighed FP-variant [29]. Therefore, cut addition in each pumping cycle is clearly beneficial although quite expensive. Note that no results for the *clustern* and *quadn* instances are presented, because the objective value of their linear relaxation is zero, hence the integrality gap cannot be computed; still, the performance of FP variants in terms of quality of lower bounds, time and pumping cycles is as in the instances reported in Table 3.3.

Having discussed extensively the FP heuristic and its application to the $(k, s)AP_n$, let us now move forward to the integration of this heuristic in an exact algorithm. This raises further questions, first on the performance of the exact algorithm using this heuristic and secondly, if this approach can tackle large instances of the $(k, s)AP_n$. That is the topic of the next chapter.

Table 3.1: $(3,1)AP_n$, FP variants: integrality gap, cycles, time

Instance		OFP1	OFP2	OFP3	ORFP1	ORFP2	ORFP3
axial25	Gap	257.09	551.78	181.08	270.19	363.25	154.92
	Cycles	7	12	3	21	9	5
	Time	0.17	2.02	0.9	0.46 (0.8)	1.56	1.25
axial54	Gap	517.22	729.58	985.86	520.72	1,062.69	485.25
	Cycles	11	7	4	8	11	4
	Time	12 (0.6)	65.06	85.31	10.42 (0.8)	95.56	46.26
axial66	Gap	588.91	1,233.59	1,165.46	814.86	634.44	477.92
	Cycles	5	14	4	7	4	4
	Time	12.3 (0.8)	287.51	251.12	23.86	122.41	122.54
axial80	Gap	647.02	1,364.05	1,392.16	818.07	1,290.83	747.96
	Cycles	5	7	4	8	10	4
	Time	31.68	466.15	581.06	49.38 (0.8)	592.23	310.14
axial100	Gap	860.82	3,070.28	999.21	850.36	1363	1,093.36
	Cycles	3	7	3	4	10	6
	Time	52.3 (0.6)	1,401.86	2,306.47	65.72 (0.8)	1,867.76	1,251.77
bs25	Gap	216.54	190.73	92.73	69.15	180.61	40.07
	Cycles	50	7	3	3	20	3
	Time	0.96 (0.8)	1.34	0.89	0.1 (0.4)	2.97	0.9
bs54	Gap	61.57	38.52	85.18	28.71	81.11	20.37
	Cycles	4	4	4	4	10	4
	Time	5.33 (0.8)	42.7	63.93	5.7 (0.8)	81.81	48.61
bs66	Gap	63.25	70.3	47.88	35.35	35.15	25.76
	Cycles	11	8	4	6	7	4
	Time	22.96 (0.8)	187.32	185.77	16.83 (0.6)	178.88	133.58
bs80	Gap	53.75	48.0	28.25	34.06	45.5	19.25
	Cycles	10	5	4	10	9	3
	Time	38.07(0.4)	366.77	498.53	48.88 (0.8)	535.71	260.34
bs100	Gap	11	59.2	24	16.5	16.8	15
	Cycles	3	8	3	13	5	5
	Time	38.96 (0.2)	1,462.98	1,770.24	116.94(0.8)	1,124.84	1,135.26
gp25	Gap	0.87	2.36	2.32	0.72	4.26	0.71
	Cycles	1	2	2	1	31	1
	Time	0.1 (0.8)	0.62	0.63	0.1	4.43	0.52
gp54	Gap	0.88	1.85	1.1	0.88	0.67	0.42
	Cycles	3	3	1	2	7	1
	Time	3.42	25.33	20.04	2.97	42.18	14.68
gp66	Gap	0.63	0.63	0.78	0.63	1.34	0.78
	Cycles	1	1	1	2	6	2
	Time	8.55 (0.8)	61.26(0.8)	89.38	11.36 (0.8)	144.34 (0.8)	68.23
gp80	Gap	1.29	1.2	1.02	0.95	1.22	0.68
	Cycles	29	3	2	14	7	3
	Time	120.06(0.4)	261.38	302.21	74.96 (0.8)	440.67	236.08
gp100	Gap	2.19	3.95	3.95	2.96	2.89	2.36
	Cycles	9	4	4	5	11	3
	Time	77.89 (0.6)	1,012.17	1,884.71	75.82(0.8)	1,961.55	1,079.67
normal25	Gap	4.64	5.87	5.88	5.73	6,23	3.6
	Cycles	5	4	3	14	8	3
	Time	0.12 (0.8)	0.83	1.01	0.27	1.30	0.81
normal54	Gap	6.35	7.28	7.54	5.88	5.83	6.06
	Cycles	4	4	3	16	7	4
	Time	5.89 (0.8)	42.96	64.03	18.18 (0.8)	59.35	43.58
normal66	Gap	6.34	5.36	6.12	5.18	6.43	5.58
	Cycles	4	4	3	17	7	4
	Time	15.93 (0.6)	108.04	162.78	39.49	176.66	116.24
normal80	Gap	9.14	6.71	8.11	7.25	5.54	5.78
	Cycles	18	4	4	6	7	4
	Time	68.09 (0.8)	323.56	653.21	50.30 (0.8)	176.66	310.31
normal100	Gap	8.91	8.92	8.65	7.87	7.76	6.95
	Cycles	4	4	4	8	8	5
	Time	80.30 (0.6)	987.52	1,999.58	116.65 (0.6)	1,523.25	1,094.26

Chapter 4

Integrated methods for three-index assignment

In this chapter we address the question of whether an exact method can solve large instances of the 3-index axial and planar problems. A relevant question is whether algorithmic and software components that work effectively for different types of assignment are plausible. Here, we propose a *Branch & Cut* (B&C) solver integrating several components, namely cuts that are specific per assignment type, branching on 'Special Ordered Sets of type I', a tabu scheme that is simple enough to remain applicable for all assignment problems, a constraint propagator that can also be used for all assignment problems and *feasibility-pump* (FP) as an LP-based heuristic that also sustains applicability across different assignment problems. In fact, our method uses the improved FP-variant that employs both constraint propagation and cutting planes at each 'pumping cycle' (see Section 3.2). That is, cuts are used in a primal-dual mode to improve the lower bound in a typical manner and guide the heuristic towards a better upper bound. This experimentation shows that the proposed B&C method outperforms a commercial solver, particularly for large-size instances and for planar problems.

The remainder of this chapter flows as follows; Section 4.1 presents the motivation for an exact method that can handle large-size instances of the problem at hand. Section 4.2 presents the cutting planes for axial and planar multi-index assignment, Section 4.3 presents the constraint propagation mechanism that works for any type of assignment, Section 4.4 presents the proposed tabu meta-heuristic, while Section 4.5 describes in detail the exact algorithm that employes the components above. Section 4.6 a detailed computational analysis and Section 4.7 our concluding remarks.

4.1 Motivation and overview

As discussed in Chapter 2, the multi-index assignment problem participates in several optimization problems, thus making it a core problem of its kind. Apart from their nice combinatorial structure, assignment problems enjoy a broad range of applications, thus requiring an effective optimization approach. For example, axial assignment applies to data-association problems [91], to the classification and pairing of human chromosomes [17] and to wafer-to-wafer yield optimization in 3D electronic circuit printing [103]. Planar assignment shares the diverse applications of Latin squares [65] from the statistical design of experiments [54] to error correcting codes [86]. The size of these real problems varies significantly but normally assumes $k \geq 3$ and $n \geq 50$.

Despite the extensive literature on assignment problems, discussed in Chapter 2, there is a gap in exact optimization methods that tackle large-scale instances. In addition, although assignment problems for different values of k and s share a common structure, most existing computational methods focus on a specific s and frequently on a specific k. This appears reasonable given that the $(k,s)AP_n$ becomes $\mathcal{NP}$-complete already for $k = 3$: for $s = 1$ this follows from an early result on 3-index matching [61], while for $s = 2$ it is shown by Frieze [44]. Moreover, the fast-growing size of the $(k,s)AP_n$'s Linear Programming (LP) relaxation made it memory-wise intractable even for small n, thus discouraging the design of appropriate 'Branch & Cut' (B&C) methods. This is indicated by the fact that existing exact methods normally rely on 'Branch & Bound' (B&B) and subgradient algorithms for solving a Langrangean relaxation, as in [13] or [71], with limited (if any) use of cutting planes, e.g., at the top node of the B&B tree by [94] or [72]. Notably, all existing approaches focus on small instances (e.g., for $s = 1, k = 3$ and $n \leq 30$), although related applications ask for much larger ones [33] and B&C approaches can cope with large instances on other optimization problems, e.g., Lysgaard et al. [69].

Several non-exact methods have also been proposed for the 3-index axial problem in the form of either approximation methods for special (or even polynomially solvable) cases as in [26] or meta-heuristics (e.g., [59]) that indeed deal with large instances. A metaheuristic for the 3-index planar problem has been presented by Magos [70]. Let us note here that these approaches focus on a specific type of assignment, thus it is doubtful whether they could become applicable for different values of k and s.

This study addresses the question of whether an exact method can provide optimal or provably near-optimal solutions for large instances of the 3-index axial and planar problems, by exploiting cutting planes that apply for different values of k as in [72]. A relevant question is whether algorithmic and software components that work effectively for different types of assignment are plausible. This attempt is also motivated by the recently formulated notion of *integrated methods for optimization* [55], i.e., methods that exploit the complementary strengths of IP, Constraint Programming and (meta-)heuristics.

Therefore, we propose a B&C algorithm integrating several components, namely cuts that are specific per assignment type but apply for different values of k, branching on 'Special Ordered Sets of type I', a tabu scheme that is simple enough to remain applicable for all values of k and s, a constraint propagator with similar properties and feasibility-pump (FP) as an LP-based heuristic that also sustains applicability across different assignment problems. In fact, the improved FP-variant described in Chapter 3 is used that employs both constraint propagation and cutting planes at each 'pumping cycle'. That is, cuts are used in a primal-dual mode to improve the lower bound in a typical B&C manner and assist the heuristic in improving the upper bound.

Focusing on the 3-index axial and planar assignment problems, we experiment with the individual components and the B&C method on medium and large-size literature instances and on further instances generated for this study. This experimentation shows that indeed the FP variant produces better feasible solutions compared to existing variants, as described in chapter 3, while the B&C method outperforms CPLEX [25] in terms of time and number of nodes in the search tree, especially for large-size instances (several of which remain memory-wise intractable for CPLEX if solved to optimality). The improvement could be attributed more to cutting planes in the axial case, whereas in the planar case constraint propagation and the FP variant, although slower than in the axial case, are the components yielding a more significant improvement.

4.2 Cutting planes

The IP formulation of the $(k, s)AP_n$ gives rise to the so-called intersection graph that has one node per variable (i.e., n^k nodes) and an edge between any two nodes whose corresponding variables cannot both receive value 1, i.e., for any two variables appearing in the same constraint; the formal definition can be found in [72]. Induced subgraphs of the intersection graph give rise to well-known families of inequalities like cliques and odd-holes, as originally presented by Padberg [82]. In fact, the IP formulation contains some of these clique inequalities as equalities. The used cut families in this implementation are discussed below.

For the $(3,1)AP_n$, are used the two families of clique inequalities that are not included in the IP formulation. As shown by Balas and Saltzman [12], these inequalities are facet-defining (i.e., the strongest possible in polyhedral terms) and are the only such inequalities arising from cliques of the intersection graph. Both families can be separated in $O(n^3)$ steps through the algorithms of Balas and Qi [11]. Inequalities arising from odd-holes of size 5 and also separable in $O(n^3)$ steps appear in [94] but not used here, since computationally less effective; i.e., their addition improve only slightly the lower bound. Families of clique inequalities for the $(k,1)AP_n$ appear in [72], generalizing for all k the ones of Balas and Saltzman [12] and also separable in polynomial time. The separation procedure presented by Magos and Mourtos [72] is employed as the main cut generation algorithm for the $(k,1)AP_n$.

For the $(3,2)AP_n$, there are no clique inequalities other than the ones used (as equalities) in its IP formulation. Inequalities arising from odd-holes are presented in [37], while a much broader class is identified in [73], accompanied by a standard separation routine that runs in $O(n^8)$ steps. This procedure separates *all* odd-hole inequalities hence employed also here.

Let us also mention that all general-purpose cuts offered by CPLEX have also been tested without however offering any further improvement.

4.3 Constraint propagation

A mechanism that performs constraint propagation for all classes of $(k,s)AP_n$ is implemented (i.e., for all values of k, s and n) and is capable of supporting any heuristic. This mechanism includes an oracle returning which variables participate in which constraint; two propagating functions *setToOne* and *setToZero* and a backtracking function, invoked when infeasibility is detected.

A stack of size n^k is used, i.e., equal to the number of binary variables, to store the variables that are set to a value together with a flag per variable indicating whether the variable pushed is *set-by-choice* (e.g., by a heuristic) or *set-by-force* (i.e., forced to a value by constraint propagation). This allows backtracking to jump to the last *set-by-choice* variable and set it to its complement hence making it *set-by-force*. At any point, the stack being empty (full) implies that infeasibility is reached (a feasible solution is found). Algorithms 9-11 display in detail how the propagation mechanism works.

Algorithm 9 Pseudocode of the setToOne(x_j, forceFlag) function

/*forceFlag is used to indicate if the variable is set by force or by choice*/
if x_j is undefined **then**
    /*undefined is a variable that is not set to any value, i.e., 0 or 1*/
    $x_j = 1$;
    fetch the constraints that x_j participates in;
    for every constraint c_i of the above **do**
        fetch the variables that participate in c_i;
        for every variable x_ℓ of the above **do**
            forceFlag = TRUE
            if (setToZero(x_ℓ, forceFlag)==FALSE) **then**
                return FALSE;
            end if
        end for
    end for
    push x_j in the stack
    return TRUE;
else if $x_j == 1$ **then**
    return TRUE;
else
    return FALSE;
end if

Once a variable is set to one, *setToOne* calls *setToZero* for all variables appearing in some constraint together with that variable. Once a variable is set to zero, *setToZero* checks if in any constraint this variable appears, there remains a single variable not set to a value, and calls for this variable *setToOne* (if this fails, infeasibility is detected). Using these two functions, this code implements a recursive depth-first propagation on the constraints of $(k,s)AP_n$. Since the number of these constraints is $\binom{k}{s} \cdot n^s$ [6], i.e., it increases with s, constraint propagation is more effective for planar rather than axial problems. Stronger propagation is possible via the representation of the $(k,s)AP_n$ as a set of *all-different* constraints [9], but not implemented here, as it is too expensive computationally.

4.4 Tabu-search

The tabu-search mechanism presented here applies to the $(k,s)AP_n$ for any values of k and s. It uses a fixed-size list to keep track of the tabu moves and terminates the

Algorithm 10 Pseudocode of the setToZero(x_j, forceFlag) function

```
/*forceFlag is used to indicate if the variable is set by force or by choice*/
if x_j is undefined then
    /*undefined is a variable that is not set to any value, i.e., 0 or 1*/
    x_j = 0;
    fetch the constraints that x_j participates in;
    for every constraint c_i of the above do
        fetch the variables that participate in c_i;
        if there is only one undefined variable x_ℓ in c_i then
            forceFlag = TRUE
            if (setToOne(x_ℓ, forceFlag)==FALSE) then
                return FALSE;
            end if
        end if
    end for
    push x_j in the stack
    return TRUE;
else if x_j == 0 then
    return TRUE;
else
    return FALSE;
end if
```

Algorithm 11 Pseudocode of the backTrack() function.

```
fetch the variable x_j that was set to one or zero by choice;
undo the changes that were caused by this choice;
/*i.e., pop the variables set by force from the stack and mark them as undefined*/
if (x_j == 1) then
    forceFlag = TRUE
    if setToZero(x_j, forceFlag)==FALSE then
        return FALSE;
    end if
else
    forceFlag = TRUE
    if setToOne(x_j, forceFlag)==FALSE then
        return FALSE;
    end if
end if
return TRUE;
```

Figure 4.1: Tabu moves on a solution of the $(3,1)AP_3$ and a solution of the $(3,2)AP_3$

	x^0	$\rightarrow swap(2,1,3) \rightarrow$	x^1	$\rightarrow interch(2,3) \rightarrow$	x^2
index	1 2 3		1 2 3		1 2 3
$(3,1)AP_3$	1 2 3		1 2 3		1 **3** **2**
	2 3 1		2 **1** 1		2 **1** **1**
	3 1 2		3 **3** 2		3 **2** **3**
$(3,2)AP_3$	1 1 3		1 **3** 3		1 **3** **3**
	1 2 2		1 2 2		1 **2** **2**
	1 3 1		1 **1** 1		1 **1** **1**
	2 1 2		2 **3** 2		2 **2** **3**
	2 2 1		2 2 1		2 **1** **2**
	2 3 3		2 **1** 3		2 **3** **1**
	3 1 1		3 **3** 1		3 **1** **3**
	3 2 3		3 2 3		3 **3** **2**
	3 3 2		3 **1** 2		3 **2** **1**

procedure, if no solution of better quality is found after a fixed number of iterations.

Recall that a solution of the $(k,s)AP_n$ is a collection of n^s disjoint k-tuples, each such tuple representing a variable set to one in the corresponding vector. The tabu-move is defined as a swap of values in the same index, denoted hereafter as $swap(index, value1, value2)$. Applying this move to a solution x^0 leads to another solution x^1 that differs from x^0 in exactly $2 \cdot n^{s-1}$ variables set to one [6, Remark 7]. If within a fixed number of iterations of tabu-swaps no better solution is found, a restart is performed defined by the *interchange* move which is defined as the swap of all values in 2 indices, i.e., $interch(index1, index2)$. In this way, the algorithm restarts from a different neighbourhood hoping that a new better feasible solution will be found. Figure 4.1 displays these tabu moves on solutions of $(3,1)AP_3$ and $(3,2)AP_3$; at the second index of x^0, values 1 and 3 are swapped, hence attaining a new feasible solution x^1, while by interchanging indices 2 and 3 of x^1 a new feasible point x^2 is attained.

The value of a swap move is calculated with respect to the coefficients of the $2 \cdot n^{s-1}$ variables affected by that move. For example, the value of the first move in Figure 4.1 for the $(3,1)AP_3$ is $c_{332} + c_{211} - c_{231} - c_{312}$. At each iteration, tabu-search selects the swap of minimum value not appearing in the tabu list, among all indices and all value pairs per index. Since each swap affects $2 \cdot n^{s-1}$ variables in a feasible solution, each tabu iteration requires $O(k \cdot n^{s+1})$ steps, i.e., it gets more demanding as s increases. If within 50 iterations no better feasible solution is found the *interchange* move is performed to restart the algorithm from a new neighborhood. The cost of the move is again calculated with respect to the coefficients of the variables affected by that move, hence the *interchange* move with the less possible objective value is chosen. Note that this is a simple yet fast and (the first) generic for the $(k, s)AP_n$ tabu mechanism.

4.5 Branch & cut

Let us now describe the integration of all the above within a B&C method. As evident from Table 2.1, each constraint in the IP model defines a *Special Ordered Set of type I* (SoS-I). Hence, the B&C performs SoS-I branching and prioritizes (i) each variable in an SoS-I in increasing order with respect to their reduced cost at the LP-optimum of the top node and (ii) each SoS-I in increasing order with respect to their minimum integer infeasibility. For example, suppose that the variables of the constraint $x_1 + x_2 + x_3 + x_4 = 1$ have reduced costs at the top node $100, 200, 400, 500$ respectively. Furthermore, assume that there is a known fractional solution of $x_1 = 0.1$ and $x_4 = 0.9$. In SoS parlance, the weighted average of the set is $\frac{0.1\cdot 100+0.9\cdot 500}{0.1+0.9} = 460$. The set is then split before the variable with reduced cost exceeding the weighted average, i.e., x_1, x_2, x_3 will be in one subset and x_4 in the other. This means that when branching over this SoS-I, one branch will have $x_1 = x_2 = x_3 = 0$ and the other $x_4 = 0$. The node selection criterion is 'best bound'.

Alternative branching strategies have also been examined, namely

- prioritizing variables in respect with their reduced costs at each node and constraints according to the sum of the reduced costs of variables in their support;
- prioritizing variables according to the number of active constraints (including cuts) in which they participate [85];
- a combination of the two above strategies.

These branching strategies form a representative set, since the first is guided mainly by optimality and the second by feasibility (i.e., it performs well if obtaining a feasible solution is hard). Interestingly, none of these elaborate strategies performed better than the simpler one that has been selected, as shown by preliminary experimentation (not presented here).

The efficiency of a B&C scheme often depends on the frequency and the intensity of cut addition, e.g., it is common to add cuts at several, but not all, nodes usually of small depth [69, 34]. To investigate the maximum depth and the maximum rounds of cut addition, several experiments on different strategies are performed. The following strategies have been examined:

- **Aggressive**: adding cuts at nodes that are up to 10% of the maximum tree depth, for one round and up to n/k cuts per round;

- **Bold**: adding cuts at nodes that are up to 10% of the maximum tree depth, for one round and up to n/k^2 cuts per round;

- **Moderate**: adding cuts at nodes that are up to 5% of the maximum tree depth, for one round and up to n/k cuts per round;

- **Light**: adding cuts at nodes that are up to 5% of the maximum tree depth, for one round and up to n/k^2 cuts per round.

Also, cuts added at some node are maintained in all its antecedent nodes. However, the decision of whether a cut is added or not at the cut addition phase is left to CPLEX. Preliminary experimentation with these cut addition strategies showed that the **Light** strategy has the most robust performance across the testing bed.

Although there are several heuristics for $(k, s)AP_n$ that are tailor-made for specific values of k and s, here we focus on the effectiveness of general-purpose heuristics that work over any assignment type. Hence the emphasis on FP variants, any of which can be employed as the primal heuristic. Calling such an expensive heuristic too often would be a burden in terms of solution time, hence the simple strategy of calling this heuristic a few times (or only at the top-node) is preferred and then using just the default heuristic of CPLEX; this gives also a more sound basis of comparison. As usual, the tabu-search can be invoked whenever a new or an improved solution is found either by the heuristic or by branching.

4.6 Computational results

All components and methods are coded in ANSI C, using the IBM-ILOG CPLEX 12.5 callable library. The experiments are conducted under Linux Ubuntu 14.04, on a quad-core machine (Intel i7, 3.6GHz CPU speed, 16GB RAM). Each experiment includes 5 instances of the same size and the same range of (randomly generated) cost coefficients, thus the average results over each such set of 5 instances are reported per experiment.

All FP variants are allowed up to 2000 pumping cycles, except when employed into a B&C algorithm where the maximum number of pumping cycles is reduced to 20. The main performance metric is the (average) integrality gap, defined as $IG = [(z^* - z_{LP})/z_{LP}] \cdot 100$, where z^* is the value of the solution found by the FP variant and z_{LP} the value of the LP-relaxation. The CPU time required is also reported (in seconds). Additionally, the success ratio if less than 1 is reported, i.e., the percentage in each set of 5 instances for which a solution is found. Last, the average number of pumping cycles needed by each variant to find a feasible solution is also presented.

Regarding the exact algorithms, preliminary computational experience showed that for certain large instances, i.e., $n \geq 50$ for the axial and $n \geq 20$ for the planar case, CPLEX runs out of memory way before reaching the optimal solution. For this reason and in order to have a sound basis of comparison, a time limit of 3 hours is set on all exact schemes and the integrality gap is calculated, i.e., $IG = [(z_{IP} - z_{LP})/z_{LP}] \cdot 100$, where, z_{IP} is the upper bound and z_{LP} is the lower bound reached within this time-frame. To test the effect of each component (i.e., SoS-I branching, cuts and ORFP3 as the most competitive FP-variant), the results of four exact methods are shown compared to the CPLEX default (i.e., single-threaded mode, pre-solver turned off, with all other CPLEX features left to default settings). Each method has an additional component compared to the previous one; that is, the *SoS-I* scheme uses only the respective branching strategy, *SoSCuts* uses SoS-I branching and cuts, *SoSCutsFP* includes also ORFP3 and, last *SoSCO-T* uses all the aforementioned components plus the tabu-search. For all four methods, all general-purpose CPLEX cuts and the pre-solver are turned-off, the single-threaded mode is used and the CPLEX heuristic is set on default settings, except from the *SoSCO-T* where it is turned off. The results are discussed next.

4.6.1 3-index axial assignment

We experiment with four classes of non-polynomially solvable instances in the literature, denoted as *bsn* [13], *gpn* [47], *clustern* and *quadn* [45], n being the instance size (the classes in [22, 26] are polynomially solvable hence excluded).

- The class *bsn* uses integer cost coefficients sampled uniformly from the interval $[1, 100]$.

- The class *gpn* uses cost coefficients sampled uniformly from the interval $[1, 300]$; in addition, each instacne is generated via an algorithm ensuring that the uniqueness of the optimal solution.

- The class *clustern* [45] uses coefficients sampled uniformly from three intervals $[0, 49], [450, 499], [950, 999]$, where each interval is selected with a probability equal to 1/3.

- The class *quadn* [45] uses cost coefficients with a value $10000 \cdot z^2$, where z is uniformly distributed in the interval $[0, 1]$.

Let us recall from Karapetyan and Gutin [59] that, with cost coefficients sampled in a range $[a, b]$, the optimal solution as n increases tends to an, i.e., the minimum possible assignment weight. Notice that all the above classes sample from an interval that does not increase with n, hence the optimal solution in each instance tends to a constant or n; in addition, their cost coefficients always follow a uniform distribution. Therefore, to enrich this computational study, two further classes of instances are generated:

- the class *axialn* whose coefficients are integer numbers sampled from $U[1, n^k]$, and

- The class *normaln* whose coefficients are integer numbers sampled from normal distribution with $\mu = 1000$ and $\sigma = 200$.

Instances for $n \in \{25, 54, 66, 80, 100\}$ are examined. For each instance 5 different objective functions are generated hence the results reported per instance are averages over 5 objective functions.

Table 2 shows the results of the FP variants when tested on these instances. In general, OFP1 is the fastest variant, but sometimes fails to find a feasible solution. When constraint propagation and cuts are incorporated (in variants OFP2 and OFP3), the quality of solution improves in most cases while feasibility is reached in all instances and the number of pumping cycles is reduced; as expected, this comes at the expense of time. ORFP1 is comparable to OFP1, however constraint propagation and cuts (ORFP3) indeed improve the solution quality and reduce the number of pumping cycles in the majority of the instances, while the computational time remains comparable to ORFP1. That is, the best upper bound is reached only if cuts and constraint propagation are integrated into the reweighed FP-variant [29]. Therefore, cut addition in each pumping cycle is clearly beneficial although quite expensive. Note that, no results for the *clustern* and *quadn* instances are presented, because the objective value of their linear relaxation is zero, hence the integrality gap cannot be computed; still, the performance of FP variants in terms of quality of lower bounds, time and pumping cycles is as in the instances reported in Table 2.

Table 3 shows the average percentage of the decrease in the upper bound achieved by tabu-search on a subset of instances, i.e., $[(z^* - z^t)/z^*] \cdot 100$ where z^* and z^t are the values of the solutions found by an FP variant and tabu-search respectively. Additionally, the required time for tabu-search to improve a given solution is reported. It is obvious that tabu-search often improves significantly its starting solution, although this depends on the quality of this solution. The smaller decrease achieved regarding ORFP3.0 indicates that tabu-search succeeds fewer times in improving the solution obtained by ORFP3.0, whereas in most cases it does improve the upper bound obtained by other variants.

Table 4.1: $(3,1)AP_n$, tabu-search: objective value reduction (%), time

Instance		OFP1-T	OFP2-T	OFP3-T	ORFP1-T	ORFP2-T	ORFP3-T
axial25	objRed	25.04	40.33	12.56	23.70	27.26	9.29
	Time	0.05	0.03	0.11	0.07	0.0	0.05
axial54	objRed	24.27	23.19	25.40	20.74	27.24	16.08
	Time	0.47	0.58	0.01	0.01	0.01	0.01
axial66	objRed	17.37	39.63	43.56	18.68	28.72	8.72
	Time	0.66	0.02	0.02	1.03	0.53	1.03
axial80	objRed	0.19	34.63	30.03	6.81	28.45	5.79
	Time	3.60	0.04	1.82	2.29	0.94	1.83
axial100	objRed	4.0	38.36	7.57	8.90	20.45	17.75
	Time	2.99	0.09	3.55	2.31	3.54	3.55
bs25	objRed	33.67	33.00	3.71	7.40	27.14	0.0
	Time	0.0	0.0	0.05	0.07	0.03	0.14
bs54	objRed	23.63	9.95	21.87	4.5	23.70	0.87
	Time	0.36	0.29	0.01	0.71	0.29	1.13
bs66	objRed	18.88	22.06	13.18	11.17	11.14	3.41
	Time	0.02	0.02	0.02	0.02	0.53	0.53
bs80	objRed	24.40	13.21	6.59	10.36	12.86	4.79
	Time	0.04	0.04	1.84	0.07	0.95	1.83
bs100	objRed	0.0	18.5	5.93	3.58	4.42	3.78
	Time	8.85	0.09	0.11	2.28	3.59	3.64
gp25	objRed	0.32	0.48	0.79	0.0	2.19	0.0
	Time	0.12	0.13	0.12	0.15	0.09	0.15
gp54	objRed	0.36	1.17	0.54	0.36	0.01	0.0
	Time	0.87	0.87	0.87	0.87	0.88	1.15
gp66	objRed	0.0	0.0	0.0	0.0	0.56	0.0
	Time	2.6	2.6	2.6	2.6	1.95	2.6
gp80	objRed	0.57	0.44	0.28	0.19	0.49	0.0
	Time	3.67	2.76	3.68	3.45	1.86	3.68
gp100	objRed	31.03	27.01	39.51	0.82	0.30	0.0
	Time	2.98	0.09	1.82	0.09	0.09	8.77

Table 4 shows the results of the exact algorithms. When looking at the performance of CPLEX default, one can easily notice the difference in the number of nodes and solution time among different classes of instances. Figure 4.2 shows this differentiation of the required time in CPU seconds where the new instances (*axialn*, *normaln*) for $n \geq 66$ require the maximum time ($3h$). The literature instances *bsn* and *gpn* are solved significantly faster compared to *axialn* and *normaln* ones. This is more evident in instances with $n \geq 25$. Given that all literature instances use an non-increasing with n range of cost coefficients, this verifies that the optimal solution asymptotically tends to n, i.e., the minimal possible assignment weight [59]. However, this phenomenon is rarely evident in the *axialn* instances, because the range widens as n increases, thus making these instances 'harder' to solve.

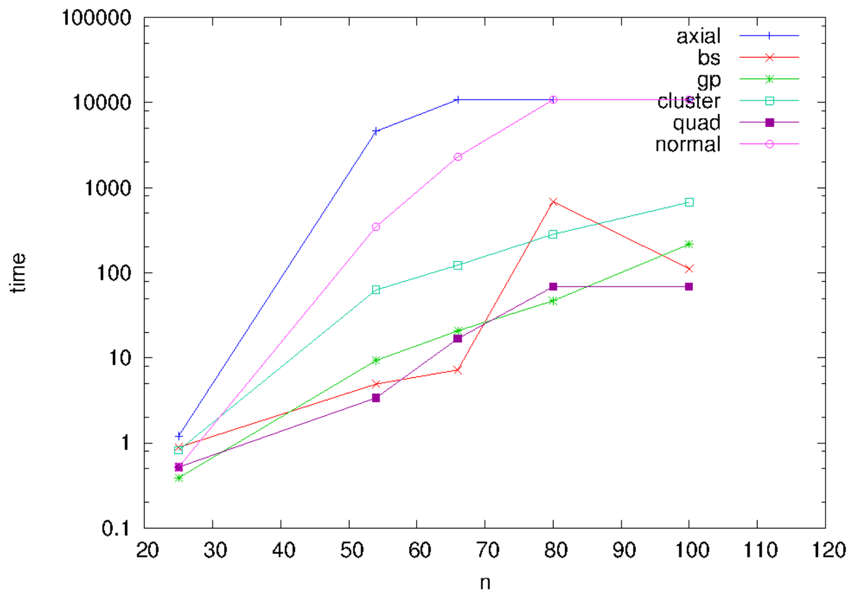


Figure 4.2: CPLEX performance on 3-index axial instances

When looking on the *bsn*, *gpn* and *quadn* classes (Table 4), the new exact algorithms perform comparably to CPLEX, in terms of nodes. However, this does not apply in terms of time, that is, these new schemes are time-wise more expensive due to the computational time required by ORFP3. This means that CPLEX suffices for these classes, mostly because of the particularity described above. Regarding the *clustern* class, it is obvious that the SoSCuts and SoSCutsFP exact schemes improve on CPLEX in terms of both nodes and time. Additionally, when looking on the *axialn* and *normaln* classes, an improvement on CPLEX in terms of nodes, time and integrality gap (presented in the table if non-zero) is notable, which indicates that these B&C methods indeed prune the search tree more effectively. When comparing the new B&C methods with each other on the *axialn* and *normaln* classes, it seems that sometimes ORFP3 reduces the number of nodes but increases the time to optimality. However, this burden, in terms of time, seems to pay off for larger instances ($n \geq 66$) given the integrality gap that is reached within the time-frame of $3h$. When the tabu-search (column 'SoSCO-T') is used, the results are again comparable to the performance of CPLEX; this happens because, as the quality of the FP variant improves, tabu-search rarely improves it. Overall, the improvement over CPLEX appears more substantial for the *axialn*, *clustern* and *normaln* classes, and can be summarized as follows:

- regarding the *axialn* class ($66 \leq n \leq 100$), the integrality gap reached by the SoSCutsFP scheme within $3h$ is on average 12.06% smaller than the one reached by CPLEX;
- regarding the *clustern* class whose instances are all solved to optimality, the SoSCuts scheme reduces on average for all instances ($25 \leq n \leq 100$) the number of nodes for up to 70.51% and the required time for up to 52.44% when compared to CPLEX;
- regarding the *normaln* class ($80 \leq n \leq 100$), the integrality gap reached by the SoSCutsFP scheme within $3h$ is on average 20.57% smaller than the one reached by CPLEX.

4.6.2 3-index planar assignment

For the $(3,2)AP_n$, let us focus on the instances with cost coefficients sampled from $U[1, n^k]$. As previously, 5 different objective functions for $n \in \{10, 20, 30, 40, 50\}$ are generated, thus, the average results over these 5 functions per instance are reported. Table 5 shows the results of the FP variants. As for the axial case, OFP1 requires less time than the other variants, however it provides the poorest quality of solutions on average. When constraint propagation and cuts are employed (OFP3) the number of pumping cycles is reduced and the solution quality improves. Regarding ORFP1, it provides solutions of better quality than OFP1 in a comparable amount of time. When constraint propagation and cuts are employed (ORFP3) the number of pumping cycles is reduced and the solution quality is further improved. However, time-wise ORFP3 is far more expensive. This is obviously due to the cut addition (recall that the separation of odd-hole cuts takes $O(n^8)$ steps).

It is also worth noting that tabu-search fails in all instances to improve a given solution of a planar assignment. This is attributed to the fact, that the quality

Table 4.2: $(3,1)AP_n$, exact methods: nodes, time, integrality gap within 3 hours

Instance		CPLEX	SoS-I	SoSCuts	SoSCutsFP	SoSCO-T
axial25	Nodes	627	565.4	**429**	520.4	587
	Time	1.21	1.12	**0.98**	2.37	2.68
axial54	Nodes	368,075	335,959	354,046	**313,483**	380,417
	Time	4,608.23	**4,073.78**	5,126.36	4,654.48	5,615.26
axial66	Nodes	445,512	400,366	359,688	366,384	327,937
	Time	$3h$	$3h$	$3h$	$3h$	$3h$
	Gap	34.24	38.2	**29.97**	31.79	51.196
axial80	Nodes	179,122	180,121	167,720	173,515	165,235
	Time	$3h$	$3h$	$3h$	$3h$	$3h$
	Gap	133.07	92.33	106.53	**74.02**	94.91
axial100	Nodes	71,284	66,864	72,323	70,859	68,889
	Time	$3h$	$3h$	$3h$	$3h$	$3h$
	Gap	193.11	192.76	154.97	**153.48**	210.41
bs25	Nodes	399	247	205	**187**	445
	Time	0.89	**0.65**	0.67	1.52	2.07
bs54	Nodes	**64**	99	80	72	630
	Time	**4.93**	7.99	8.51	56.23	68.96
bs66	Nodes	0	0	0	0	357
	Time	**7.22**	12.26	14.77	147.14	162.65
bs80	Nodes	7,270	586	465	**446**	869
	Time	686.67	72.28	**59.16**	318.12	352.50
bs100	Nodes	511	659	**118**	570	804
	Time	**112.01**	187.30	120.86	1,297.36	1,351.56
gp25	Nodes	0	0	0	0	0
	Time	0.39	**0.2**	0.19	0.67	0.66
gp54	Nodes	0	**0**	0	0	0
	Time	9.31	**3.92**	4.61	18.08	18.10
gp66	Nodes	0	0	**0**	0	0
	Time	20.82	14.05	**12.41**	79.48	79.42
gp80	Nodes	0	0	**0**	0	0
	Time	46.72	33.19	**30.63**	263.84	265.10
gp100	Nodes	889	23,934	1,097	893	**360**
	Time	**216.45**	4,469.35	735.61	1,643.05	5,603.17
cluster25	Nodes	289	388	**258**	361	568
	Time	0.82	0.88	**0.76**	1.85	2.70
cluster54	Nodes	5,282	1,330	667	**507**	1,194
	Time	62.87	24.72	**19.25**	54.86	68.45
cluster66	Nodes	4,975	1,205	937	**583**	1,271
	Time	122.81	57.75	**44.29**	143.6	171.59
cluster80	Nodes	5,901	902	**734**	1,050	1,102
	Time	282.42	97.40	**98.77**	394.76	440.21
cluster100	Nodes	6,968	1,636	1,018	**529**	1,025
	Time	674.92	355.66	**294.86**	1,127.91	1,250.70
quad25	Nodes	105	176	**58**	58	589
	Time	0.52	0.52	**0.39**	1.23	2.38
quad54	Nodes	**0**	0	0	0	550
	Time	**3.39**	5.18	6.72	51.87	68.21
quad66	Nodes	318	228	185	**154**	1,070
	Time	**16.87**	24.34	24.20	116.72	157.41
quad80	Nodes	746	186	**142**	266	871
	Time	68.53	**51.76**	52.83	379.85	459.32
quad100	Nodes	306	656	313	**180**	672
	Time	**68.53**	302.87	159.84	143.45	1,180.39
normal25	Nodes	100	84	**68**	69	100
	Time	0.52	0.39	**0.38**	1.16	1.65
normal54	Nodes	24,557	17,680	16,786	16,972	**15,009**
	Time	348.38	**253.41**	314.70	369.75	518.29
normal66	Nodes	86,725	75,181	61,144	**48,830**	58,384
	Time	2,301.12	2,080.07	2,023.60	**1,894.42**	3,761
normal80	Nodes	125,863	139,554	115,368	122,508	41,741
	Time	$3h$	$3h$	$3h$	$3h$	$3h$
	Gap	1.44	1.13	1.34	**1.0**	1.72
normal100	Nodes	24,646	22,870	20,835	21,328	16,947
	Time	$3h$	$3h$	$3h$	$3h$	$3h$
	Gap	3.48	3.15	3.29	**3.11**	3.46

of the solution found by any FP variant is substantially better than in an axial assignment problem. Therefore results of tabu-search on solutions of the $(3,2)AP_n$ are not presented.

Table 6 shows the results of the exact algorithms for the planar instances (using again the 'Light' cut strategy that outperforms all others as in the axial case). All of the new algorithmic schemes perform better than CPLEX in terms of nodes and time. Among them, the *SoSCutsFP* scheme requires more time, which is reasonable given the time required by ORFP3. The effect though of ORFP3 is far more evident on larger instances: the number of nodes visited by the schemes that employ cuts is much smaller but time is longer due to the time required by the separation algorithm, i.e., CPLEX and *SoS-I* scheme spend less time at each node. However, as shown in Figure 4.3, the integrality gap reached by *SoSCutsFP* scheme is the minimum possible, which indicates that indeed all these components prune significantly the search tree. The effect on the integrality gap is also depicted in Figure 4.3.

Table 4.3: $(3,2)AP_n$, exact methods: nodes, time, integrality gap within 3 hours

Instance		CPLEX	SoS-I	SoSCuts	SoSCutsFP
$n = 10$	Nodes	813	695	597	**624**
	Time	2.9	1.66	**1.6**	1.92
	Gap	0	0	0	0
$n = 20$	Nodes	403,464	350,702	300,106	296,380
	Time	$3h$	$3h$	$3h$	$3h$
	Gap	24.47	27.63	29.25	**17.07**
$n = 30$	Nodes	42,738	37,188	18,029	18,030
	Time	$3h$	$3h$	$3h$	$3h$
	Gap	89.36	80.39	87.51	**26.20**
$n = 40$	Nodes	11,986	10,266	7,927	5,927
	Time	$3h$	$3h$	$3h$	$3h$
	Gap	159.68	137.33	160.38	**46.92**
$n = 50$	Nodes	1,374	1,685	1,332	23
	Time	$3h$	$3h$	$3h$	$3h$
	Gap	199.34	202.56	202.44	**53.64**

4.7 Beyond three-index assignment

This chapter presents a solver for the 3-index axial and planar problem that integrates constraint propagation, problem-specific cuts, SoS-I branching and feasibility-pump enhanced by cut addition in each pumping cycle. This solver performs better than

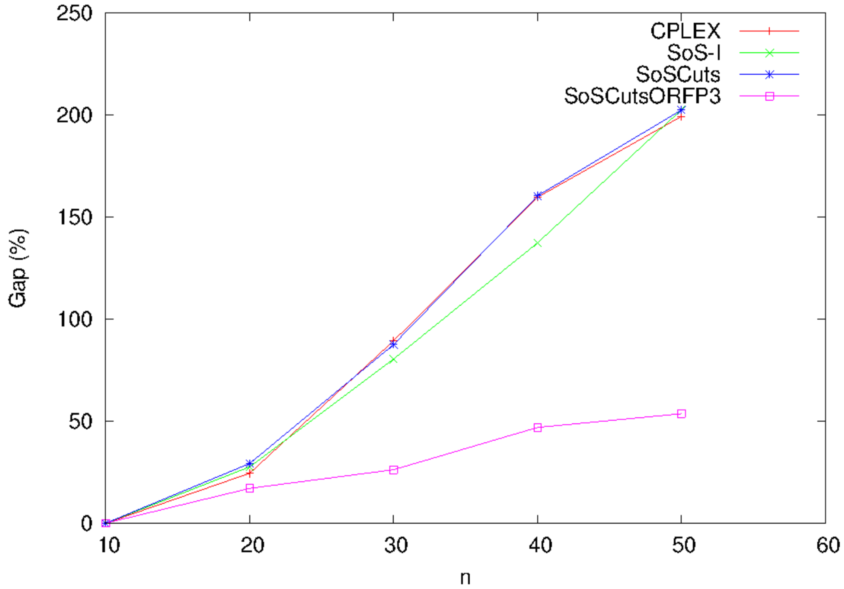


Figure 4.3: Integrality gap differentiation of exact schemes on planar instances

CPLEX, particularly for larger instances and more evidently in the planar case. Further experimentation may offer deeper insights on both the performance of such an approach and on the ability to solve exactly even larger-scale instances or other multi-index assignment problems. An important aspect of this approach is its versatility, for example in terms of including a subset of the selected components or an alternative FP variant as a primal heuristic. To demonstrate that, Table 7 shows some indicative results for larger values of k, using different algorithmic components (column 'Custom'): for $k = 4$ these are only ORFP1 and SoS-branching, while for $k = 5$ they include ORFP3, SoS-branching and cuts. Overall, this work, apart from addressing a literature gap concerning exact methods on large instances of a widely-studied class of problems, offers sufficient motivation for further research.

Table 4.4: $(4,1)AP_n$ and $(5,1)AP_n$, exact methods

Instance		CPLEX	Custom
$k = 4, n = 10$	Nodes	516	343
	Time	0.76	0.60
	Gap	0	0
$k = 5, n = 10$	Nodes	22,785	17,364
	Time	170.71	135.04
	Gap	0	0

Chapter 5

Decision support for multi-index assignment

This chapter presents a Decission Support System (DSS) for the multi-index assignment problem. Considering that numerous applications include an assignment structure or are modelled after the mathematical model of the problem at hand, it would be useful to have a system that includes algorithmic components and can efficiently provide solutions regardless of the application context. Therefore, in this chapter we present a general-purpose DSS for the $(k, s)AP_n$, namely MAPS (Multi-index Assignment Problem Solver). The analysis of this system includes the user requirements of the system, the workflow of the use-cases and the algorithmic components incorporated in the system. This work is part of a research project, supported by the National Research Fund, namely Archimedes III sub-project 28, focusing on the Multi-index Assignment problem and all-different Systems.

The remainder of this chapter goes as follows; In Section 5.1 we present the requirements analysis of the proposed DSS; Section 5.2 lists and shortly describes the employed algorithms and Section 5.3 presents the DSS in terms of final user screens and usage.

5.1 Requirements analysis

This section describes the user requirements, functional and non-functional, along with the data requirements of the integrated solver (MAPS). We present a list of proposed use cases in subsection 5.1.1, which we consider sufficient for DSS design but non-exhaustive as real-life applications may raise additional such cases. Nevertheless, it indicates some basic non-functional and functional user requirements of an integrated solver.

In the rest of this section we describe in Subsection 5.1.2 the non-functional requirements, while Subsection 5.1.3 presents the required data entities with their associations.

5.1.1 Description of use cases

The major functionality provided by the system is depicted in the following use case diagram (Figure 5.1). This diagram includes the entities (external systems or human) that interact with the system, i.e., actors, and specifies in a more detailed way the functionality of MAPS.

Table 5.1 presents a list of the aforementioned use cases, while each of these use cases is described thoroughly in Appendix A.

Table 5.1: List of use cases

Use Case ID	*Use Case Name*
1	Register
2	Log in
3	View saved solved instances
4	Solve a new $(k, s)AP_n$ instance
5	View a $(k, s)AP_n$ solution from a single algorithm
6	Load costs of a $(k, s)AP_n$ instance and solve it
7	Save a $(k, s)AP_n$ instance solution
8	Delete a saved $(k, s)AP_n$ instance solution
9	Solve a new *all-different* instance
10	Save a *all-different* instance solution
11	Delete a saved *all-different* instance solution
12	View the MAPS manual
13	Edit user account
14	Log out
15	Delete personal account

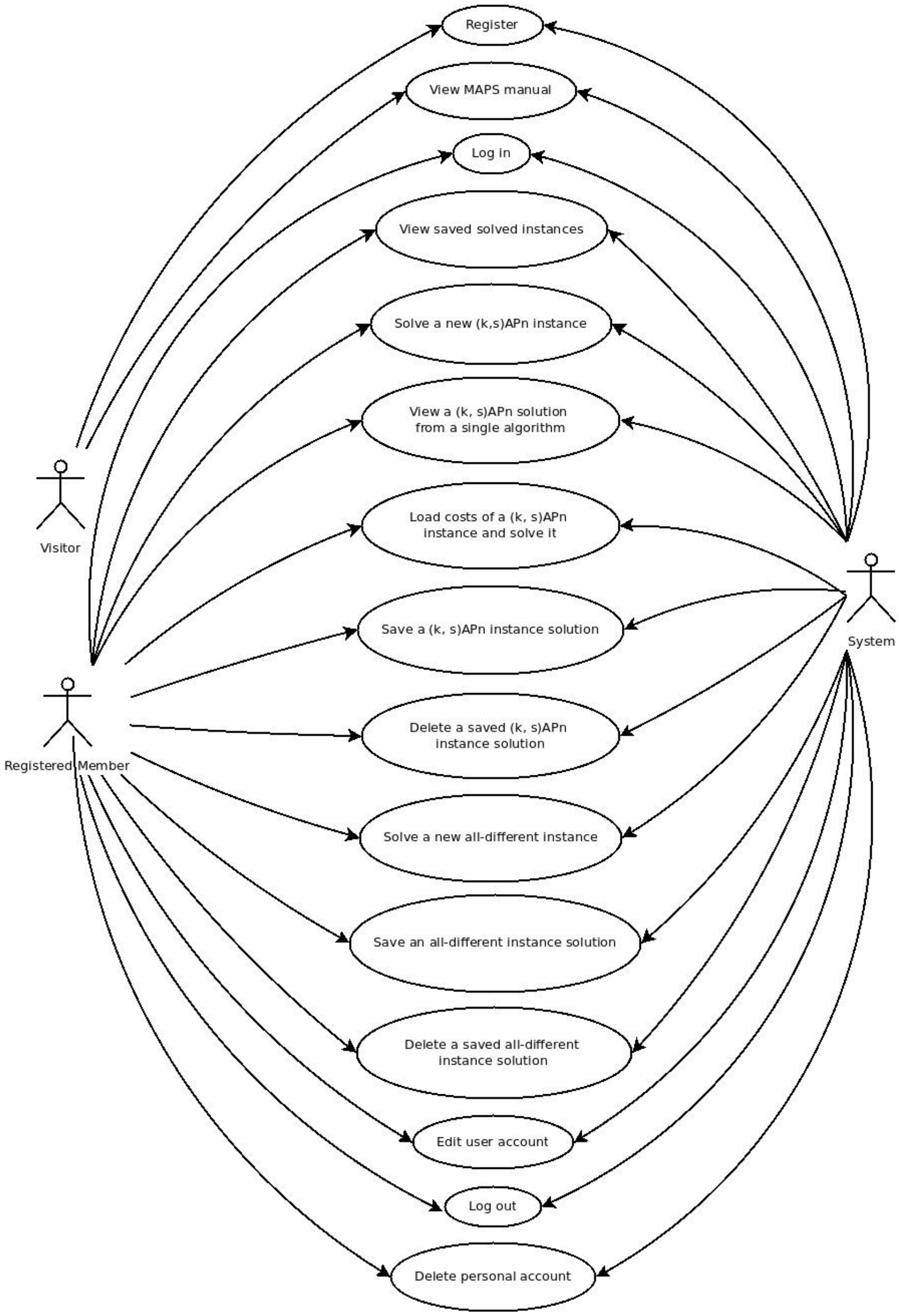


Figure 5.1: Main use case diagram

5.1.2 Non-functional requirements

The architecture of MAPS should also satisfy some non-functional requirements, presented in Table 5.2, which will ensure the normal operation of the system and the provision of a proper environment for the desired functionalities.

Table 5.2: Non-functional requirements

id	*Requirement*	*Description*
1	Storage	The system will contain a database where all user data can be stored, following a relational schema.
2	Back-up	The system should be supported by a back-up mechanism for the contents of the database.
3	Security	The system should be secured against sabotages arising from all types of hacking attacks.
4	Privacy	An authentication process will be required for accessing the functionalities offered by the system. Essential user data such as the password have to be encrypted before storing.
5	Scalability	The system should be able to handle the potentially increased number of users. Additionally, the system must be able to run on any of the major modern hardware platforms and operating systems. Specifically, it must run on Windows, Linux, Solaris or Mac-OS operating systems and any hardware architecture that is supported by these operating systems.
6	Availability	The system should ensure that users have always access to data and associated assets 24/7 with 99.9% reliability. This requirement entails stability in the presence of localized failure.
7	Usability	Easy to use. User documentation should not be necessary for ordinary tasks.
8	User Interface	Should give access to all system functionalities providing easy navigation through all features.
9	Interoperability	Individual components should be able to exchange information and use the information exchanged.

5.1.3 Data view

The following domain diagram (Figure 5.2) presents the main entities that build-up the required dataset of this system. Additionally, we include a short description of each entity.

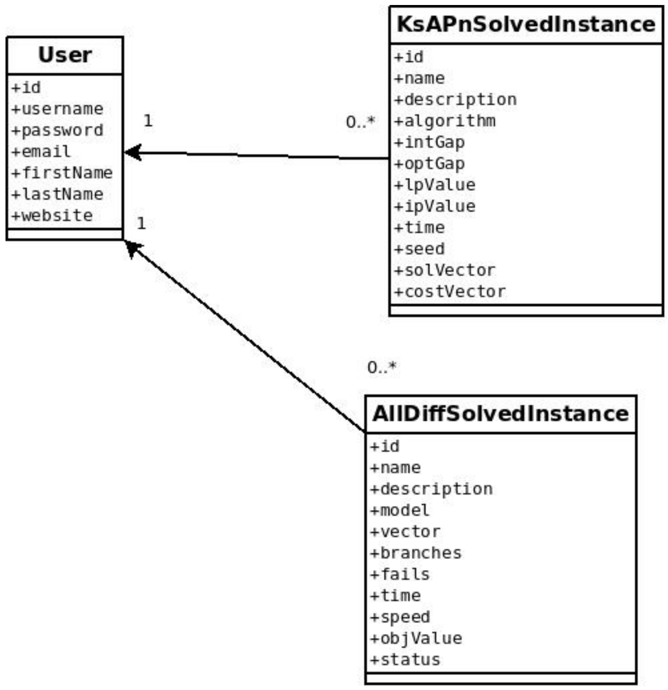


Figure 5.2: Domain diagram

User: This entity models every user within the system. Its attributes are an *id*, a *username*, a *password*, an email, a *first* and *last name* and a *website*.

KsAPnSolvedInstance: This entity models any $(k, s)AP_n$ solved instance. Objects of this class are solved instances saved by a user. Its attributes are an *id*, a *name*, a *description*, the *algorithm* with which the instance has been solved, the integrality gap (*intGap*), the optimality gap (*optGap*), the objective value of the linear relaxation (*lpValue*), the objective value of the integer solution (*ipValue*), the solution time, the *seed* with which the cost coefficients were pseudo-randomly generated, the solution vector (*solVector*) and the cost vector (*costVector*).

AllDiffSolvedInstance: This entity models any *all-different* solved instance. Objects of this class are solved instances saved by a user. Its attributes are an *id*, a *name*, a *description*, the generated ILOG *model*, the solution vector (*solVector*), the total branches (*branches*), the failures during the search for a solution (*fails*), the total time (*time*), the search speed in branches/second (*speed*), the objective value

($objValue$) and the solution status ($status$), i.e., an identifier showing if a solution has been found if the instance is infeasible or if during the solution process an unexpected error occurred.

Following the domain diagram, Figure 5.3 shows the generated entity-relationship diagram.

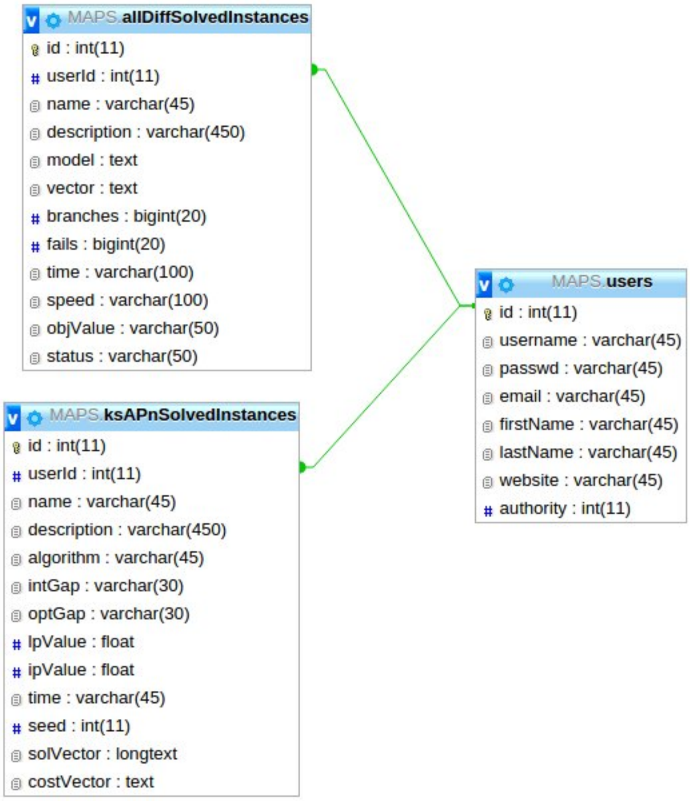


Figure 5.3: Entity-Relationship diagram

5.2 Algorithms

Following the analysis of requirements, this section lists the encoded algorithms that build-up the callable library and the core of this system for the $(k,s)AP_n$. Considering that there is much to be gained by exploiting the complementary strengths of approaches to optimization, Constraint Programming (CP) and Integer Programming (IP) methods among with heuristics and meta-heuristics are employed.

The main goal of this attempt is to have a solver that includes state-of-the-art encoded algorithms for the $(k,s)AP_n$. In this context, to enhance MAPS some standard greedy heuristics, some versions of a state-of-the-art heuristic for Mixed Integer

Programming (MIP) called Feasibility pump, and a tabu-search meta-heuristic for all classes of multi-index assignment problems, including axial and planar ones, are deployed. All these components were described in detail in Chapters 3 and 4, however we sustain some of their basic features here for self-containment of this chapter. Furthermore, four new versions of the feasibility-pump heuristic are deployed and tested, also not existing in the research literature. These new versions include problem specific cutting planes in this heuristic. This is the novelty of this idea, i.e. including exact optimization approaches into heuristics.

Preceding the development of the heuristics, a mechanism that performs constraint propagation has been developed that works for all classes of the $(k,s)AP_n$. Given the problem dimensions, i.e. parameters k, s and n, this mechanism builds an index of variables that exist in a constraint and vice versa. Additionally, this mechanism can support the development of any heuristic for the $(k,s)AP_n$ wishing to include constraint propagation.

Note again, that the aim of this section is not to describe in detail the employed algorithms. However, for completeness is provided a list with the encoded algorithms. Further details can be found in Chapters 3 and 4. Hence, the employed algorithms are the following:

- a constraint propagation mechanism that works for all classes of the $(k,s)AP_n$;
- Best-in greedy heuristic;
- Worst-out greedy heuristic;
- Basic version of Feasibility Pump, hereafter denoted as *Feasibility Pump v1.0*;
- Feasibility-Pump with constraint propagation, hereafter denoted as *Feasibility Pump v2.0*;
- Objective Feasibility-Pump, hereafter denoted as *Objective Feasibility Pump v1.0*;
- Objective Feasibility-Pump with constraint propagation, hereafter denoted as *Objective Feasibility Pump v2.0*;
- Basic version of Feasibility Pump with cutting planes, hereafter denoted as *Feasibility Pump v1.0 with cliques*;

- Feasibility Pump with constraint propagation and cutting planes, hereafter denoted as *Feasibility Pump v2.0 with cliques*;

- Objective Feasibility Pump with cutting planes, hereafter denoted as *Objective Feasibility Pump v1.0 with cliques*;

- Objective Feasibility Pump with constraint propagation and cutting planes, hereafter denoted as *Objective Feasibility Pump v2.0 with cliques*;

- Tabu-search meta-heuristic;

- a standard Branch and Bound (B&B) algorithm obtaining optimal solutions.

The inclusion of any new algorithm does not affect the data-model of the system or any functionality described previously. It only affects the user interfaces, where the new option of a new algorithm should be available on the respective screens.

5.3 Overview of screens

This section presents the overview of some core use cases in terms of system screens. By 'core' use cases we imply these that are related to solving an instance of te problem. Each use case is described in the respective subsection with screen-shots from the main and alternative flows. The screens of the remaining use cases that are listed in the previous section can be found in Appendix A.

5.3.1 Solve a new $(k, s)AP_n$ instance

Once the user has selected to solve a new $(k, s)AP_n$ instance he has to fill the dimensions of the instance, i.e.

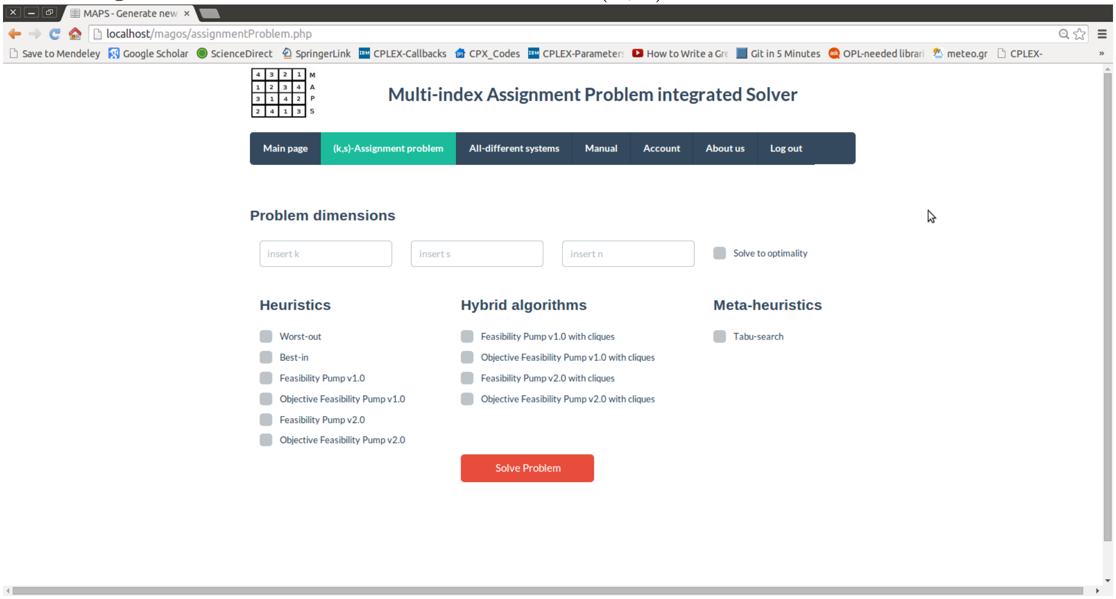
- parameter k: with value between 3 and 6;

- parameter s: with value between 1 and $k - 1$

- parameter n: this field can be filled either with a single value, or with multiple values (e.g. 4,5,7,9), or with a range of values (e.g. 5-10). Every value should be in the range $[2, 49]$.

The user must also select the algorithm or algorithms with which he wishes to solve the new instance. Note that for multiple values or a range of values for parameter n the user may

Figure 5.4: Use Case 4 - Solve a new $(k, s)AP_n$ instance: Main screen



- select only a single algorithm;
- select not to solve the instances to optimality.

If the given dimensions are not correct, the user is appropriately prompted to correct them. Once the user has given correct parameters of the instance he can view the results from the selected algorithms and compare. Furthermore, he can download the table of the results in LaTeX (tex), PDF and Comma Separated values (CSV) format.

Figure 5.5: Use Case 4 - Solve a new $(k, s)AP_n$ instance: Correct parameters

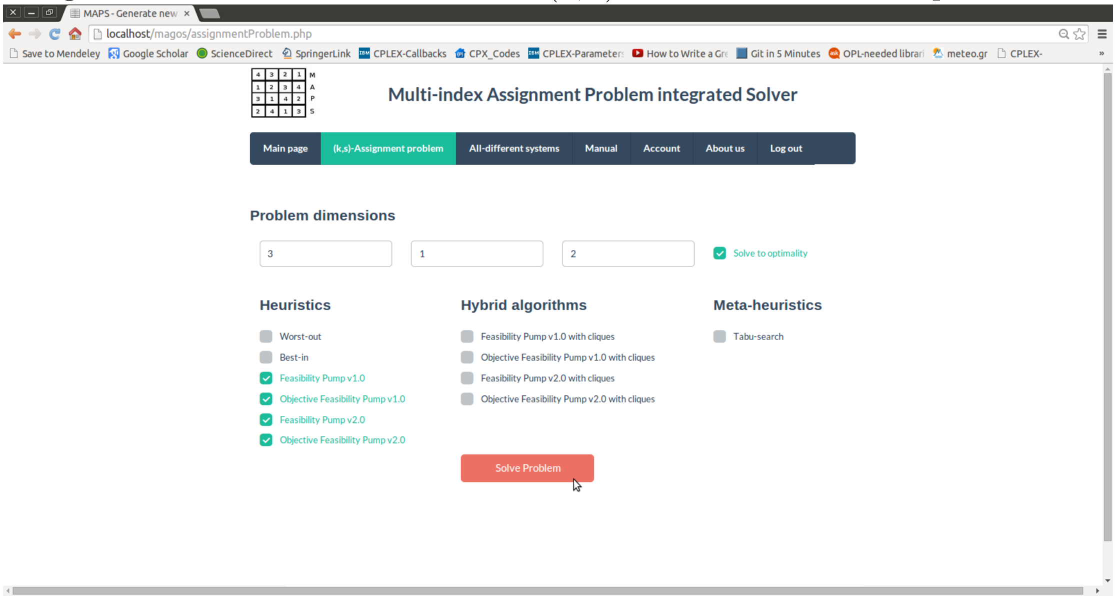


Figure 5.6: Use Case 4 - Solve a new $(k, s)AP_n$ instance: False parameters

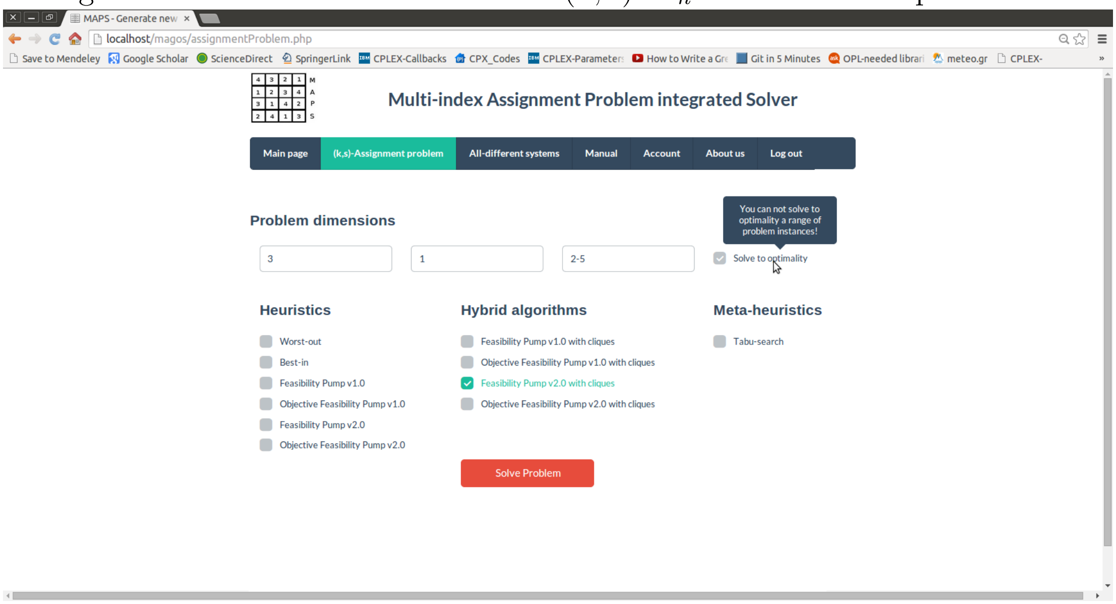
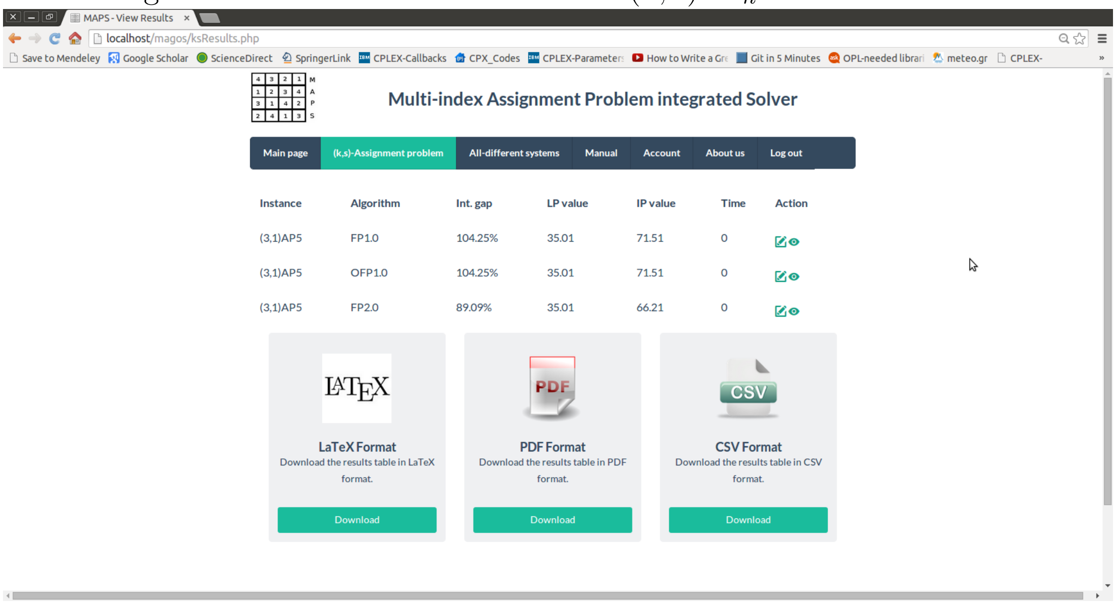


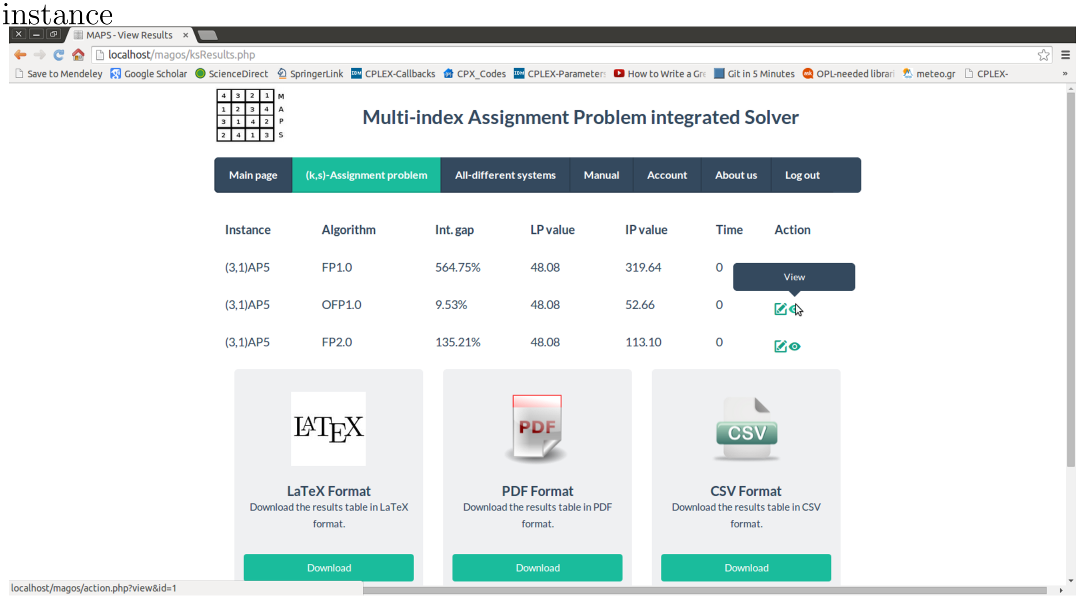
Figure 5.7: Use Case 4 - Solve a new $(k, s)AP_n$ instance: Results



5.3.2 View a $(k,s)AP_n$ solution obtained from a single algorithm

Once the user has selected from the table of the results which instance he wishes to see, he is transferred to the screen with the details of the solution.

Figure 5.8: Use Case 5 - View a $(k,s)AP_n$ solution from a single algorithm: Selection of instance



If the user wishes to see the solution vector he can do it by clicking on the 'Solution Vector' drop-down list. Additionally, he can download the cost vector in CSV format.

5.3.3 Load costs of a $(k,s)AP_n$ instance and solve it

Once the user has selected from the main menu to load the cost vector of a $(k,s)AP_n$ instance and solve it, he is prompted to enter the dimensions of the instance and load a CSV file with the cost vector. Multiple values or a range of values for parameter n are not permitted.

5.3.4 Solve a new *all-different* instance

Once the user has selected to solve a new *all-different* system, he first has to enter the number of the *all-different* constraints, the number of the variables and the number of the domains in which the variables take their values. Note that only numerical values (positive) are accepted. Additionally, there have to exist at least two variables, otherwise the *all-different* instance has no meaning.

Figure 5.9: Use Case 5 - View a $(k, s)AP_n$ solution from a single algorithm: View details

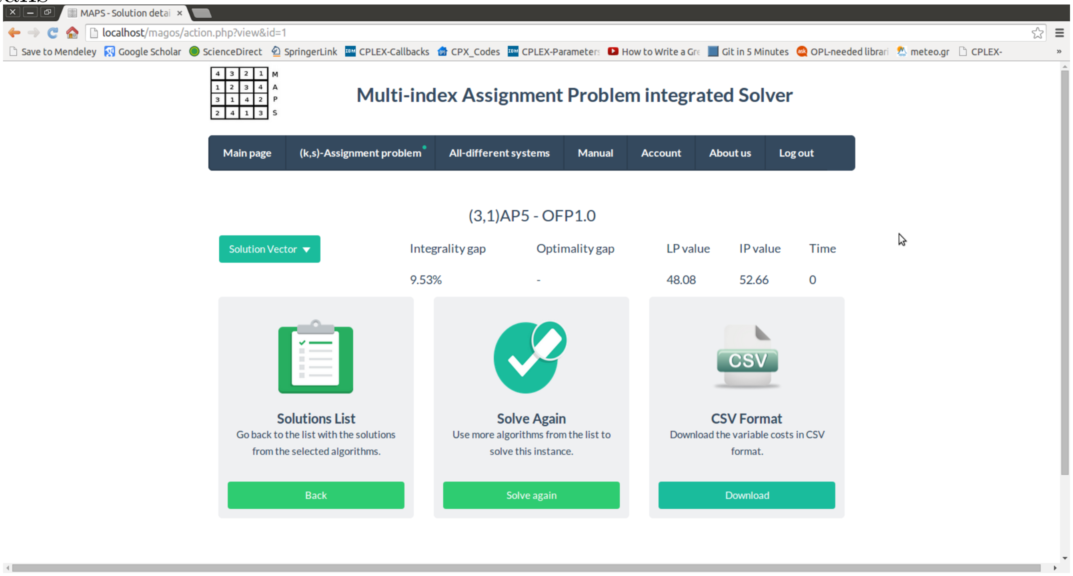


Figure 5.10: Use Case 5 - View a $(k, s)AP_n$ solution from a single algorithm: View cost vector

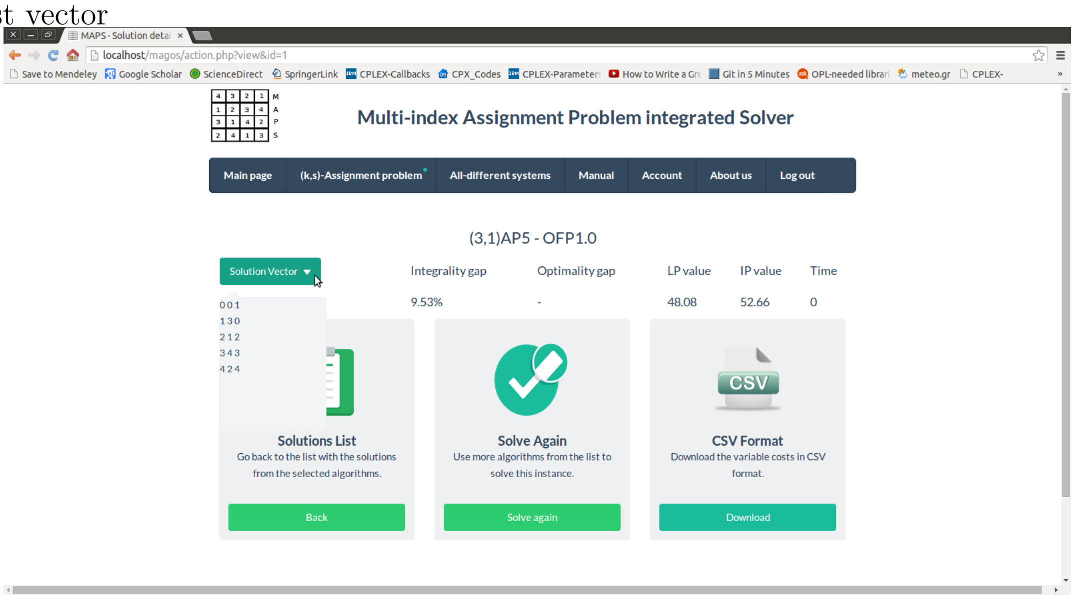


Figure 5.11: Use Case 6 - Load costs of a $(k, s)AP_n$ instance and solve it : Main screen

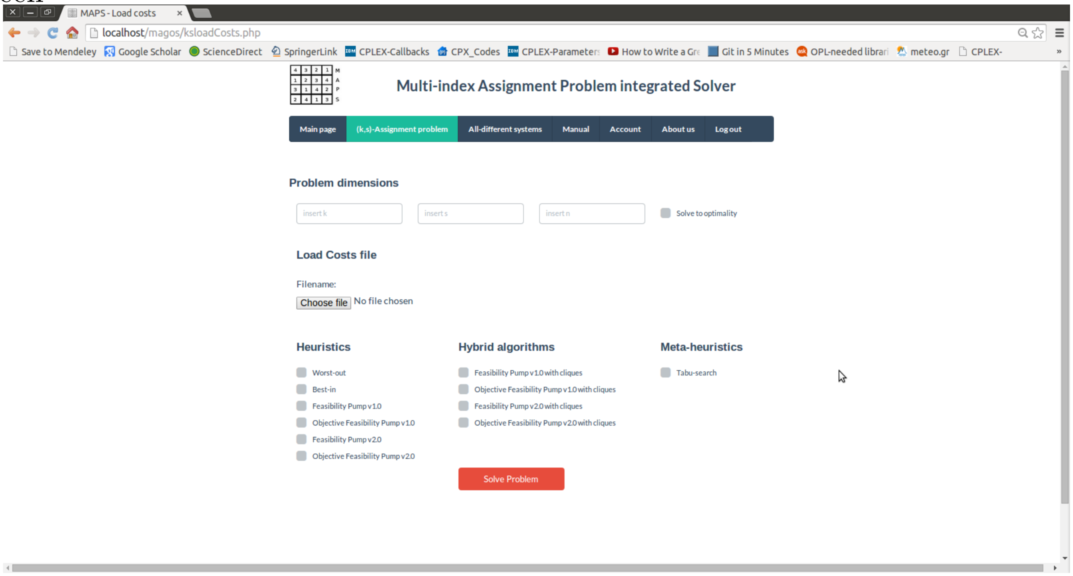


Figure 5.12: Use Case 6 - Load costs of a $(k, s)AP_n$ instance and solve it : Correct entries example

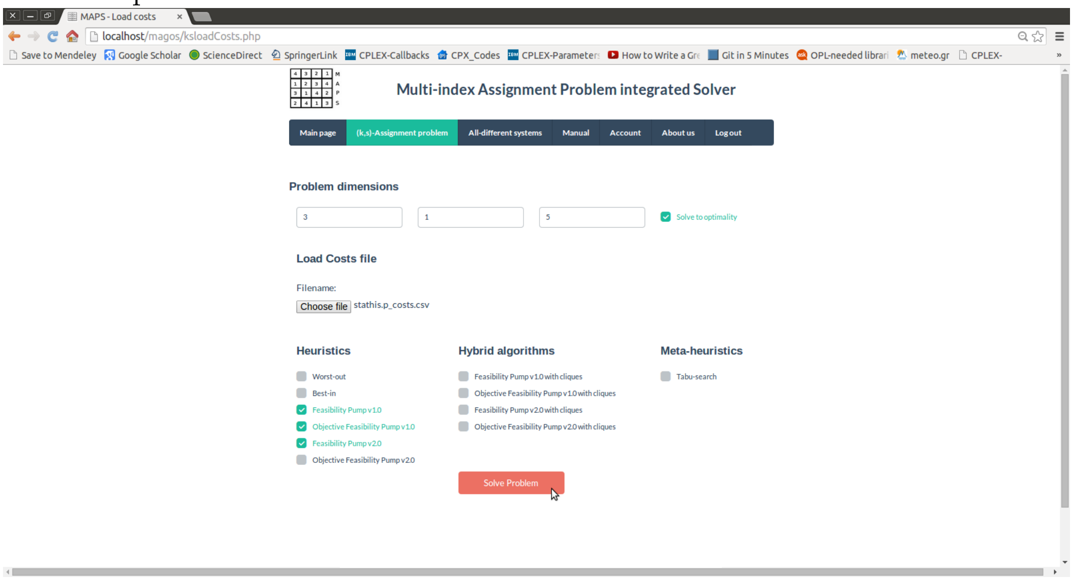


Figure 5.13: Use case 9 - Solve a new *all-different* instance: Initial screen

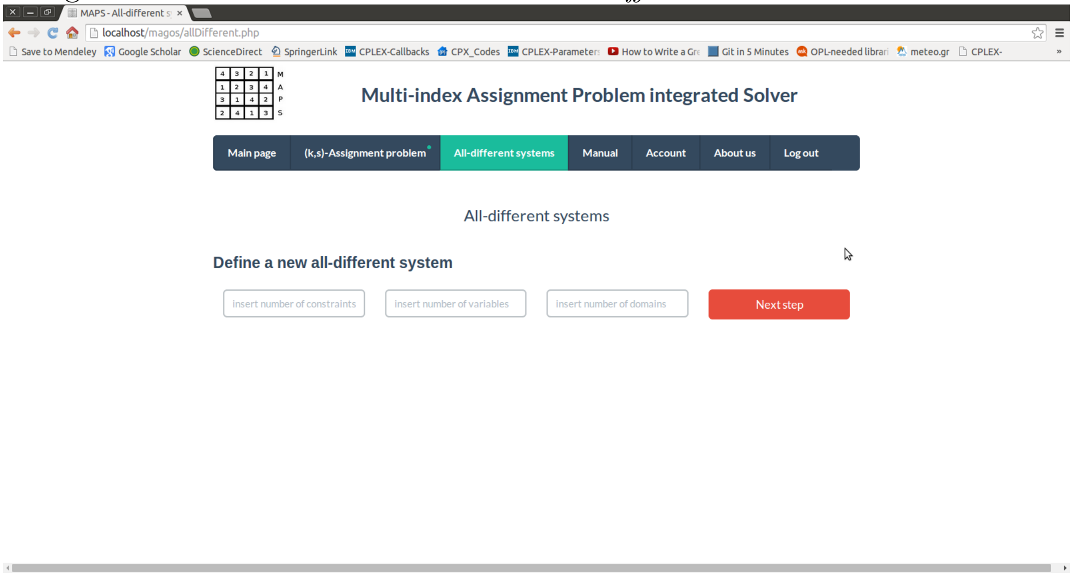


Figure 5.14: Use case 9 - Solve a new *all-different* instance: Inserting values for constraints, variables and domains

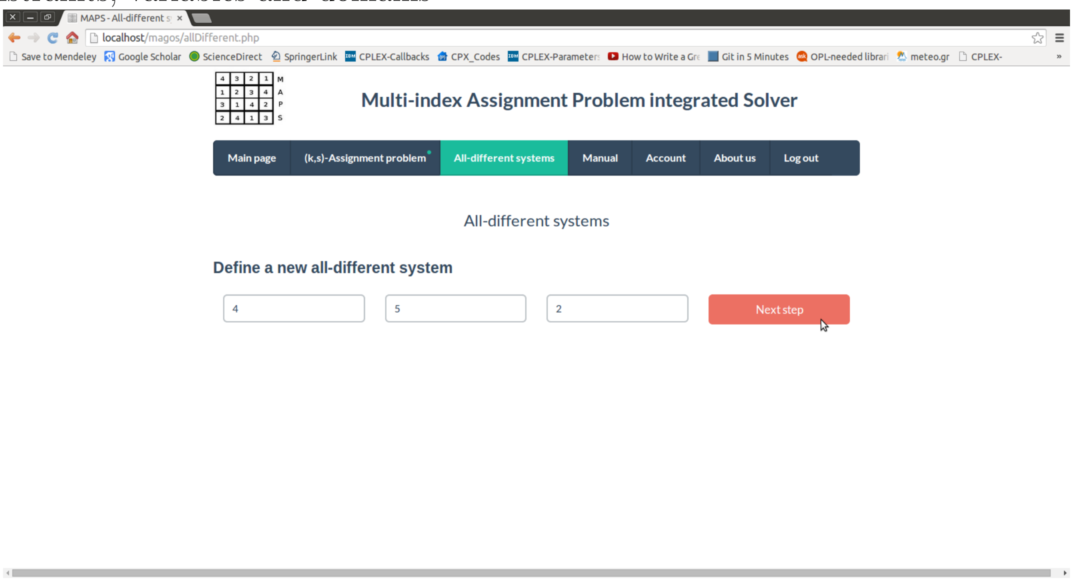
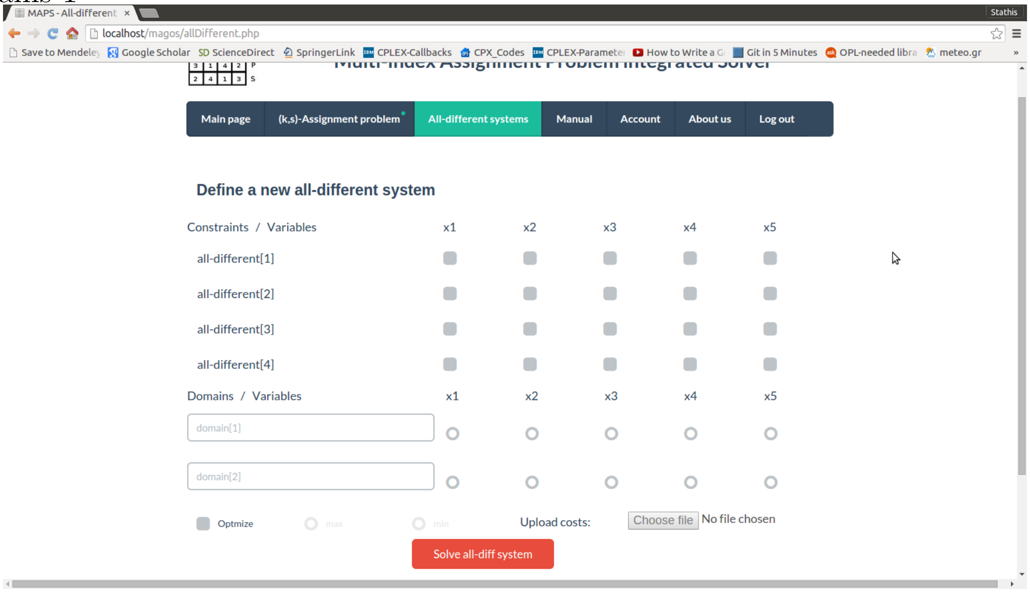


Figure 5.15: Use case 9 - Solve a new *all-different* instance: Defining variables and domains 1



After entering the dimensions of the instance, the user has to define which variables exist in which constraint, and from which domain the variables take their values. Note that any *all-different* constraint must have at least two variables. Every variable must take its value from a single domain and each domain should at least include one variable. Also note that the domains can either be formed as ranges, e.g. 4-10, or have discrete values e.g. 1,6,8,19. Any other form of a domain is not acceptable.

The user can also solve to optimality an *all-different* system by enabling the 'Optimize' selection. Once this check-box is enabled, the user can select either to maximize or minimize the objective value of the *all-different* system. Note here that a file with costs, in CSV format can also be uploaded. If no costs-file is uploaded, then the system will automatically generate random integer values for cost coefficients within the range $[0, 1000]$. An example for the format of the costs file is given in Figure 5.20.

Finally, the user can view the statistics of the solution of the *all-different* system and the solution vector. Note here that the system generates an Optimization Programming Language (OPL) script, right after the user has defined an *all-different* system. Additionally, he can download the generated OPL script by clicking in the download button or solve a new instance of an *all-different* system.

To conclude, let us highlight that integrated solvers coupled by such a DSS are not reported in the literature. Additionally, we consider such interfaces an important step for the adoption of such methods in practical situations, while the versatility of

Figure 5.16: Use case 9 - Solve a new *all-different* instance: Defining variables and domains

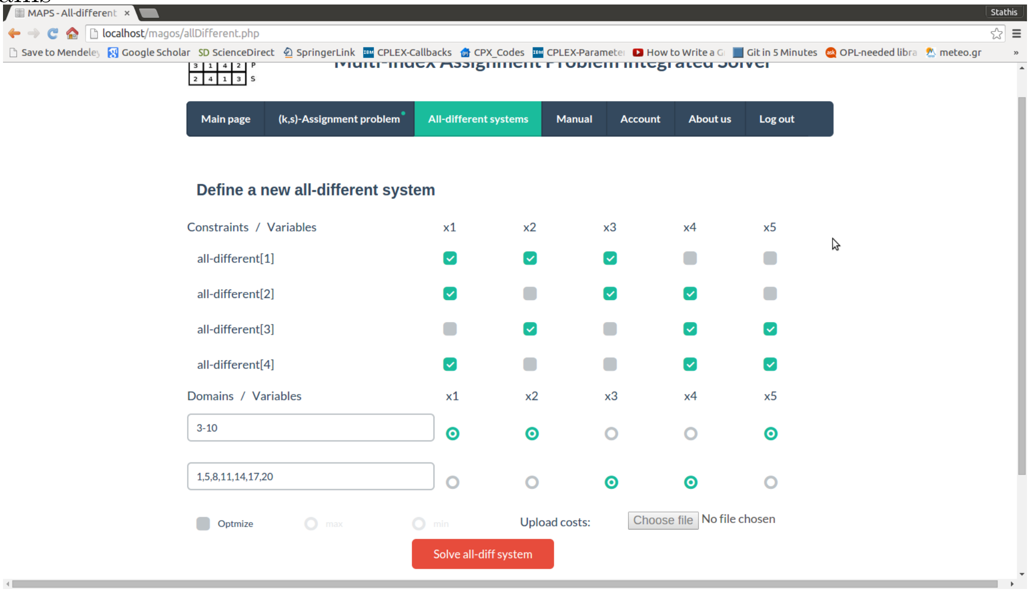


Figure 5.17: Use case 9 - Optimize an *all-different* instance

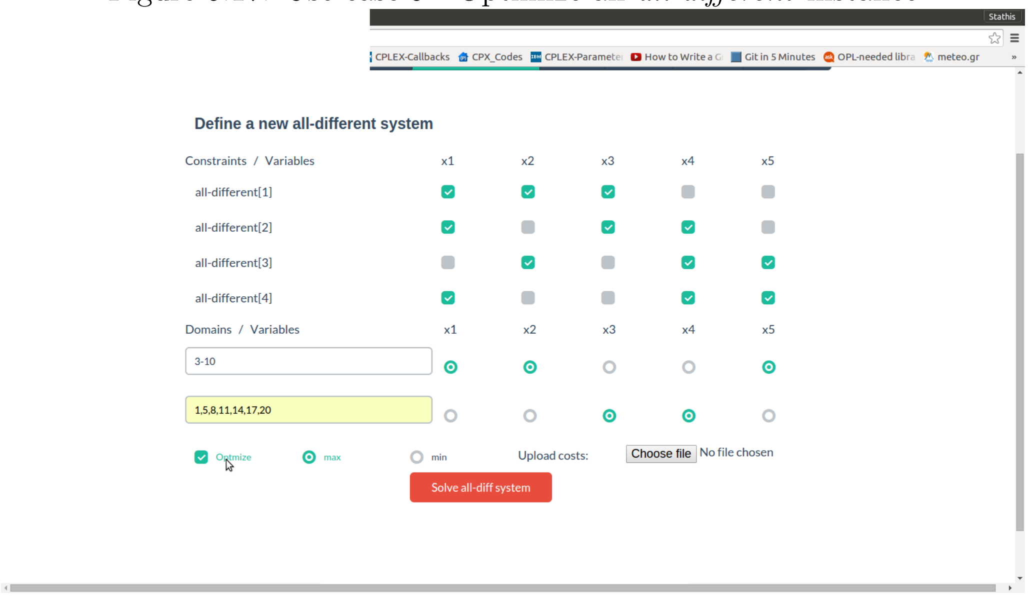


Figure 5.18: Use case 9 - Minimizing an *all-different* instance

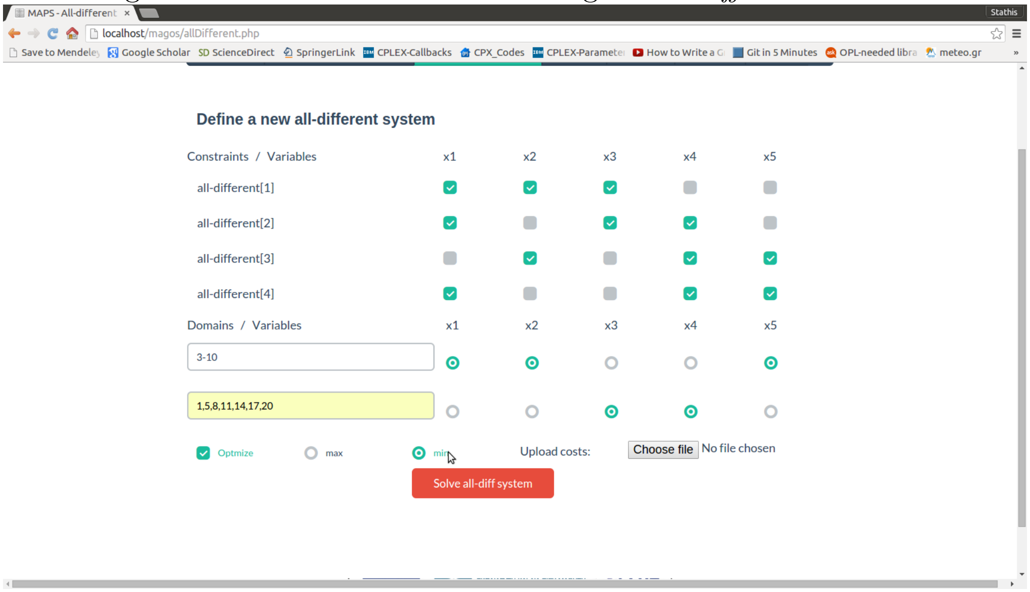


Figure 5.19: Use case 9 - Load the costs-file of an *all-different* instance

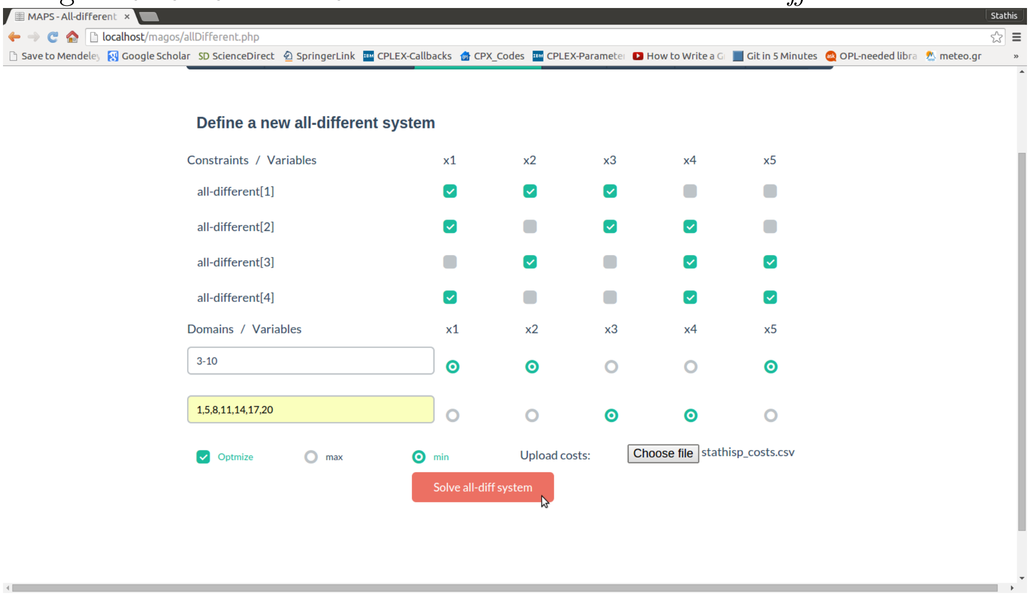


Figure 5.20: Use case 9 - *All-different* costs-file.csv example

```
variable,cost
x1, 27.10
x2, 2.72
x3, 3.45
x4, 56.89
x5, 9.70
```

Figure 5.21: Use case 9 - Solve a new *all-different* instance: Viewing solution statistics

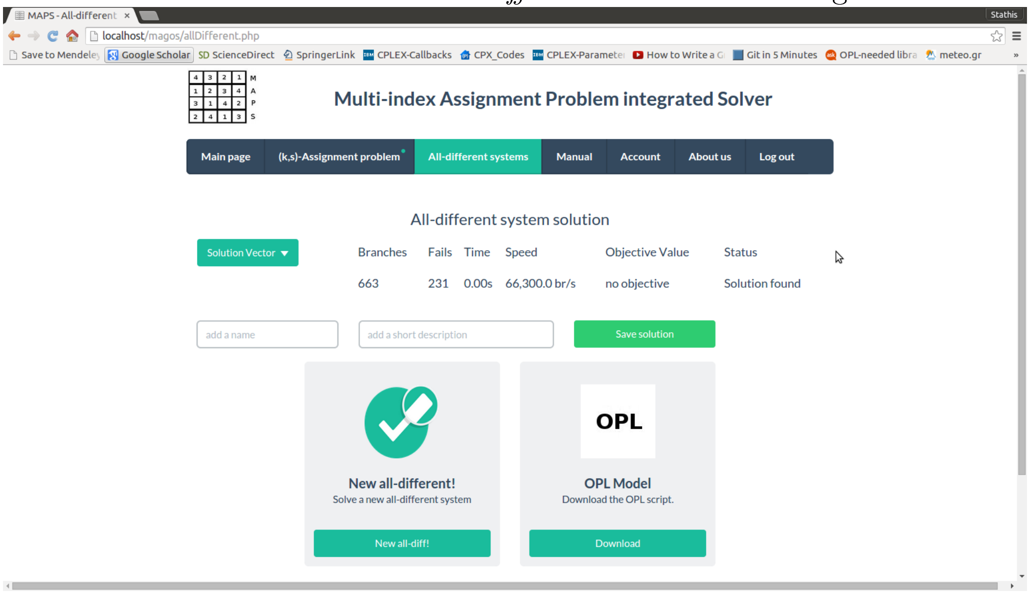


Figure 5.22: Use case 9 - Solve a new *all-different* instance: Viewing solution vector

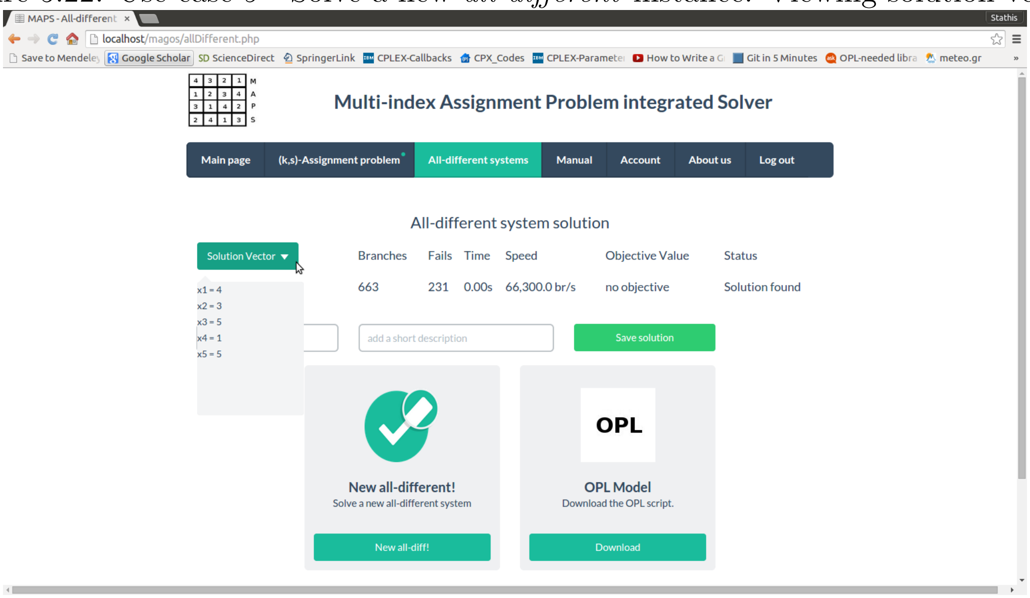
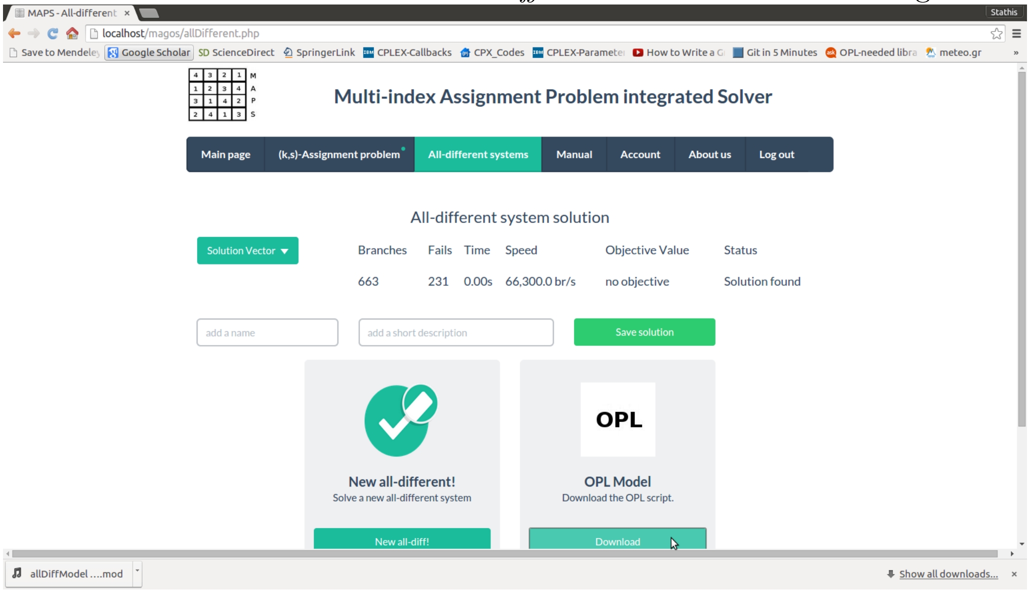


Figure 5.23: Use case 9 - Solve a new *all-different* instance: Downloading OPL script



integrated methods yields a breadth of parameterization for which such a DSS could be of help. Therefore the work presented here could be of both practical and academic use beyond the scope of the optimization problem underlying it.

Chapter 6

Decision support for energy-aware production scheduling

Modern manufacturing companies are forced to become energy-aware under the pressure of energy costs, legislation and consumers' environmental awareness. Production scheduling remains a critical decision making process, although demanding in computational terms and sensitive on data availability and credibility. Hence, incorporating energy-related criteria in production scheduling has become more important.

This chapter describes an energy-aware production scheduling decision support system (DSS), composed by an Iterated Local Search algorithm that offers hierarchical optimization over multiple scheduling criteria and a generic yet concise data model whose entities are extracted from the literature and actual user requirements. The results of embedding this DSS in an integrated system used by two textile manufacturers show that it indeed supports efficiently energy-aware production scheduling.

The remainder of this chapter goes as follows; In Section 6.1 we provide the motivation and the context of the problem at hand. Section 6.2 provides the research background motivating this study. Section 6.3 presents the production scheduling problem followed by the user and data requirements and the algorithmic scheme. Section 6.4 discusses the application of the proposed DSS along with implementation issues and Section 6.5 presents the evaluation and the benefits measured in a real context.

This is a joint work with P. Repoussis, I. Mourtos and C.D. Tarantillis, a significant result of a European research project, namely ARTISAN (GA no. 287993), that is published in the Decision Support Systems journal [90].

6.1 Motivation

Public and industry concerns over energy efficiency and environmental sustainability have grown considerably over the last decade. Particularly in the industrial sector, energy efficiency becomes an even more important pillar since it accounts for more than one third of energy consumption worldwide [108, 83] of which the manufacturing sector accounts for about 73%. Despite these numbers, industrial practices towards energy-efficient manufacturing have traditionally been viewed as a ‘cost of business’, and positioned as the voluntary responsibility of companies. Nowadays, this perception is changing as stricter legislation, industrial standards and energy costs require that companies not only adopt a strategy of minimal compliance, but also treat such a strategy as a catalyst for sustainable practices. Furthermore, consumers are becoming increasingly aware of whether the product they purchase comes from a sustainable source and is produced through eco-friendly methods that, ideally, guarantee minimum environmental impact [74]. Betraying the consumer’s confidence can damage a company and its brand image [35]. Altogether, legal compliance, energy costs and customers’ increasing ecological awareness [21] are driving companies towards measurable energy efficiency improvements, thus motivating in broad terms this research effort.

Although the manufacturing sector has advanced towards energy efficiency, the economic benefits arising from energy efficiency have not been fully exploited [78]. Both academic and business studies indicate that there is an “energy efficiency gap” and highlight that there are strong barriers which impede energy-efficient manufacturing. Systems supporting relevant decisions can help minimize these barriers by monitoring energy consumption and carbon emissions, thus pinpointing areas for savings [28] as a basis for energy-based optimization and intelligent decision making [104]. Several enterprise systems have been enhanced by energy management capabilities, although typically limited to energy monitoring, analysis and reporting [21]. These Energy Management Systems (EMS) do not support management decisions in a coherent way due to a lack of integration of information from shop-floor to top-floor [105]. Apparently, apart from the gap between industrial needs and the academic literature [21, 81], there is also a gap between the solutions available and the support of sophisticated decision making such as production scheduling [109].

Indeed, scheduling is an important decision-making process in manufacturing that drills down to deciding on

(i) *which* tasks to execute,

(ii) *where* to process the production tasks and in which sequence and

(iii) *when* to execute the production tasks.

Typically these decisions are strongly coupled, thus ideally taken simultaneously [53]. Due to the complexity and the increasing production volumes, such decisions cannot be addressed without an automated optimization support. This functionality is typically considered part of a Manufacturing Execution System (MES) [53] and is normally supported by an Enterprise Resource Planning (ERP) system through data exchange. Note that, apart from reducing monetary cost, good production schedules can reduce the environmental load through energy demand reduction [53]. Still, due to its nature, scheduling remains computationally complex and data intensive, because it requires not only production data but also the availability status of resources, e.g., machines, even in real-time. In fact, the more complex a scheduling system is, the more information to be collected and managed [53], and the more competitive the algorithms to be used for obtaining valid schedules of good quality.

This study focuses on energy-aware flexible shop scheduling production environments and is motivated by the scarce optimization algorithms (and the limited background on the data required) for energy-aware support at the shop floor. The adopted production scheduling framework is as generic as possible and takes into account various operational aspects and utility or resource constraints. In particular, the machines of each process step share and consume one or more 'resources', e.g., electricity. The constraints can be imposed in one or more process steps, and the maximum levels per resource may also vary across the planning horizon (e.g., flexible electricity consumption pricing). The goal is to find the optimum schedules for different scenarios of peak energy utility consumption and to also minimize indirect energy consumption.

An Iterated Local Search (ILS) is used introducing new compound moves and an adaptive perturbation mechanism. It also offers optimization using either energy or temporal scheduling criteria. On one hand, the energy-related criteria include the total energy consumed by machines during production and idle time (direct energy) along with the energy consumed by subsidiary equipment (indirect energy). On the other hand, the temporal scheduling criteria, namely makespan, total flow time and total machine idle time, can be optimized hierarchically over all possible combinations. Although temporal, these objectives also target both direct and indirect energy consumptions. That is, minimizing the makespan increases total throughput and reduces the number of shifts, thus reducing the indirect energy consumption (e.g., heating); minimizing total flow time reduces total production time, hence total direct energy consumption (e.g., machine gas consumption); and minimizing total idle time reduces the energy consumed by machines in idle mode. Hence, hierarchical optimization over these temporal criteria covers all major aspects of energy consumption, whereas individual minimization of a single temporal objective is insufficient as also shown by the pilot use and experimentation. Furthermore, by imposing time-varying restrictions on the energy consumption of machines, the optimization algorithm provides schedules that alleviate energy peaks by distributing more 'uniformly' the consumption across time. That is, energy-awareness amounts not only to energy-driven optimization objectives, but also to energy-consumption constraints.

To the best of our knowledge, an explicit description of the data entities supporting energy-aware production scheduling is at the moment not available, although scheduling-related entities have been presented early enough (e.g., object-oriented modelling [88]). To identify these data entities, a set of user requirements is formulated as acquired from the academic literature and validated within the textile manufacturing domain. The algorithm is aligned with these entities and remains operative even if certain data are missing or are less credible. Thus, in this study is proposed a DSS for energy-aware production scheduling composed by an algorithm and a generic, concise and validated data model.

Overall, this work contributes to decision support for energy-efficient manufacturing by

(a) a metaheuristic algorithm that hierarchically optimizes flexible shop scheduling problems,

(b) a set of data requirements in the form of a data model,

(c) the integrated deployment of the above as a web-service and

(d) the evaluation of the proposed DSS in real settings and the tangible benefits obtained by its use.

6.2 Research background

It is being increasingly required that energy-intensive industries need tools and methods to optimize production processes that take into consideration energy-related criteria [21]. Several approaches for achieving this have been proposed. Indicatively, accurate changes on resources or processes can effectively reduce energy usage [60].

Prior studies have been conducted to evaluate the energy-burden of different processes [98], which are used as a roadmap to identify alternatives in the production process. However, these changes impose a significant initial investment, because they may involve radical changes in the manufacturing process [16]. An alternative, less costly approach, is the modification of production settings, such as the temperature of the machines. Although several studies focus on such practical changes [16], it seems that these approaches are usually related to specific processes that can lead to new problems, e.g. lower product quality [81]. As simple and well-suited for their domain these approaches may appear, they tend to be static. Hence the need for dynamic adaptations to production conditions in future factories [60].

The optimization of energy use via production scheduling has received particular focus in the past decade. One early attempt formulates a multi-objective optimization problem for an electroplating line [102]. Another study has shown that up to 65% of the energy consumption comes from non-productive machine modes (e.g., stand-by or idling) [31]. Embedding energy aspects into scheduling can be tackled by both exact and heuristic approaches. Exact mathematical programming based methods can obtain optimal solutions but require considerable time given that the job-shop and flow-shop scheduling problems are NP-hard [80]. Indeed, by integrating energy constraints in a set of 100 instances, Artigues et al. [10] show that the solution time for a mixed integer programming model is high. This is also outlined by Fang et al. [38] by testing an exact method against heuristic algorithms to minimize the makespan, the peak consumption and the carbon footprint of the production process. This trade-off between the solution quality and the required time is usually resolved by the requirements of the application domain. Regarding the manufacturing domain, fast re-scheduling is often required in the presence of unpredictable events such as machinery malfunctions.

Metaheuristic methods trade optimality so as to provide high quality feasible solutions within reasonable time, while their current state-of-the-art for production scheduling algorithms includes genetic and hybrid evolutionary algorithms [81, 95]. Indicatively, Shrouf et al. [99] focus on scheduling on just a single machine, taking into consideration the variable energy cost during daytime hours and use a genetic algorithm to provide feasible schedules of good quality. Rager et al. [95] use a combination of genetic and memetic algorithms to acquire schedules minimizing the energy demand of multiple parallel machines by first splitting production orders into operations that have constant energy demand. This results in a schedule defined by the underlying ‘identical parallel machine’ environment and the resource-levelling objective. Notably, the approach of [95] has been tested in the textile domain particularly at the dying stage of yarns.

The problem addressed here is more generic: it includes various operational constraints (e.g., non-identical machines, sequence-dependent set-up times) and is applicable with minor modifications to almost any shop scheduling environment. Also, the proposed algorithm seems highly competitive and incorporates novel local search components and re-start mechanisms to escape from local optima. Moreover, this algorithm performs, apart from minimization of the direct and indirect energy as a single objective, hierarchical optimization over three different criteria, i.e., makespan, total flow time and total idle time. Note here that hierarchical optimization in manufacturing typically covers two criteria [38, 95, 81] and has never targeted total idle time. However, an approach that covers three completely different criteria appears in the human resources allocation domain [23]. This is rather surprising since minimizing total idle time (where the idle time of a machine is possibly weighted by its stand-by consumption) indeed minimizes machine energy use in non-production mode; thus, if minimized together with makespan in some hierarchical fashion, it does encompass energy-awareness in scheduling decisions. Another innovative feature of this approach is that it deals effectively with several time-varying resource constraints, thus encompassing the sharing of one or more energy resources (e.g., electricity, gas, steam) by multiple machines.

To identify the entities that build up the data model, user requirements for production scheduling are collected from the literature [53] and validated from requirements as extracted from the textile industry, i.e., from two textile manufacturers that are the end-users of the proposed DSS plus several other textile producers (mostly SMEs). Then, the framework of [109] is utilized regarding the integration of data from enterprise systems (ERP, MES, EMS), thus offering a coherent data model for energy-aware production scheduling. This is of no surprise, since the proposed algorithm is insufficient if not relying on appropriate data, and vice-versa: the data entities to be incorporated in the data model are selected exactly because they are required for supporting scheduling decisions.

6.3 Problem definition, data requirements and algorithm

In this section are presented the modules that build-up the proposed DSS, namely the scheduling problem with resource constraints (Section 6.3.1), the data requirements (Section 6.3.2) and the components of the proposed ILS algorithm.(Section 6.3.3).

6.3.1 Energy-aware production scheduling problem with resource constraints

Production scheduling can be defined as the allocation of available production resources over time to perform a series of activities. Suppose that a set of unrelated parallel machines M_j ($j = 1, ..., m$) have to process a set of production orders or jobs J_i ($i = 1, ..., n$). Each job i has a release date, a due date and consists of a k_i number of operations $O_{i1}, ..., O_{ik_i}$, while each operation O_{ij} is associated with a subset of machines μ_{ij} and a (normally machine-dependent) processing time p_{ij}. At any time, each job can be processed by at most one machine and each machine can process at most one job. It takes each job a different amount of time to be processed by each machine.

Once a job is processed on a machine, it cannot be interrupted before completion. Additionally, whenever a machine finishes the processing of an operation, a set-up (changeover) time occurs before processing the next operation. The length of the set-up can be sequence dependent (i.e., the set-up depends on the job just completed and on the one about to be started) and/or machine dependent (with or without a predefined frequency). Overall, the objective is to find a sequence for the processing of the jobs in the machines so that a given objective function is optimized. A schedule is for each job an allocation of one or more time intervals to one or more machines. To that end, the associated scheduling problem is to find a schedule that satisfies a given set of precedence restrictions among the operations of each job and respects its due and release dates.

The above production scheduling problem can be formally depicted as flexible multi-processor job-shop scheduling problem with unrelated parallel machines, due dates, and set-up times that depend on both the job sequence and the machine. There is significant research work on production scheduling problems with parallel machines and makespan minimization [89], but significantly fewer for unrelated parallel and sequence-dependent set-up times. Note that minimizing the makespan on two identical machines is NP-hard [56]. For mixed integer mathematical formulations as well as exact and metaheuristic solution approaches see Rocha et al. [96].

Additionally, machine availabilities, shifts and resource constraints are taken into consideration in this framework. Regarding the former, each machine is coupled with a number of qualified employees. There are 3 different shifts per day, while the person-shift allocation plan is known in advance. Based on this qualification matrix and the available personnel per shift, one can determine the machine availability during the planning horizon.

Furthermore, it is assumed that machines consume one or more energy utilities, e.g., electricity, gas and/or steam. The utility consumption is directly related with the time elapsed, and may also depend on the machine mode, i.e., start-up, clean-up, stand-by and production mode. The time spent during the first three modes is considered as 'idle time', which practically is a necessary, yet non-productive time. The amount of energy consumed by machines is considered as *direct* irrespectively of whether the machine is in idle or production mode; i.e., 'direct productive' and 'direct idle' energy are considered as different, but both types are consumed directly by production machinery in the shop floor. However, there may also exist additional subsidiary energy-consuming equipment (e.g., air-conditions) in a shop floor related to the production process. As described in [109], although these amounts of consumed energy are *indirect*, they are important and should be taken into account. Energy-aware production scheduling is expected to minimize both direct and indirect energy consumptions, while the restriction of energy peaks can be achieved by adding resource consumption constraints.

In this study, two types of objectives are taken into consideration. The first type is a single energy-related criterion and considers the total direct and indirect energy. In more detail, the energy consumed during production, $E_{production}$, and idle time, E_{idle}, (i.e., direct energy) along with the indirect nergy consumed by subsidiary equipment (e.g., air-conditions), $E_{indirect}$ are taken into account. The goal is to minimize the sum of these three energy consumptions. The second type includes three temporal-scheduling criteria considered and treated as single objectives in a lexicographic order (hierarchical tri-objective), namely makespan (C_{max}), total flow time (F_t), and total idle time D_t. The minimization of the makespan (completion time of the last job in the schedule) is expected to maximize the utilization of machines [38], to reduce the required shifts of the machine personnel and to increase the throughput. Thus, it reduces mainly the indirect energy consumed by subsidiary equipment. The minimization of the total flow time is related to the time the machines spend in production mode, hence it can be seen as minimization of the direct energy consumption. To the contrary, the minimization of the machine idle time is related to both direct and indirect energy consumptions.

In the remainder of this chapter, the notation $f|g|h$ is used to indicate the hierarchy of the temporal scheduling objectives. For example, the $C_{max}|F_t|D_t$ indicates that C_{max} is the (dominant) primary objective, F_t is the secondary objective and D_t is the last objective in consideration.

6.3.2 Data requirements & information flows

Production scheduling requires several data entities, such as the machines, the employees, the production orders and the status of the machines. Introducing energy-aware related criteria in production scheduling implies embedding energy-related data in the data model. Indeed, manufacturing-related data entities appeared pretty early in the literature [39], yet without incorporating energy-related data. Furthermore, production management data entities can also be found in simulation-related literature [97] or object-oriented modelling, again disregarding energy-related data [88].

The entities proposed here can be classified into three information flows (see also [109]):

- *Operational flow*: information regarding specific orders, the types of products and the materials used for each product.

- *Production process flow*: information regarding the production processes, such as machines, process steps, production and product tracking.

- *Energy consumption flow*: information regarding energy consumption that is either measured by energy sensors or specified by traditional audits.

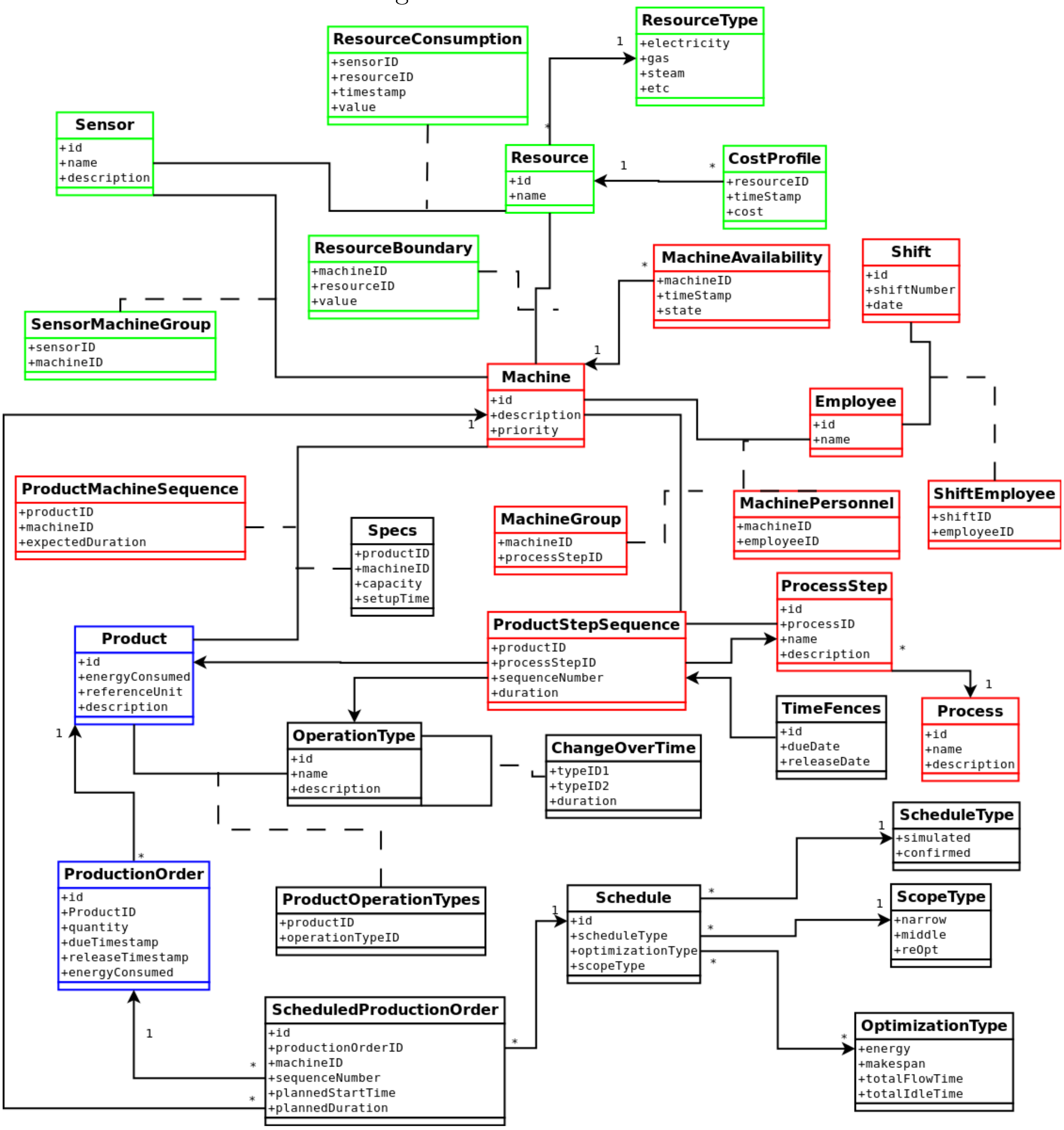
Based on the description of the problem (see Section 6.3.1) and the above flows, the proposed data model is shown in Figure 6.1. Blue color is used for operational, red for production process and green for energy consumption entities; black color distinguishes operational entities that are used only for scheduling. The description of these entities is as follows;

- **Employee**: Models each employee that works in the enterprise.

- **MachinePersonnel**: Displays which employee can work on which machine.

- **Shift**: Models the shifts of the employees within a facility.

Figure 6.1: Data model



- **ShiftEmployee**: Displays which employee is assigned to which shift.
- **Resource**: Models the consumable resources, e.g., electrical energy, gas, steam.
- **CostProfile**: The time-varying cost profile per resource across time.
- **ResourceBoundary**: The maximum resource consumption per machine.
- **Sensor**: Models each metering device measuring the consumption of a resource.
- **SensorMachineGroup**: Maps a group of machines to a sensor.
- **ResourceConsumption**: Models the direct energy consumption.
- **Machine**: Models each machine that exists in a production environment.
- **MachineAvailability**: Models the availability of a machine.
- **Specs**: Models the specifications of a machine for a product.
- **Process**: Models a process of the manufacturing procedure.
- **ProcessStep**: Models a part of a process.
- **Product**: Models the products.
- **ProductStepSequence**: Models the process step sequence for a product.
- **TimeFences**: The time fences of a product in a process step.
- **ProductMachineSequence**: Models the machine sequence for a product.
- **OperationType**: Models the different operation types that may occur during the processing of a product across a machine sequence.
- **ProductOperationType**: Models the sequence of operation types across machines for a product.
- **ChangeOverTime**: Models the changes over time, of all possible combinations of OperationTypes.
- **ProductionOrder**: Models the production orders that have been placed.
- **Schedule**: Models the schedules that may be produced.

- **ScheduledProductionOrder**: Maps production orders over different possible schedules.

- **ScopeType**: An enumeration defining every possible scope type for a schedule within the system.

- **OptimizationType**: An enumeration defining every possible optimization type for a schedule.

- **ScheduleType**: An enumeration defining the two different types of schedule.

As expected, the entities in this model reflect data entities in existing ERPs, MESs and EMSs. The novelty of this model is the combination of these flows that support energy-aware production scheduling. This is a prerequisite for applying these algorithmic components that support production scheduling. Furthermore, this requires interfaces that allow the data exchange between

- the proposed DSS and existing systems;

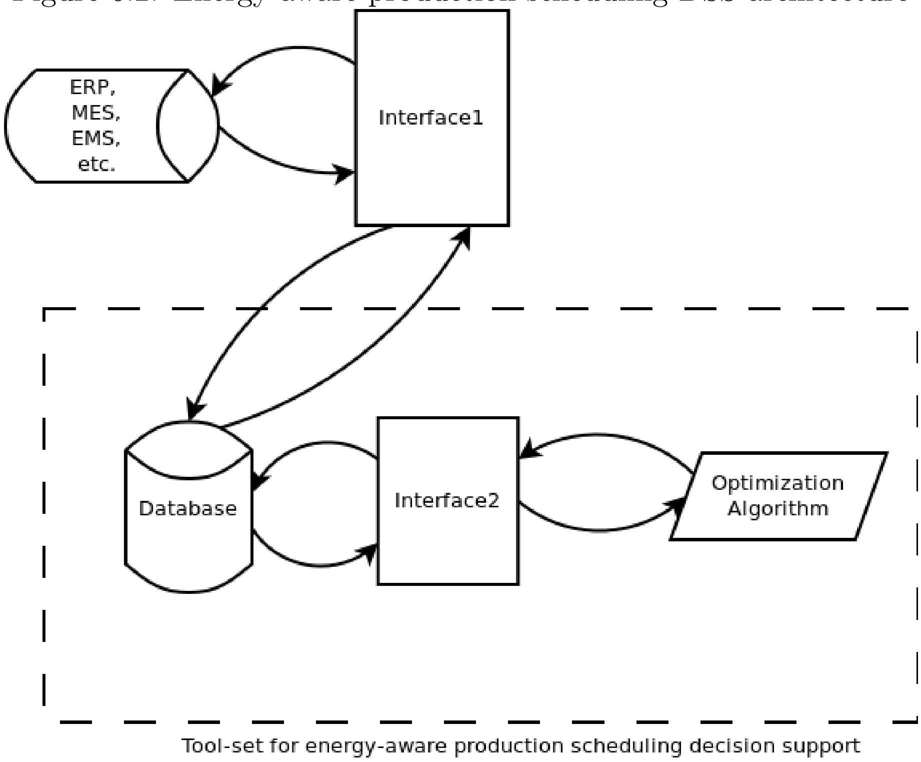
- the elements comprising the DSS,

as shown in Figure 6.2. This is in simple terms the high-level architecture of the proposed DSS. The interaction with external systems is performed via 'Interface 1' (this reflects a series of interfaces, one per enterprise system), while the database with the algorithmic components interacts through 'Interface 2'. Note here that the latter interface can also deal with issues of data consistency. This prevents the algorithmic components from having erratic input data.

6.3.3 Iterated local search method

An ILS method has been developed for solving the hierarchical production scheduling problem with utility constraints. ILS is a perturbation-based multi-restart local search metaheuristic algorithm introduced originally by [68], in which the initial solutions for local search are generated by perturbing local optimum solutions obtained during previous searches. The implementation originates from a starting solution s, local search is initially applied until a local optimum solution s^* is found. At this point, a random perturbation is applied that leads to an intermediate state s'. Local search is triggered starting from s' until a local optimum solution $s^{*\prime}$ is reached. If $s^{*\prime}$ improves s^*, then it becomes the next solution for local search; otherwise, the procedure is

Figure 6.2: Energy-aware production scheduling DSS architecture



restarted from a new starting solution s. The oscillation between perturbation and local search is repeated for a number of iterations.

We adopt a sequential insertion-based construction scheme to generate starting heuristic solutions. At each iteration, one operation is selected and added in the permutation of a machine. The rule followed is to schedule the operation as early as possible. All available machines and all feasible insertion positions (with respect to machine availabilities, release and due dates, precedence relationships and resource constraints) for each operation are examined. The main effort is to schedule the operation that performs best with respect to the hierarchy of objectives. Based on this greedy criterion, a restricted candidate list of positions at the available machines is generated for each operation, and one position from this list is selected randomly. Note that every iteration the actual schedule partially constructed solution is updated.

An iterative improvement local search scheme is employed. In particular, the solution neighbourhoods are generated by applying the relocate and exchange operators [111] on a representation based on the permutation of operations on the machines. Equal selection probability is assumed for each operator, while a best admissible strategy is followed for moving in the solution space. For the evaluation of the neighbourhoods, a lexicographic search scheme has been developed that considers all feasible inter and intra machine move combinations. The main effort of this scheme is to expedite the process by avoiding unnecessary feasibility checks.

Lastly, a 'ruin-and-recreate' mechanism is applied for perturbation. In particular, a number of jobs is randomly removed from the schedule and the greedy randomized construction scheme described above is applied to reschedule them. The number of the rescheduled jobs is determined by a self-adapted length that is regulated based on the search progress.

6.4 Energy-aware production scheduling in the textile industry

In this section are described the implementation details of the proposed DSS as part of the ARTISAN system [1], whose aim is the reduction of the energy use in textile manufacturers. The ARTISAN system integrates data from several enterprise systems and from real-time energy monitoring to assist the enterprises in reducing the energy consumption through monitoring and allocating energy consumption per production order and also through energy-aware production scheduling. This has been achieved by deploying and validating the proposed DSS for energy-aware production scheduling. The whole ARTISAN system, including the proposed DSS, has been installed and tested in two industrial partners, a small-to-medium enterprise (SME) focusing on production of yarns and a large-size enterprise (LSE) with a vertical production line.

6.4.1 Decision support and production scheduling in the textile industry

The primary scope of the proposed DSS is to reduce the direct and indirect energy consumption by delivering scheduling scenarios and master production plans that significantly improve all time-related factors of the underlying production processes (i.e., total running, deadhead and set-up times of different sub-processes and machines) under constrained resources.

Energy audits on the premises of the industrial users have shown that the most energy-consuming process in the textile manufacturing chain is the finishing mill. Hence, this particular process is to be optimized. In ARTISAN, the finishing mill is modelled as a multi-step (multi-stage) production flow shop facility, where a production step is made up of one or more related and/or unrelated parallel machines (or production lines / machine groups). The term job refers to a 'production order' and a product is also called an 'article'. The speeds, capacities and programs/settings of the machines are known and they may depend - among others - on the articles and the process quantities (lot size). In addition, set-up times and costs are incurred (changeover time/cost) when machines have to be reconfigured and/or cleaned between operations on different articles. The length of the set-up can be sequence dependent (i.e., the set-up depends on the job just completed and on the one about to be started) and/or machine dependent (with or without a predefined frequency).

Each machine directly consumes one or more limited resources, namely electricity, gas, steam, water and compressed air, that should not exceed certain thresholds per certain groups of machines or for the entire process. Steam is produced by a continuous steam production unit, while water is heated by a gas-powered combined heat and power unit. Both of them have a predefined capacity limitation. Regarding electricity, cost profiles as well as maximum capacities for different hours per day are provided. Additionally, limitations may also occur in terms of drainage, rinse water and emissions. Furthermore, there is indirect energy consumption from auxiliary energy conservation installations (e.g., air-conditioning facilities, lighting, heating, etc.) that they are related to each machine or each process step(s). This production environment fits perfectly with the job-shop problem described in Section 6.3.1.

Regarding the optimization criteria, the total energy consumed during machine idle time has been pointed out as important both in the related literature [31] and by the experts (production engineers, floor managers, energy auditors) of the textile manufacturing domain in the context of the ARTISAN research project. This partly relates to the fact that machines cannot be easily shut down and then restarted, in order to avoid energy consumption while producing nothing, and partly to the fact that the energy consumption of several machines is comparatively large even if they are idle. However, minimizing only the total idle time may increase the makespan or vice-versa, although what a manufacturer seeks is the minimization of total direct and indirect energy consumed. Consequently, apart from a technical novelty, supporting the hierarchical minimization of the three aforementioned temporal-scheduling criteria is also a mandatory requirement.

6.4.2 User requirements and system functionality

Apart from the problem characteristics and the operational aspects, various user requirements are taken into consideration. Although these requirements have been collected from the textile manufacturing domain, they are also identified as basic user requirements from the academic literature [53]. A production scheduling software should provide, schedules for short-term periods or for a specific group of machines and production orders and master schedules for longer periods of time that affect every process in the production floor. It is important that the user reviews such schedules and selects among several "simulated" schedules the one to be implemented in the shop floor. For each schedule, the user wishes to alter provisionally the factory environment settings, e.g., the availability of a machine. When unexpected events occur, such as machinery malfunctions, re-optimizing the master schedule is necessary. Last, schedules are expected to be provided in a reasonable amount of time and presented in Gantt charts.

The proposed DSS (as part of the ARTISAN system) provides two services supporting the user to generate master production scheduling scenarios. These two services are titled "Resource Constraint Shop Floor Scheduling at the process level" (narrow scope) and "Resource Constraint Multiprocessor Shop Floor Scheduling" (middle scope). Apart from the scope (single and multiple process steps), a major is that the second takes into account cross-processing resource limitations and capacity thresholds across the process steps. The third service, "Reactive Shop Floor Scheduling" (middle scope) aims at assisting the user to responsively adjust the planned production schedule due to the occurrence of expected or less expected (e.g., machine breakdown) disturbing events. Its primary scope is the on-demand re-optimization of the current production schedule as new information arrives. Apart from ordinary static information, this function exploits all available time-related (article tracking) and energy consumption data of the processes and takes into account time profiles of the planned energy consumptions for each production phase (i.e., set-up, production, and cleaning) of a machine (or a process step).

6.4.3 Implementation details

The successful industrial application of the proposed DSS depends on resolving many practical issues when embedding this DSS in an actual system, such as ease of use, data availability, application development and maintenance.

Regarding the ease of use, after consulting with the end-users, following the waterfall model, the work-flow of the proposed services has taken its shape. Figures 6.3, 6.4 and 6.5 are the service blueprints of these services along with some indicative screen-shots. The scenario of use of these services is the following; after the user selects what kind of scheduling is to be performed (Resource Constraint Shop Floor Scheduling, Multiprocessor Shop Floor Scheduling or Reactive Shop Floor Scheduling), it is important to identify the scheduling horizon, that is the time-span of the optimization. For the defined horizon, the user should be able to see and verify the production orders that have due dates within this horizon. At any point, editing the production configuration is also important, e.g., excluding a machine from a particular schedule or changing the specifications of an article on a machine. This essentially allows the user to test several scenarios for the job floor, being able to include aspects like machine maintenance that are not explicitly formulated as part of the optimization problem. Right before producing a feasible schedule, the user has to define the optimization criteria which the scheduling algorithm tries to minimize, either one or multiple in a hierarchical order. Once the algorithm provides the schedule, the user can see it as a Gantt-chart and choose among a list with other simulated or confirmed schedules, for the same planning horizon, whether the schedule in hand is "confirmed" (it is the schedule to be executed) or "simulated" (i.e., one more schedule scenario in the pool of simulated ones).

Data availability implies that the requested pieces of information are available and reliable. The integration of energy-monitoring aspects in a factory environment is a new requirement, hence it is expected that energy-related data may be absent. In fact, the smaller industrial user did not have an EMS and only recently started using metering equipment for electricity consumption. Beyond that, tracking of production orders on machines is not available in real-time for this user and has been inserted using historical data. Energy-constraints are set over machines and not over production orders. Furthermore, the required time for a machine to change from one operation (e.g. dyeing blue) to another (e.g. dyeing red) is not available (for both users) but have been manually inserted.

The integration of the DSS with the ARTISAN system and the existing systems (e.g., MES), implies the implementation of interfaces enabling the data exchange between these systems. Apart from that, these interfaces ensure that changes on the DSS will not affect the proper function of individual components. This is an important aspect regarding the maintenance of the DSS.

6.5 Impact and benefit on industrial users

This section presents the benefits from the use of the proposed DSS in textile manufacturing. Sections 6.5.1 and 6.5.2 present the structure of the production lines and the specific scheduling requirements of two industrial users, namely an LSE (User A) and an SME (User B). Additionally the hierarchy preferences over the optimization criteria is discussed. Next, Section 6.5.3 contains various computational experiments

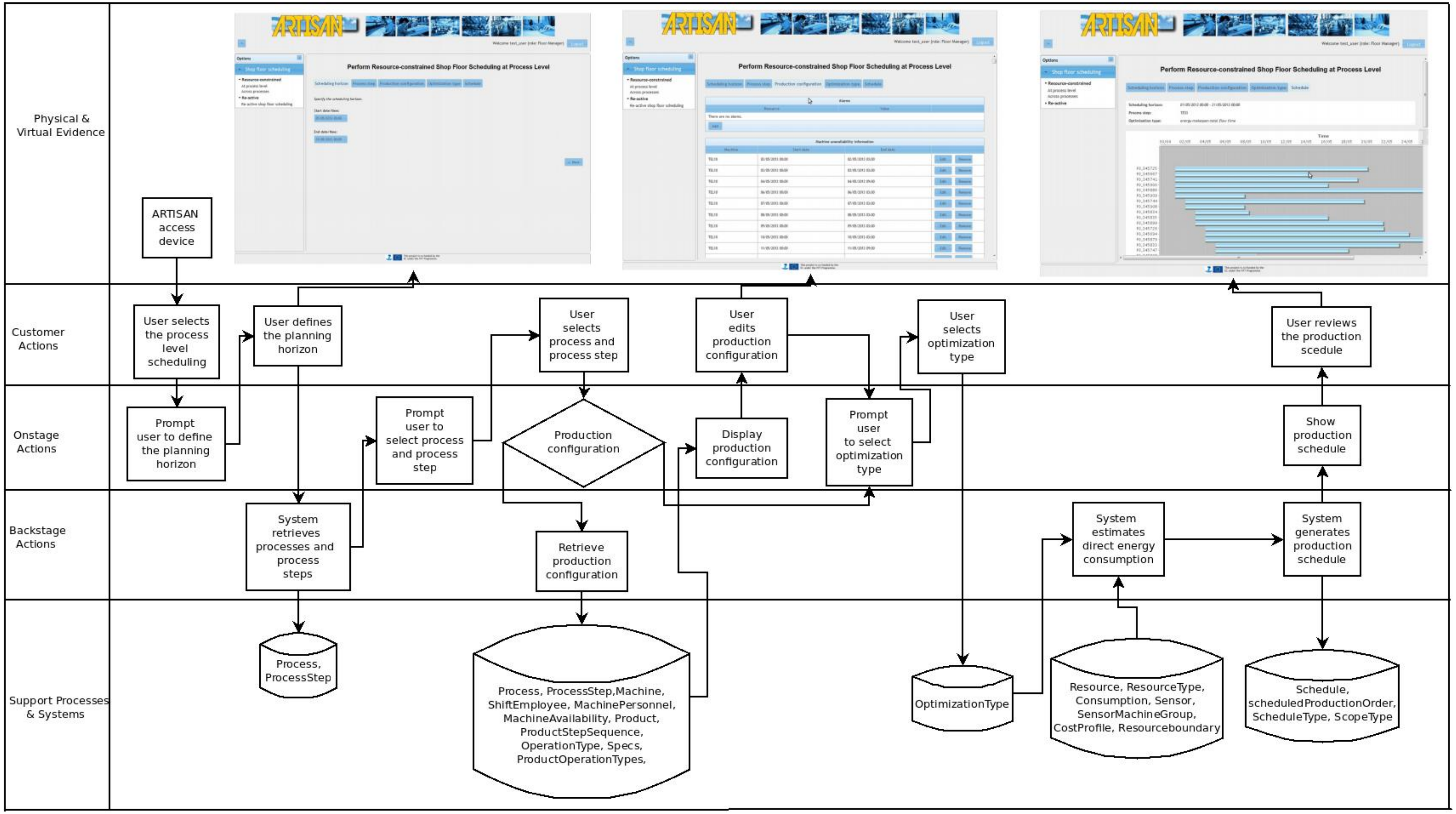


Figure 6.3: Resource-constrained shop floor scheduling at the process level

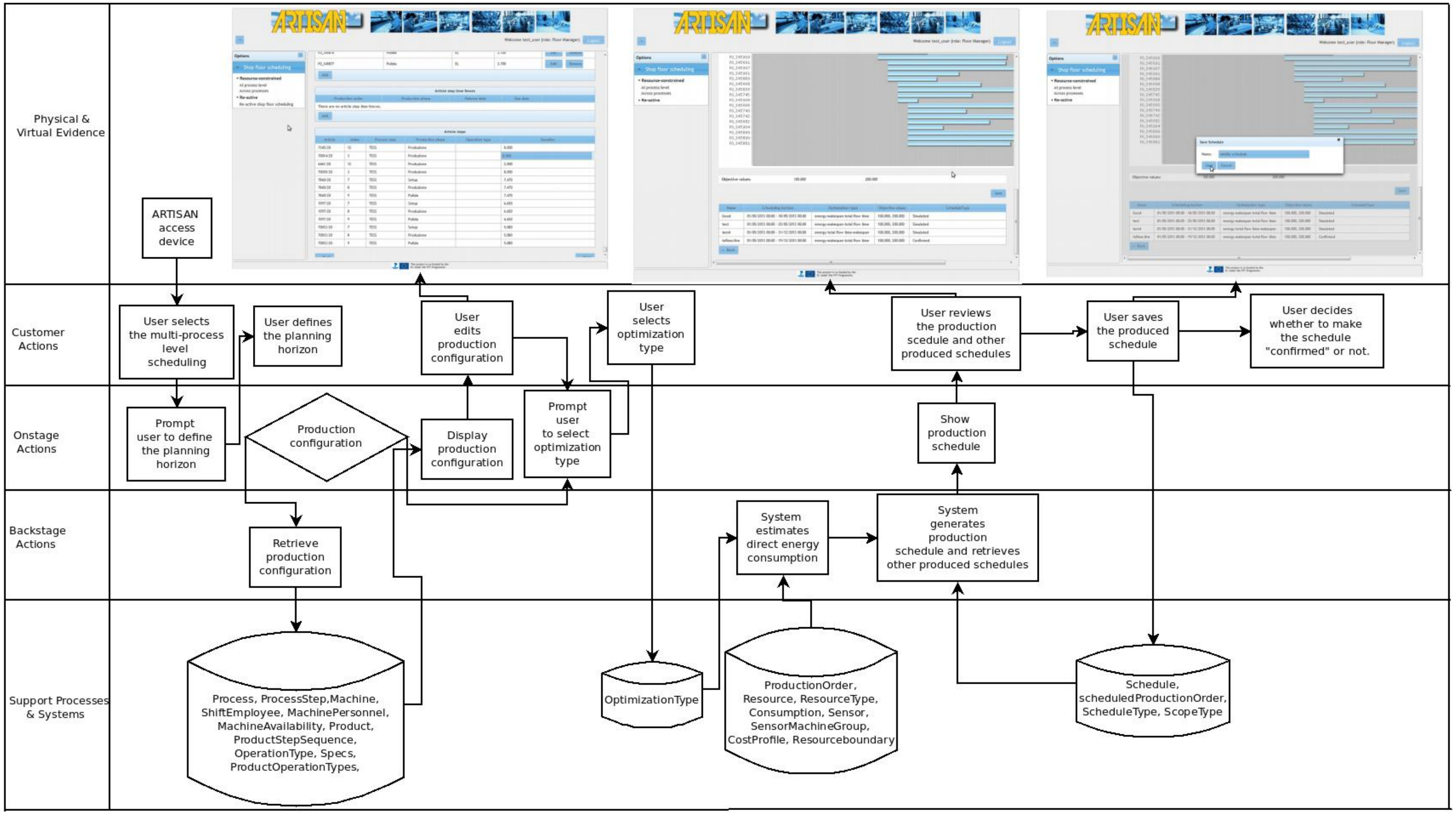


Figure 6.4: Resource-constrained multiprocessor shop floor scheduling

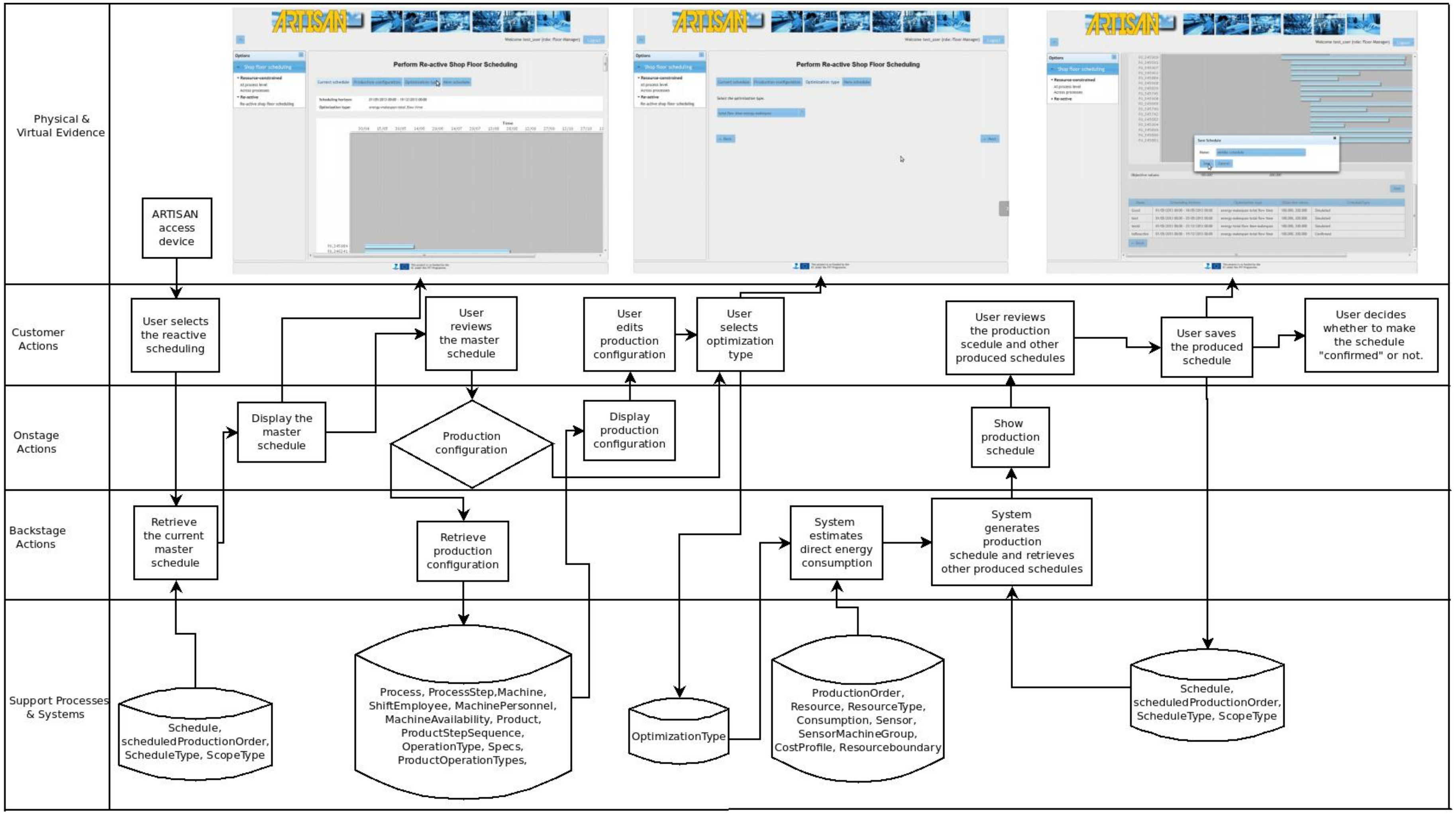


Figure 6.5: Reactive shop floor scheduling

and the reduction of energy consumption as calculated by both users. Section 6.5.4 concludes with a qualitative evaluation.

6.5.1 Large-size enterprise - Industrial User A

For User A focus is given on the processes related to the finishing department, whose key characteristic is that most machines run on high temperatures. Therefore, the heating-up and the cooling-down consumptions are significant. Therefore, it is common practice that many machines work in standby mode for long periods of time, i.e., all week except weekends. All optimization scenarios are performed assuming a mid-term planning horizon (more than a month).

A data set containing 642 production orders on 122 articles that have to go through 14 process steps with 38 machines in total (in all 14 process steps) is examined. It is worth highlighting that this problem size is considered as large-scale in the literature. No restrictions are imposed regarding the machine availability and no shortages are considered regarding the machine operators across the shifts. The machine capacity is the main bottleneck. User A prioritises the optimization criteria as $C_{max}|D_t|F_t$ so as to favour more the reduction of shifts (and thus indirect energy consumption) and then the standby-consumption; however, this user is also interested in the hierarchy $D_t|C_{max}|F_t$.

6.5.2 Small-to-medium enterprise - Industrial User B

The production in User B has fragmented production volumes, hence it becomes difficult to maintain a continuous production schedule. In particular, the small production lots cause frequent stops of machineries for setting-up the new production orders. Therefore, the machine idle times can be significant but the total flow time is also important. To that end, Use B selects the optimization hierarchy $D_t|F_t|C_{max}$.

A data-set containing 33 production orders on 20 articles, each passing through 24 process steps and a total of 29 machines is examined. The main restriction is represented by the operators dedicated to warp change (shift) and loom preparation. They are highly specialized, therefore the availability of the machine is strictly linked to this last rule. All optimization scenarios are performed assuming a thirty day planning horizon. Hence, this is a small-sized data set over a long horizon.

Table 6.1: Results for monthly production schedules with different optimization priorities

Month	**Hierarchy**	C_{max}	F_t	D_t
M1	$F_t \| C_{max} \| D_t$	8803	297780	122346
	$D_t \| C_{max} \| F_t$	8803	366077	71474
M2	$F_t \| C_{max} \| D_t$	9201	403351	103536
	$D_t \| C_{max} \| F_t$	9201	470706	64300
M3	$F_t \| C_{max} \| D_t$	8802	657615	131496
	$D_t \| C_{max} \| F_t$	8802	683585	88289

6.5.3 Computational experiments and evaluation

The proposed DSS has been evaluated in both industrial users by comparing the energy consumption of the machines before and after the adoption of the optimized schedule. This has been calculated externally, using energy consumption data in combination with the optimized schedules offered. Overall, the schedules provided by the proposed DSS have a considerable impact on the energy consumption in both users. The calculation at User A has shown an average reduction in energy consumption by 15.9% per month, mainly by reducing the number of shifts hence the indirect energy consumption but also by reducing the idle time of machines. User B has reported a slightly higher reduction of 16.5% in average that is, to the largest part, attributed to much smaller idling times of machines. This is of no surprise, since User A has a large manufacturing site with very high fixed energy costs but a rather smooth production that utilizes machines quite well thus allowing for no significant savings because of reducing the idle times. In contrast, User B has a smaller installation, in which fixed energy consumption is not particularly high; however, because of small lot sizes and only fewer machines, it suffers from long idle times especially in some energy-intensive machines that consume in stand-by mode approximately the same energy as in production mode.

Besides the observed energy consumption at the industrial users, various computational experiments have been performed to study the interrelationships among the optimization criteria, and in particular between the total flow time F_t and the total idle time D_t. Table 6.1 shows the results obtained for three different monthly planning periods (first column - M1, M2 and M3) for different objective function priorities (second column) for User A. The last three columns show the values (in thousands of minutes) for each optimization criterion per month.

The results of Table 6.1 show that, for this particular test-bed, the selection of the optimization hierarchy can play a critical role. In all cases, the makespan seems to be insensitive. On the contrary, the values of F_t and D_t are much in conflict, and they are significantly affected by the choice of the optimization hierarchy. Whenever focus is primarily given on minimizing F_t, the idle times are significantly increased and vice versa. For example, in month M3 the lowest total flow time is 657615 minutes for completing all orders of the production schedule and this comes with a total of 131496 thousand minutes idle time. Instead, if the total idle time is minimized first, the result is a 48% improvement from 131496 to 88289. The price is an increase of 3.7% regarding the total flow time from 657615 to 683585. The managerial implication here is that if direct idle energy consumption is more critical than indirect energy consumption, priority should be given to D_t; if, to the contrary, indirect energy dominates total energy consumption, then focus should be given to F_t or to C_{max} regarding the optimization hierarchy. Note that the results for User B are pretty similar but less illuminating since that user's data set is quite small-sized.

An additional set of computational experiments have been performed to study the effect of resource constraints on the optimization criteria. For this purpose, six indicative problem instances are generated from I_1 to I_6 based on the shop floor characteristics and the production scheduling attributes of both industrial users. Tables 6.2 and 6.3 summarize the results obtained. The first set of columns show the basic structural properties and the actual size of each problem instance in terms of number of jobs, operations and machines. The second set of columns indicate the hierarchy of objectives. In all experiments, the $C_{max}|F_t|D_t$ problem is solved. The last set of columns provide the results obtained by solving the scheduling problem without any resource constraints (RC_0) and with up to 3 resource constraints (i.e., RC_i is instance RC_0 with i resource constraints, $i = 1, 2, 3$) on the machine and process step level. The three renewable resources correspond to electricity, gas and high pressure steam. Without loss of generality it can be assumed that the available parallel machines at each process step are identical, there are sequence dependent changeover (set-up) times, all production orders have the same release date and half of the production order have a due date earlier than the end of the planning horizon. To that end, the first group of problem instances (I_1 to I_3 in Table 6.2) are identical to the second group of problems (I_4 to I_6 in Table 6.3); however, in the instances of the first group all machines are available at all times, while in the ones of the second group the availability of machines varies (up to 20% machine unavailability throughout the planning horizon).

Overall, the following observations can be made. At first, as resource constraints are added, the quality of the schedules is significantly affected and all optimization

Table 6.2: Results with renewable resource constraints

Instance			Criterion	Resource Constraints			
				RC_0	RC_1	RC_2	RC_3
I_1	Operations	100	C_{max}	1037	1565	2755	3241
	Jobs	10	F_t	8779	13740	23005	25327
	Machines	20	D_t	3621	6123	18847	22927
I_2	Operations	123	C_{max}	1504	2269	3995	4320
	Jobs	12	F_t	12378	19373	32437	35711
	Machines	20	D_t	5395	9123	28082	34161
I_3	Operations	148	C_{max}	2395	3615	6364	6881
	Jobs	14	F_t	20894	32701	54752	60278
	Machines	20	D_t	8437	14267	43914	53420

Table 6.3: Results with renewable resource constraints (cont.)

Instance			Criterion	Resource Constraints			
				RC_0	RC_1	RC_2	RC_3
I_4	Operations	100	C_{max}	1045	1737	2897	2979
	Jobs	10	F_t	9209	15245	23524	27659
	Machines	20	D_t	5357	9364	19811	25765
I_5	Operations	123	C_{max}	1515	2519	4201	4699
	Jobs	12	F_t	12985	21495	33169	38999
	Machines	20	D_t	7982	13952	29518	38390
I_6	Operations	148	C_{max}	2414	4012	6692	7487
	Jobs	14	F_t	21917	36283	55987	65828
	Machines	20	D_t	12482	21818	45160	67032

criteria are deteriorated; however, the effect on the total idle time seems to be very strong (total idle time gets many times higher) compared to the other optimization criteria. This is an indication that there is a trade-off between violating a resource (energy consumption) versus huge machine idle (or stand-by) times. Moreover, once machine unavailability occurs (see Table 6.3), the effect of adding energy resource constraints is even greater.

6.5.4 Deployment and experiences

The qualitative evaluation of the proposed services, has been performed following the ISO/IEC 9126 standard for evaluation of software quality. The personnel that has used these services has rated it in terms of efficacy (2.63/3), efficiency (2.68/3), understandability (2.51/3), satisfaction (2.63/3), learnability (2.44/3) and adaptability (2.40/3). All these scores show that the proposed DSS meets in a satisfying degree the end-users' expectations.

6.6 Concluding remarks

This study presents an energy-aware production scheduling DSS as designed, implemented and evaluated in a real context. In short, this work contributes to decision support for energy-efficient manufacturing by a metaheuristic algorithm that hierarchically optimizes flexible job-shop scheduling problems, a set of data requirements, the integrated deployment of this DSS as a web-service and the evaluation of the DSS in real settings.

The adopted scheduling framework incorporates various operational issues, while the data entities accompanying it meet generic energy-related requirements, as obtained from the literature and the textile industry. Apart from examining theoretical aspects regarding the design of energy-aware DSSs, this work presents the significant tangible benefits obtained from the use of such systems within the textile manufacturing industry. Hence, the applicability of the proposed DSS, as deployed in two significantly different users and production environments, is shown to be both feasible and effective.

Chapter 7

Towards primal-dual methods for binary multi-dimensional knapsack

This chapter paves the way towards future research, given the methods and algorithms obtained from Chapter 3. Here, we describe a new primal-dual method for the binary multi-dimensional knapsack problem, which is a well known (and strongly NP-hard) combinatorial optimization problem with many applications.

A binary multi-dimensional Knapsack Problem (0-1 MKP) is a problem of the form

$$\max\{c^T x : Ax \leq b, x \in \{0,1\}^n\},$$

where $c \in \mathbb{Z}_+^n$ is the objective function vector, $A \in \mathbb{Z}_+^{m \times n}$ is the matrix of constraint coefficients, and $b \in \mathbb{Z}_+^m$ is the vector of right hand sides. In other words, the 0-1 MKP is the special case of integer programming in which all variables are binary, all constraints are of less-than-or-equal-to type, and all objective and constraint coefficients are positive integers.

Given the above formulation, it is obvious that the problem at hand is so general in nature that encompasses all binary Integer Programming (IP) problems. Hence, there is a vast literature on the 0-1 MKP. Previous surveys on applications, complexity results, approximation algorithms, heuristics, upper bounds and exact algorithms are thoroughly covered by Fréville [42], Kelleler et al. [62] and Fréville & Hanafi [43]. Note here that the 0-1 MKP is strongly NP-hard [46], though, if the number of constraints is bounded by a constant, it can be solved in pseudo-polynomial time by dynamic programming. Recent surveys focus mainly on heuristics [20, 92, 64], genetic algorithms [67] or primal-dual methods [51] used as heuristics to obtain close-to-optimal solutions. Still, current exact methods can run into difficulties even for instances with $n \leq 200$ and $m \leq 5$, although larger instances can be solved if they have special structure.

In this work focus is given on a new primal-dual method for the problem at hand. This new algorithm uses the linear relaxation of the 0-1 MKP, enhanced by global lifted cover inequalities [57] to improve the upper bound and the proposed variant of the feasibility-pump heuristic (Chapter 3) that employs this family of cuts, improving the lower bound. Note here that, this algorithm is not a Branch and Cut (B&C) method. It is an algorithm that hopefully converges, iteration after iteration, to an optimal solution when the two bounds, i.e., upper and lower, are identical. The starting point of this research is the separation algorithm of valid global lifted cover inequalities, i.e., cover inequalities that take into consideration all the constraints of a 0-1 MKP [57]. The proposed separation algorithm by Kaparis & Letchford [58] starts from a fractional solution and produces heuristically valid cuts that apply for the whole constraint matrix. Motivated, by the previous work (Chapter 3) on the feasibility-pump (FP) heuristic that employs cutting planes in order to produce a better integer solution, we employ this separation algorithm [57] to enhance the solutions provided by FP, and use the feasible solution to generate new cuts added to the linear relaxation of the 0-1 MKP. When this is performed iteratively, it is expected that the feasible solutions provided by FP will get better and better, while the generated cuts, taking into consideration this solution will tighten more and more the relaxation. When the lower and upper bounds hopefully converge, an optimal solution is found.

The remainder of this chapter goes as follows: In Section 7.1 we describe the local and global lifted cover inequalities along with the separation algorithm proposed by Kaparis & Letchford. Section 7.2 describes the primal-dual method along with the integration of local ang global lifted cover inequalities in this new variant of the feasibility-pump heuristic. Finally, in Section 7.3 we provide preliminary computational results of the FP variant that uses local cover inequalities [48] and conclude this chapter with remarks and further research motivation.

7.1 Local and global lifted cover inequalities

This chapter is strongly based on [57, Section 2] and [58, Section 2], where the core theory and algorithms for the separation of local and global lifted cover inequalities are presented. Throughout this chapter, the following notation and terminology is used. The feasible region of the linear programming relaxation of the problem will be denoted by

$$P := \{x \in [0,1]^n : Ax \leq b\},$$

where n, A, and b are assumed to be fixed throughout this chapter. The convex hull of feasible integer solutions will be denoted by

$$P_I := \{x \in \{0,1\}^n : Ax \leq b\}.$$

Additionally it is assumed that the i-th knapsack constraint, i.e., the i-th inequality in the system $Ax \leq b$, takes the form

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i.$$

With the i-th constraint the 0-1 knapsack polytope is associated

$$Q_i := conv\{x \in \{0,1\}^n : \sum_{j=1}^{n} a_{ij} x_j \leq b_i\}.$$

Then, $P_I \subseteq \bigcap_{i=1}^{m} Q_i \subseteq P$ and for most problems of practical interest both containments are strict.

7.1.1 Lifted cover inequalities

Consider a 0-1 knapsack polytope of the form

$$Q := conv\{x \in \{0,1\}^n : a^T x \leq b\}.$$

. The set $C \subseteq N = 1, ..., n$ is a cover if it satisfies $\sum_{j \in C} a_j > b$. Given any cover C, the cover inequality $\sum_{j \in C} x_j \leq |C| - 1$ is clearly valid for Q. Moreover, the strongest cover inequalities are obtained when the cover C is minimal, in the sense that no proper subset of C is also a cover. In general, even minimal cover inequalities do not induce facets of Q. To make them facet-inducing, one must compute appropriate left hand side coefficients for the variables in $N \setminus C$, a process called *lifting*. The resulting *lifted* cover inequalities (LCIs) take the general form

$$\sum_{j \in C} x_j + \sum_{j \in N \setminus C} a_j x_j \leq |C| - 1,$$

where the lifting coefficients a_j satisfy $0 \leq a_j \leq |C| - 1$. Normally, lifting is performed sequentially, i.e., one variable at a time.

In general, different lifting sequences may give rise to different LCIs. However, it is not necessary to solve a sequence of $0-1$ knapsack problems to perform lifting. Several authors have presented results which make lifting much easier. First, Balas and Zemel [14] showed how to compute, in linear time, upper and lower bounds for the lifting coefficients which differ by at most one. For some variables, the upper and lower bounds coincide and the lifting coefficient is therefore immediately determined. Second, Zemel [110] presented a dynamic programming algorithm to compute exact lifting coefficients for the remaining variables. The algorithm runs in $O(|C||N \setminus C|)$ time and is very fast in practice. Finally, Gu et al. [49] showed how to strengthen the BalasZemel bounds without significantly increasing the time taken to compute them. As a result, even more lifting coefficients can be quickly fixed, leaving less work for Zemels algorithm to do.

Furthermore, lifting can be viewed in the following way: first take the face of Q defined by the equations $x_j = 0, \forall j \in N \setminus C$. The cover inequality $\sum_{j \in C} x_j \leq |C| - 1$ induces a facet of this face. Then, rotate this cover inequality to make it a facet of the original knapsack polytope. As noted by Wolsey and others [48, 79], an analogous procedure can be performed with faces defined by equations of the form $x_j = 1$. More precisely, for any cover C and any subset $D \subset C$, the inequality

$$\sum_{j \in C \setminus D} x_j \leq |C \setminus D| - 1$$

induces a facet of the restricted polytope

$$conv\{x \in \{0,1\}^{|C \setminus D|} : \sum_{j \in C \setminus D} a_j x_j \leq b - \sum_{j \in D} a_j\}$$

Standard sequential lifting then yields an inequality of the form

$$\sum_{j \in C \setminus D} x_j + \sum_{j \in N \setminus C} a_j x_j \leq |C \setminus D| - 1,$$

which induces a facet of the polytope

$$conv\{x \in \{0,1\}^{|N \setminus D|} : \sum_{j \in N \setminus D} a_j x_j \leq b - \sum_{j \in D} a_j\}$$

Finally, this can be lifted to obtain a facet of Q of the form

$$\sum_{j \in C \setminus D} x_j + \sum_{j \in N \setminus C} a_j x_j + \sum_{j \in D} \beta_j x_j \leq |C \setminus D| + \sum_{j \in D} \beta_j - 1.$$

The process of computing coefficients for the variables fixed at 1 (i.e., the variables in D) is sometimes called down-lifting. The computation of coefficients for the variables fixed at 0 (i.e., the variables in $N \setminus C$) is sometimes referred to as 'up-lifting'. The use of down-lifting enables one to construct LCIs which cannot be obtained using up-lifting alone. Computional results in Gu et al. [48] show that using down-lifting as well as up-lifting leads to a much more effective cutting plane algorithm.

As in the case of up-lifting, it is not actually necessary to solve a sequence of $0-1$ knapsack problems to perform down-lifting. Gu et al. [50] claim that Zemels algorithm can be adapted to compute all down-lifting coefficients in $O(|C|n^3)$ time. Although this is polynomial, it is time-consuming in practice. A faster and simpler alternative for down-lifting is to solve the LP relaxation of the auxiliary $0-1$ knapsack instances, which takes only $O(n^2)$ time in total. Of course, one should round down the optimal value of each lifting LP to the nearest integer, both to make the LCI as strong as possible, and to avoid having fractional lifting coefficients.

7.1.2 Global lifted cover inequalities

Kaparis & Letchford [57] introduced a new family of cuts based on LCIs. As mentioned above, the idea of using facets of the knapsack polytope to tackle more complex $0-1$ integer programs was already present in Crowder et al. [27]. They argued that, provided the constraint matrix A is sparse, the intersection of the individual Q_i should give a reasonable approximation to P_I itself.

However, in most instances of the $0-1$ MKP the constraint matrix A is dense, i.e., all variables participate in every knapsack constraint. In this situation, one cannot expect that valid inequalities derived from individual rows will always be useful, and it seems more sensible to attempt to derive valid inequalities which somehow take into account the global structure of the problem. Resorting to general-purpose cutting planes such as Gomory cuts is not an option, since these perform poorly for the $0-1$ MKP (Letchford & Lodi [66]). Inequalities of a more 'combinatorial' nature seem to be needed. Some explorations in this direction have been performed, for example, by Martin & Weismantel [76].

The $0-1$ MKP has the following nice property: to check if the inequality $\sum_{j \in C} x_j \leq |C|-1$ is valid for P_I, it suffices to check if it is valid for Q_i for some i, This property is not shared by general $0-1$ integer programs. Indeed, in general it is NP-hard to check if such an inequality is valid, even for $|C|=1$, as is easily shown. Given a cover inequality $\sum_{j \in C} x_j \leq |C|-1$ that is valid for P_I, and a subset $D \subset C$, an inequality of the form

$$\sum_{j \in C \setminus D} x_j + \sum_{j \in N \setminus C} a_j x_j + \sum_{j \in D} \beta_j x_j \leq |C \setminus D| + \sum_{j \in D} \beta_j - 1.$$

is a *global lifted inequality* (GLCI) if it is valid for P_I. A global LCI need not be valid for any of the individual Q_i .

The lifting process is thoroughly described by Kaparis & Letchford [57] and includes the computation of up-lifting and down-lifting coefficients. Instead of solving an instance of a $0-1$ MKP with the respective knapsack constraint in order to lift the derived valid inequality, Kaparis & Letchford [57] simply solve the LP-relaxation of these $0-1$ MKP instances, and round down to the nearest integer, in order to get the lifting coefficients. Although the resulting GLCIs are no longer guaranteed to induce facets of P_I, they can still be much stronger than standard LCIs as discussed in that paper.

Since this is a preliminary examination of this method, in this work only the local cover inequalities without lifting and the separation algorithm introduced by Gu et al. [49] are employed to

- tighten the LP-relaxation of the $0-1$ MKP;
- enhance the performance of feasibility-pump when used at each pumping cycle, forcing in this way the heuristic to converge faster to a feasible solution for the problem at hand.

7.2 Primal-dual method and FP heuristic

In this section we focus on a new primal-dual method for the problem at hand. This new algorithm uses the linear relaxation of the 0-1 MKP, enhanced by global lifted cover inequalities [57] to improve the upper bound and the proposed variant of the feasibility-pump heuristic (Chapter 3) that employs this family of cuts, improving the lower bound. Note here that, this algorithm is not a Branch and Cut (B&C) method. It is a primal-dual algorithm that hopefully converges, iteration after iteration, to an optimal solution when the two bounds, i.e., upper and lower, are identical.

As described in the previous sections, the global lifted cover inequalities take into consideration the constraint matrix as a whole. That is, the proposed separation algorithm by Kaparis & Letchford [58] starts from a fractional solution and produces heuristically valid cuts that apply for the whole constraint matrix. Note here, that the lifting procedure takes into consideration an integer point which is a round-up of the respective fractional solution. Motivated, by the previous work (Chapter 3) on the feasibility-pump (FP) heuristic that employs cutting planes, we propose employing this separation algorithm [57] to enhance the solutions provided by FP, and use the feasible solution to generate new lifted cuts added to the linear relaxation of the 0-1 MKP. When this is performed iteratively, it is expected that the feasible solutions provided by FP will get better and better, while the generated cuts, taking into consideration this solution will tighten more and more the relaxation. When the lower and upper bounds hopefully converge, an optimal solution is found.

Focusing first on the proposed FP variant, as described in Chapter 3, cutting planes can be employed to force feasibility by tigtening the LP-relaxation and by assisting the heuristic into converging on an integer solution when used in the pumping cycles. Algorithm 12 is the pseudocode of the basic version of the FP variant. All, variants of FP proposed in Chapter 3 have been encoded and tested, i.e., including the variants with different objective functions with constraint propagation and cuts. However, since it is quite straightforward for the reader to understand the cut addition phase in the pumping cycles, we provide the pseudocode only for the basic version.

Following this, Algorithm 13 is the pseudocode of the proposed primal-dual algorithm for the $0-1$ MKP.

7.3 Computational results

All components and methods are coded in ANSI C, using the IBM-ILOG CPLEX 12.5 callable library. The experiments are conducted under Linux Ubuntu 14.04, on a quad-core machine (Intel i7, 3.6GHz CPU speed, 16GB RAM). Each experiment includes 30 instances of the same dimension, i.e., number of variables and constraints, thus the average results over each such set of 30 instances are reported per experiment.

All FP variants are allowed up to 2000 pumping cycles, except when employed into a B&C algorithm where the maximum number of pumping cycles is reduced to 20. The main performance metric is the (average) integrality gap, defined as $IG = [(z^* - z_{LP})/z_{LP}] \cdot 100$, where z^* is the value of the solution found by the FP variant and z_{LP} the value of the LP-relaxation. The CPU time required is also reported (in seconds). Last, the average number of pumping cycles needed by each variant to find a feasible solution is also presented.

Algorithm 12 Pseudocode of the basic version of feasibility-pump with global lifted inequalities

$nIT := 0$
$distance = \infty$
initialize list l
$x^* = argmin\{c^T x \ : Ax \geq b\}$
if x^* is integer **then**
  return x^*
end if
while $distance \neq 0$ or $nIT < maxIterations$ **do**
  $nIT = nIT + 1$
  add global lifted cover inequalities to the LP that minimizes $\Delta(x, \widetilde{x})$
  $x^* = argmin\{\Delta(x, \widetilde{x}) \ : Ax \geq b\}$
  $distance = \Delta(x, \widetilde{x})$
  if x^* is integer **then**
    return x^*
  end if
  if $\exists\, j \in J \ : \ [x_j^*] \neq \widetilde{x}_j$ **then**
    $\widetilde{x} = [x^*]$
    if cycle detected **then**
      $\rho_j = rand(-0.3, 0.7)$
      for $i = 0$ **to** n **do**
        if $|x_j^* - \widetilde{x}_j| + max\{\rho_j, 0\}$ **then**
          flip $\widetilde{x}_j$ //Random restart
        end if
      end for
      empty list l
    end if
    keep the hash of $\widetilde{x}$ in list l
  else
    flip the $TT = rand(T/2, 3T/2)$ entries $\widetilde{x}_j$ $j \in J$ with highest $|x_j^* - \widetilde{x}_j|$
  end if
end while

Algorithm 13 Pseudocode of the primal-dual algorithm for the $0-1$ MKP

$iteration = 0$
$maxIterations = 1000$
$upperBound = 0$
$lowerBound = \infty$
$x_{IP} = 0^n$
$x_{LP} = 0^n$
while $iteration \neq maxIterations$ **do**
  solve the LP relaxation and set the fractional vector x_{LP}
  add global lifted inequalities to the LP
  reoptimize and and set the fractional vector x_{LP}
  if x_{LP} is integer **then**
    $x_{IP} = x_{LP}$
    break
  end if
  set the *upperBound* to the objective value of LP
  use the FP variant to get a feasible solution x_{IP}
  set the *lowerBound* to the objective value computed using x_{IP}
  if $upperBound = lowerBound$ **then**
    break
  end if
  $iteration = iteration + 1$
end while
return x_{IP}

We test these algorithms on a class of non-polynomially solvable instances in the literature, denoted as *mknapcbn* [24], n being the instance number as obtained from the OR-library [2].

Table 7.1: 0-1 MKP, FP variants: integrality gap, cycles, time

Instance	Dimensions			FP1	FP3	OFP1	OFP3	ORFP1	ORFP3
mknapcb1	Variables	100	Gap	47.60	45.22	54.86	53.93	56.86	55.73
	Constraints	5	Cycles	23	44	4	5	3	3
			Time	0.01	0.04	0.00	0.02	0.00	0.01
mknapcb2	Variables	250	Gap	45.47	43.62	53.86	53.73	55.77	55.65
	Constraints	5	Cycles	40	25	4	4	4	4
			Time	0.02	0.07	0.00	0.03	0.00	0.04
mknapcb3	Variables	500	Gap	43.77	39.31	54.26	54.20	56.42	55.94
	Constraints	5	Cycles	31	33	4	4	4	3.83
			Time	0.02	0.21	0.01	0.09	0.01	0.10
mknapcb4	Variables	100	Gap	46.57	38.93	48.49	48.36	53.22	51.39
	Constraints	10	Cycles	73	381	5	5	5	6
			Time	0.01	0.53	0.00	0.03	0.00	0.03
mknapcb5	Variables	250	Gap	45.04	38.36	50.13	49.97	54.76	54.47
	Constraints	10	Cycles	107	162	3	4	5	5
			Time	0.06	0.89	0.01	0.11	0.01	0.10
mknapcb6	Variables	500	Gap	43.24	44.89	51.63	51.53	54.33	54.78
	Constraints	10	Cycles	94	105	4	6	5	6
			Time	0.11	2.13	0.02	0.28	0.02	0.26
mknapcb7	Variables	100	Gap	34.02	27.43	28.92	37.25	31.27	31.07
	Constraints	30	Cycles	289	567	15	17.40	25	26
			Time	0.19	65.06	0.05	0.21	0.09	0.42
mknapcb8	Variables	250	Gap	28.68	31.89	37.60	35.44	39.68	37.41
	Constraints	30	Cycles	61	35	14	13	17	20
			Time	0.12	0.96	0.10	0.78	0.12	1.02
mknapcb9	Variables	500	Gap	40.69	34.33	41.88	42.18	40.03	40.49
	Constraints	30	Cycles	55	26	7	7	12	11
			Time	0.20	2.91	0.10	1.79	0.20	1.99

As shown by the results in Table 7.3, the variants that use cuts, are marginally better in terms of integrality gap compared to the standard variants. Most likely, this marginal improvement is achieved merely by the reduction of the upper bound when cuts are added to the LP relaxation of the problem. In some cases, the number of pumping cycles is reduced, this is not standard though. Recall that the employed cuts are unlifted cover cuts, thus are weaker compared to the lifted ones. Again, time-wise the new variants are more expensive, since there is an extra computational time required for the separation of cover cuts. Most likely, when lifted, hence tighter, cover cuts are used the performance of these new variants could outperform the standard variants. However, our effort up to this point allows for such families to easily be incorporated while our results remain motivating.

To summarize, this chapter describes a new primal-dual method for the binary multi-dimensional knapsack problem, which is a well known (and strongly NP-hard) combinatorial optimization problem with many applications. Indeed, its structure and mathematical formulation are so general in nature that it encompasses all binary Integer Programming problems. Current exact approaches and commercial solvers run into difficulties even for a small-to-medium number of constraints and variables.

The proposed primal-dual method employs the linear relaxation of the problem at hand, enhanced by global lifted cover inequalities to improve the upper bound and a new version of the feasibility-pump heuristic that uses local unlifted cover inequalities in the pumping procedure to obtain better and feasible lower bounds. This new variant of feasibility-pump is tested mainly on literature instances, for a good portion of which there are still no optimal solutions available. The results of the heuristc are interesting enough to trigger further research and development for the proposed primal-dual method.

Chapter 8

Concluding remarks

This thesis discusses optimization methods and Decission Support Systems (DSS) in an integrated approach. That is, algorithmic components for optimization are designed, considered and analyzed as part of a DSS that could be used for applications of respective problems. To do so, three different optimization methods are deployed for three different optimization problems, namely, the energy-aware production scheduling problem, the multi-index assignment and the multi-dimensional knapsack problem. For all three problems various algorithms are proposed and tested computationally, while for the first two problems two different DSS are proposed. Since these problems have no common structure and different applications, we study and present each problem separately, along with the algorithmic components, the proposed DSS and the computational results. In particular, the proposed DSS along with their user requirements, which are analyzed and discussed thoroughly, are results of two research projects.

First we focus on the multi-index assginment problem. In this part of the thesis we address the question of whether an exact method can solve large instances of the 3-index axial and planar problems. Furthermore, a subset of the algorithmic components deployed and tested, has been incorporated in a DSS, namely MAPS (the Multi-index Assignment Problem Solver), designed with higher-level user requirements so as to fit the various applications of the problem at hand. A relevant question is whether algorithmic and software components that work effectively for different types of assignment are plausible. Motivated also by the so-called *integrated methods for optimization*, we propose a *Branch & Cut* (B&C) solver integrating several components, namely cuts that are specific per assignment type, branching on 'Special Ordered Sets of type I', a tabu scheme that is simple enough to remain applicable for all assignment problems, a constraint propagator that can also be used for all assignment problems and Feasibility Pump (FP) as an LP-based heuristic that also sustains applicability across different assignment problems. In fact, here is used the improved FP-variant that employs both constraint propagation and cutting planes at each ‘pumping cycle’. That is, cuts are used in a primal-dual mode to improve the lower bound in a typical manner and guide the heuristic towards a better upper bound. This experimentation shows that the new FP variant produces better feasible solutions compared to existing one, while the B&C method outperforms a commercial solver, particularly for large-size instances and for planar problems. Indeed, this solver performs better than CPLEX, particularly for larger instances and more evidently in the planar case. Further experimentation may offer deeper insights on both the performance of such an approach and on the ability to solve exactly even larger-scale instances or other multi-index assignment problems. An important aspect of this approach is its versatility, for example in terms of including a subset of the selected components or an alternative FP variant as a primal heuristic.

Following this we focus on the energy-aware production schedulling. The results of its study are part of a European research project, namely ARTISAN, focusing on reducing the energy-consumption in the textile-manufacturing sector by 10%. Indeed, modern manufacturing companies are forced to become energy-aware under the pressure of energy costs, legislation and consumers’ environmental awareness. Production scheduling remains a critical decision making process, although demanding in computational terms and sensitive on data availability and credibility. Hence, incorporating energy-related criteria in production scheduling has become more important. In this part of the thesis we describe an energy-aware production scheduling DSS, composed by an Iterated Local Search algorithm that offers hierarchical optimization over multiple scheduling criteria and a generic yet concise data model whose entities are extracted from the literature and actual user requirements. The results of embedding this DSS in an integrated system used by two textile manufacturers show that it indeed supports efficiently energy-aware production scheduling. In short, this work contributes to decision support for energy-efficient manufacturing by a metaheuristic algorithm that hierarchically optimizes flexible job-shop scheduling problems, a set of data requirements, the integrated deployment of this DSS as a web-service and the evaluation of the DSS in real settings. The adopted scheduling framework incorporates various operational issues, while the data entities accompanying it meet generic energy-related requirements, as obtained from the literature and the textile industry. Apart from examining theoretical aspects regarding the design of energy-aware DSS, this work presents the significant tangible benefits obtained from the use of such systems within the textile manufacturing industry. Hence, the applicability of the proposed DSS, as deployed in two significantly different users and production environments, is shown to be both feasible and effective.

Finally we focus on the binary multi-dimensional knapsack problem as part of our ongoing research effort that shapes also future work. In this last part of the thesis we describe a new primal-dual method for the binary Multi-Dimensional Knapsack Problem, which is a well known (and strongly NP-hard) combinatorial optimization problem with many applications. Indeed, its structure and mathematical formulation are so general in nature that it encompasses all binary Integer Programming problems. Current exact approaches and commercial solvers run into difficulties even for a small-to-medium number of constraints and variables. The proposed primal-dual method employes the linear relaxation of the problem at hand, enhanced by global lifted cover inequalities to improve the upper bound and a new version of the feasibility-pump heuristic that uses local unlifted cover inequalities in the pumping procedure to obtain better and feasible lower bounds. This new variant of feasibility-pump is tested mainly on literature instances, for a good portion of which there are still no optimal solutions available. The results of the heuristc are interesting enough to trigger further research and development for the proposed primal-dual method.

Appendix A

Appendix A

A.1 MAPS use case analysis

Table A.1: Use case 1 - Register

Use case 1	**Register**	
Brief description	This use case states the actions taken in order to register in MAPS.	
Primary actors	Visitor	
Pre-conditions		
Post-conditions	Visitor is a registered member of MAPS.	
Basic flows	*Tasks*	*Information required*
	1. User enters the required details.	first name, last name, username, password, website
	2. System checks if the inserted information is correct.	
	3. User is prompted to log in.	
Alternative flows	*Tasks*	*Information required*
	In Task 2, if the inserted information is not correct, prompt the user to correct it.	

Table A.2: Use case 2 - Log in

Use case 2	**Log in**	
Brief description	This use case states the actions taken in order to log in.	
Primary actors	Registered member	
Pre-conditions	User has successfully completed registration.	
Post-conditions	User is logged in.	
Basic flows	*Tasks* 1. User enters his username and password. 2. System checks if the inserted information is correct. 3. User is logged in.	*Information required* username, password
Alternative flows	*Tasks* In Task 2, if the inserted information is not correct, prompt user to insert a correct username and password.	*Information required*

A.2 Overview of MAPS screens

A.2.1 Register

Once the user has selected to register in the system, he is prompted to provide some personal information including

- username: must be unique. No duplicate usernames are allowed;
- password: has to be at least 8 characters;
- first and last name: each one has to be at least 3 characters;
- email: must be of the form *someone@somewhere.something*;
- his personal web-page: must follow the template of a valid URL.

The user has to submit all of the above information. If any piece of the required information is not correct, the user is prompted to correct it.

Once the user has inserted correctly the required information, he is prompted to log-in.

Table A.3: Use case 3 - View saved solved instances

Use case 3	**View solved instances**	
Brief description	This use case states the actions taken in order to view previously solved instances.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User views a list of solved instances.	
Basic flows	*Tasks*	*Information required*
	1. User is prompted to select the category of instances he wishes to view, i.e. $(k, s)AP_n$ or *all-different* instances.	
	2. System retrieves the solved instances of the selected category.	instance category, instance id
	3. System displays the instances of the selected category.	
Alternative flows	*Tasks*	*Information required*

Figure A.1: Use Case 1 - Register: Main screen

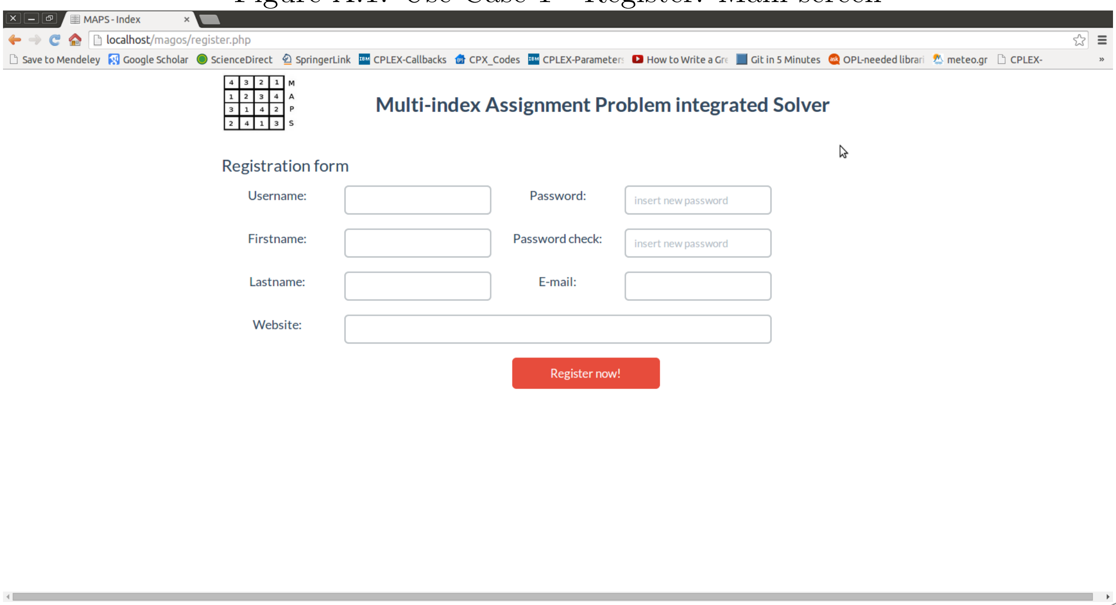


Table A.4: Use case 4 - Solve a new $(k, s)AP_n$ instance

Use case 4	**Solve a new $(k, s)AP_n$ instance**	
Brief description	This use case states the actions taken in order to solve a new $(k, s)AP_n$ instance.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User views the results of the solved instance.	
Basic flows	*Tasks*	*Information required*
	1. User enters the dimensions of the instance.	parameter k, parameter s, parameter n
	2. User selects from a list the algorithms, with which he wishes to solve the instance.	algorithm id
	3. System solves the instance with the selected algorithms.	
	4. System presents the results of the selected algorithms in a table.	parameter k, parameter s, parameter n, algorithm name, integrality gap, optimality gap, LP value, IP value, solution time
	5. User is prompted to export the table of results in *tex*, *csv* or *pdf* format.	
	6. User is prompted to save or view a particular solution.	
Alternative flows	*Tasks*	*Information required*

Table A.5: Use case 5 - View a $(k, s)AP_n$ solution from a single algorithm

Use case 5	**View a $(k, s)AP_n$ solution from a single algorithm.**	
Brief description	This use case states the actions taken in order to view a $(k, s)AP_n$ solution from a single algorithm	
Primary actors	Registered member	
Pre-conditions	User is logged in and has solved the instance, the solution of which he wishes to view.	
Post-conditions	User views the detailed results of the solution.	
Basic flows	*Tasks*	*Information required*
	1. User selects from a list the solution he wishes to view	
	2. System retrieves the detailed results for the selected solution.	solution id
	3. System presents the detailed results of the selected solution.	parameter k, parameter s, parameter n, solution vector, integrality gap optimality gap, IP value, LP value, solution time
	4. User is prompted to export the cost coefficients vector.	
	5. User is prompted to resolve the same instance or to return to the results presented in use case 4.	
Alternative flows	*Tasks*	*Information required*

Table A.6: Use case 6 - Load costs of a $(k, s)AP_n$ instance and solve it

Use case 6	**Load costs of a $(k, s)AP_n$ instance and solve it**	
Brief description	This use case states the actions taken in order to load the cost coefficients of a $(k, s)AP_n$ instance and solve it.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User has loaded the costs of an instance and solved it.	
Basic flows	*Tasks*	*Information required*
	1. User enters the dimensions of the instance.	parameter k, parameter s, parameter n
	2. User loads a *csv* file with the cost coefficients.	
	3. User selects from a list the algorithms, with which he wishes to solve the instance.	algorithm id
	4. System solves the instance with the selected algorithms.	
	5. System presents the results of the selected algorithms in a table.	parameter k, parameter s, parameter n, algorithm name, integrality gap, optimality gap, LP value, IP value, solution time
	6. User is prompted to export the table of results in *tex*, *csv* or *pdf* format.	
	7. User is prompted to save or view a particular solution.	
Alternative flows	*Tasks*	*Information required*

Table A.7: Use case 7 - Save a $(k, s)AP_n$ instance solution

Use case 7	**Save a $(k, s)AP_n$ instance solution.**	
Brief description	This use case states the actions taken in order to save a $(k, s)AP_n$ instance solution.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User has solved a $(k, s)AP_n$ instance.	
Basic flows	*Tasks*	*Information required*
	1. User selects the solution to be saved	
	2. System retrieves the solution information	solution id, parameter k, parameter s, parameter n, algorithm name, integrality gap, optimality gap, LP value, IP value, solution time
	3. User is prompted to add a short description of the solution.	solution description
	4. System saves the selected instance.	
Alternative flows	*Tasks*	*Information required*

Table A.8: Use case 8 - Delete a saved $(k, s)AP_n$ instance solution

Use case 8	**Delete a saved $(k, s)AP_n$ instance solution.**	
Brief description	This use case states the actions taken in order to delete a saved $(k, s)AP_n$ instance solution.	
Primary actors	Registered member	
Pre-conditions	User is logged in and has saved a $(k, s)AP_n$ instance solution.	
Post-conditions	User has deleted a $(k, s)AP_n$ instance solution.	
Basic flows	*Tasks*	*Information required*
	1. User selects the solution to be deleted.	
	2. System deletes the selected solution.	solution id
Alternative flows	*Tasks*	*Information required*

Table A.9: Use case 9 - Solve a new *all-different* instance

Use case 9	**Solve a new *all-different* instance.**	
Brief description	This use case states the actions taken in order to solve a new *all-different* instance.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User has solved a new *all-different* instance.	
Basic flows	*Tasks*	*Information required*
	1. User enters the number of *all-different* constraints.	constraints number
	2. User enters the number of variables.	variables number
	3. User enters the number of variables.	variables number
	4. User defines the discrete domain for each variable.	variable domain
	5. For each constraint, the user enters the variables that participate in it.	variables, constraints
	6. Given the above input, the system generates an ILOG script.	variables, constraints
	7. System solves the *all-different* instance.	
	8. System presents the *all-different* instance solution.	solution vector
	9. User is prompted to save the solution or export the generated ILOG script.	
Alternative flows	*Tasks*	*Information required*
	1. After defining the *all-different* system, the the user wants to optimize it	
	2. The user defines if he wishes to minimize or maximize the *all-different* system	optimization type
	3. The user inserts the costs of the variables	variables, costs
	4. The system continues from task 6 of the basic flow	

Table A.10: Use case 10 - Save a *all-different* instance solution

Use case 10	**Save a *all-different* instance solution.**	
Brief description	This use case states the actions taken in order to save a *all-different* instance solution.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User has solved a *all-different* instance.	
Basic flows	*Tasks*	*Information required*
	1. User selects the solution to be saved	
	2. System retrieves the solution information	variables number, constraints number, ILOG script, solution vector
	3. User is prompted to add a name and a short description of the solution.	solution description, solution name
	4. System saves the selected instance.	
Alternative flows	*Tasks*	*Information required*

Table A.11: Use case 11 - Delete a saved *all-different* instance solution

Use case 11	**Delete a saved *all-different* instance solution.**	
Brief description	This use case states the actions taken in order to delete a saved *all-different* instance solution.	
Primary actors	Registered member	
Pre-conditions	User is logged in and has saved a *all-different* instance solution.	
Post-conditions	User has deleted a *all-different* instance solution.	
Basic flows	*Tasks*	*Information required*
	1. User selects the solution to be deleted.	
	2. System deletes the selected solution.	solution id
Alternative flows	*Tasks*	*Information required*

Table A.12: Use case 12 - View the MAPS manual

Use case 12	**View the MAPS manual.**	
Brief description	This use case states the actions taken in order to view the MAPS manual.	
Primary actors	Registered member, visitor	
Pre-conditions		
Post-conditions	User can read the manual.	
Basic flows	*Tasks*	*Information required*
	1. System retrieves and presents the table of contents of the manual.	manual ToC
	2. User selects the section he wishes to read.	
	3. System retrieves and presents the contents of the selected section.	section contents
Alternative flows	*Tasks*	*Information required*

Table A.13: Use case 13 - Edit user account

Use case 13	**Edit user account.**	
Brief description	This use case states the actions taken in order to edit user account.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User's personal information is updated.	
Basic flows	*Tasks*	*Information required*
	1. System retrieves and presents user's personal information.	username, password, first name, last name, email, website
	2. User edits his personal information.	
	3. System checks if the submitted information is correct.	
Alternative flows	*Tasks*	*Information required*
	1. In step 3, if the submitted information is not not correct, user is prompted to correct it.	

Table A.14: Use case 14 - Log out

Use case 14	**Log out.**	
Brief description	This use case states the actions taken in order to log out.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User is logged out.	
Basic flows	*Tasks*	*Information required*
	1. System logs out the user.	
Alternative flows	*Tasks*	*Information required*

Table A.15: Use case 15 - Delete personal account

Use case 15	**Delete personal account.**	
Brief description	This use case states the actions taken in order to delete the personal account.	
Primary actors	Registered member	
Pre-conditions	User is logged in.	
Post-conditions	User is logged out and has deleted his personal account.	
Basic flows	*Tasks*	*Information required*
	1. User confirms that he wishes to delete his account.	
	1. System logs out the user.	
	1. System deletes user's account.	user id
Alternative flows	*Tasks*	*Information required*

Figure A.2: Use Case 1 - Register: False entry

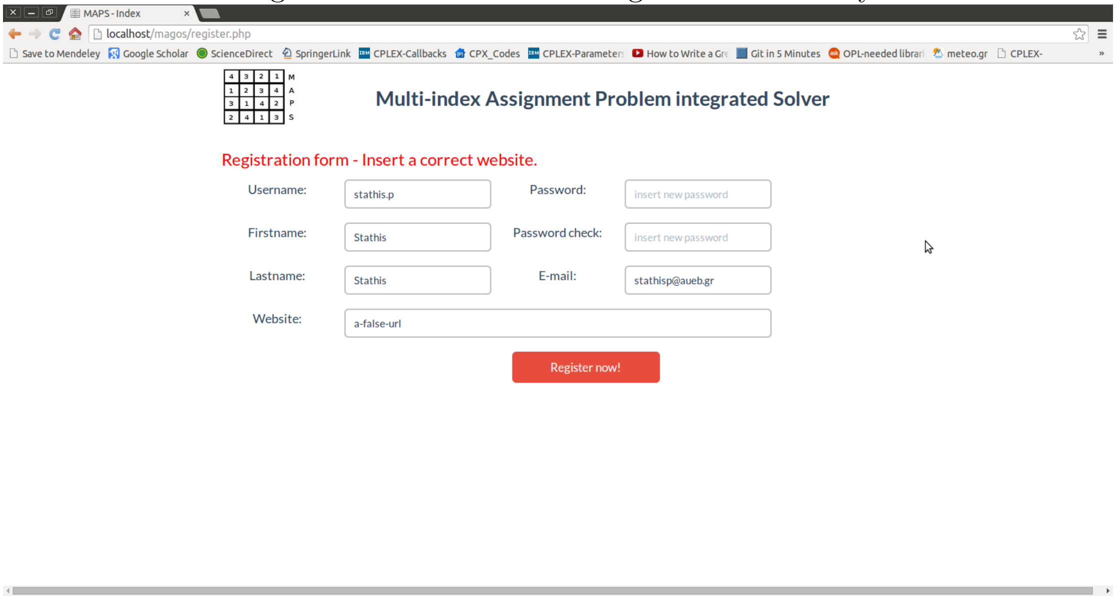
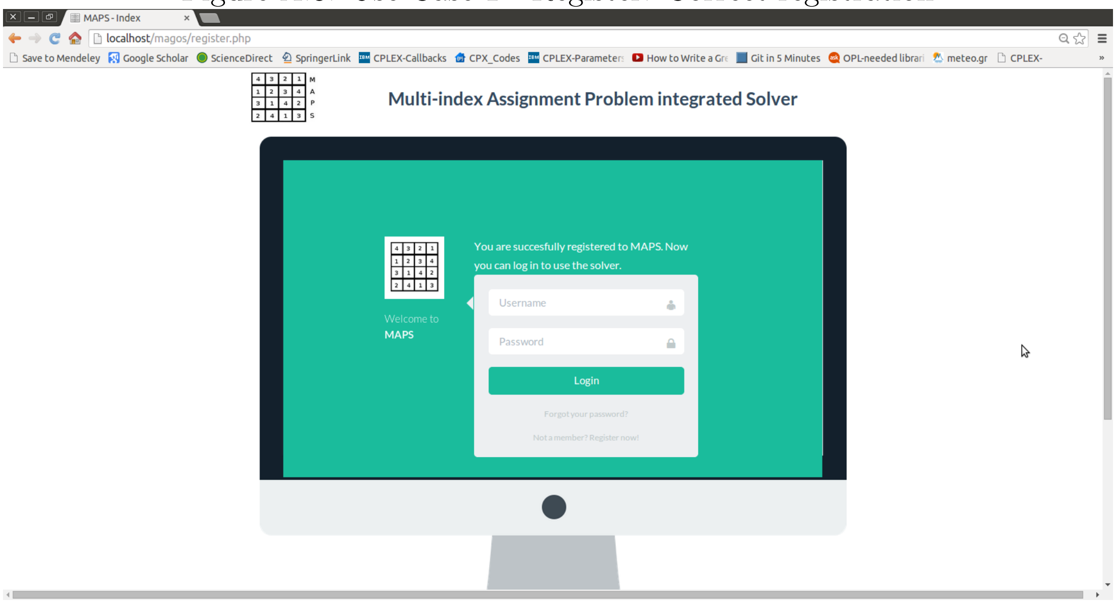


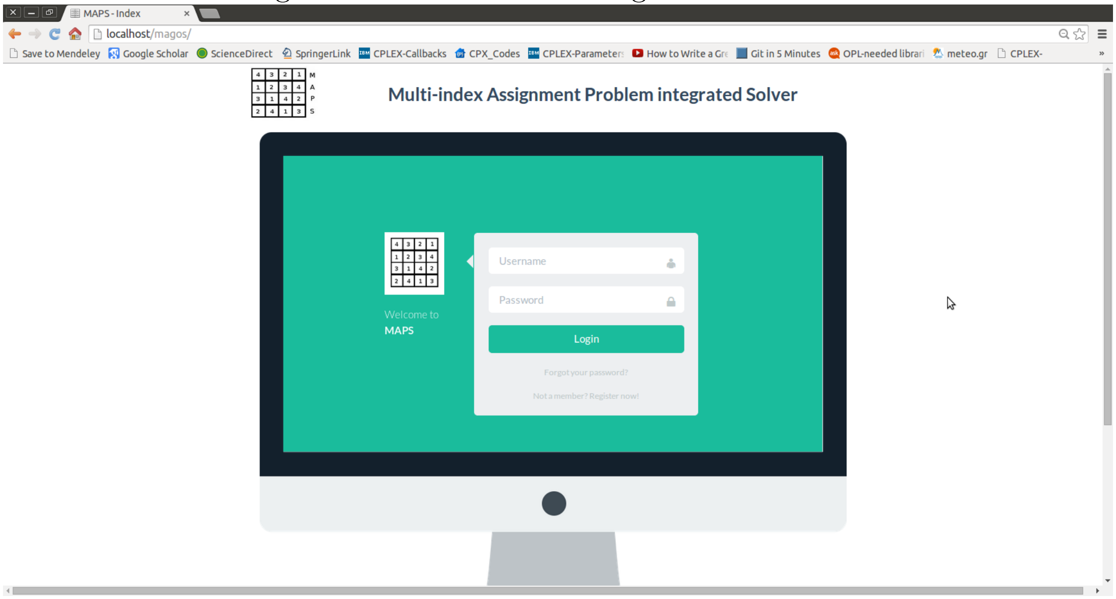
Figure A.3: Use Case 1 - Register: Correct registration



A.2.2 Log-in

Once the user has selected to log-in he has to enter his username and password.

Figure A.4: Use Case 2 - Log in: Main screen



If either the username or the password are incorrect, the user is prompted to retry.

Once the user has inserted correctly his username and password he can use the integrated solver.

A.2.3 View saved solved instances

Once the user has selected to view the solved instances he has saved, a list of these instances appear separated in two categories, i.e. the $(k, s)AP_n$ instances and the $all-different$ instances. The following screen-shots display the list and the categorization of the solved instances. Each user has a quota of 10 instances, i.e. he can not save more than 10 instances. The size of quota can be seen at any point in the bar above the saved instances.

If there are no saved instances then the list is empty and the user is informed appropriately.

A.2.4 Save a $(k, s)AP_n$ instance solution

Once the user has solved a $(k, s)AP_n$ instance he can save simply by selecting it. Consequently, he is asked to insert a description of the instance, and finally save it.

Figure A.5: Use Case 2 - Log in: False entry

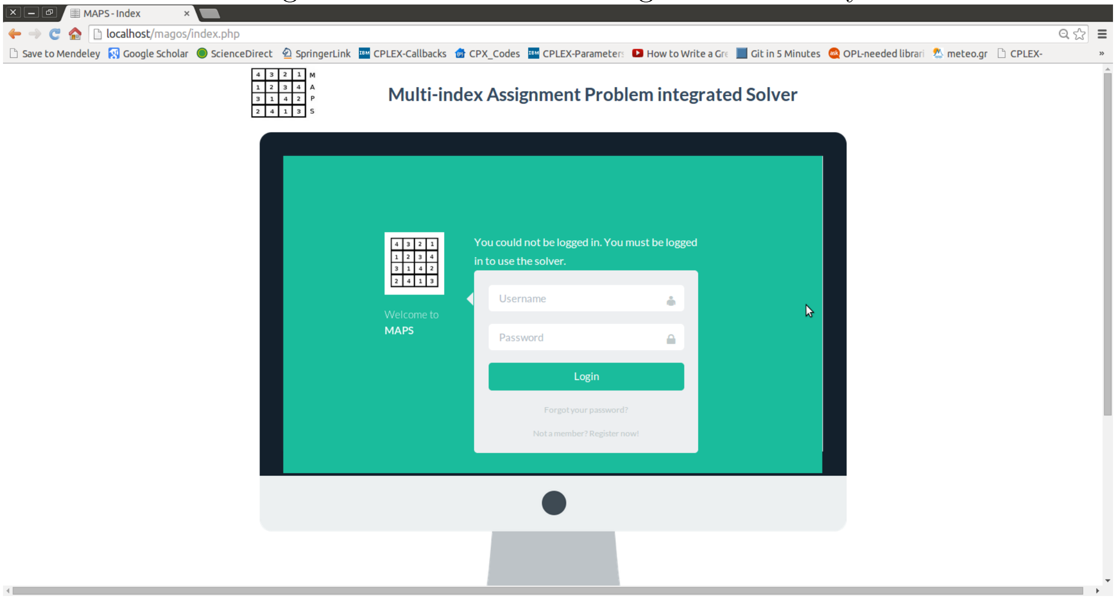


Figure A.6: Use Case 2 - Log in: Successful log-in

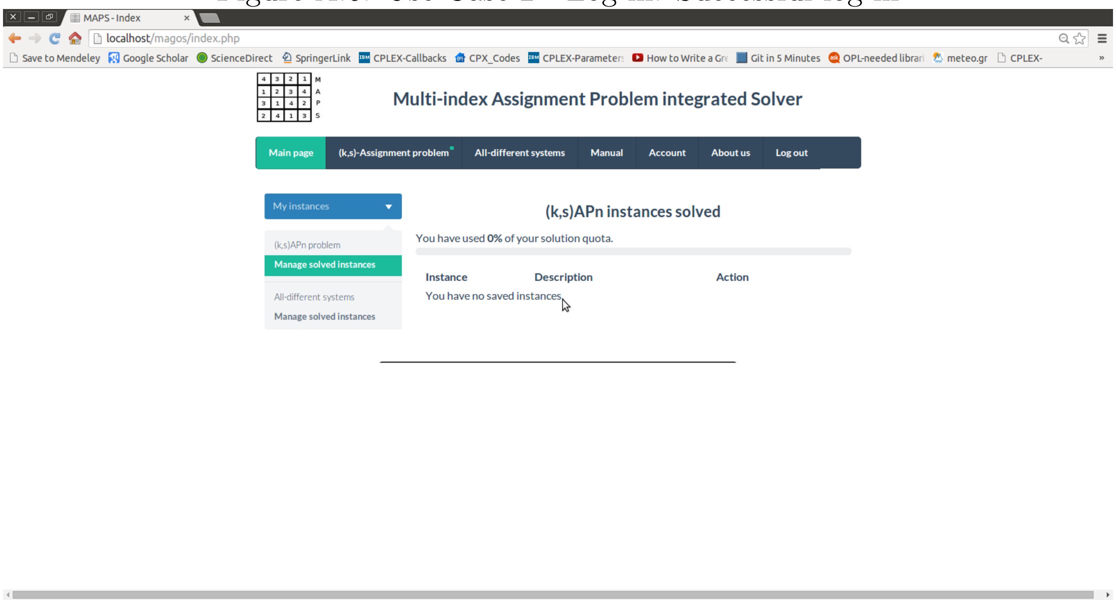


Figure A.7: Use Case 3 - View saved solved instances: $(k, s)AP_n$ instances 1

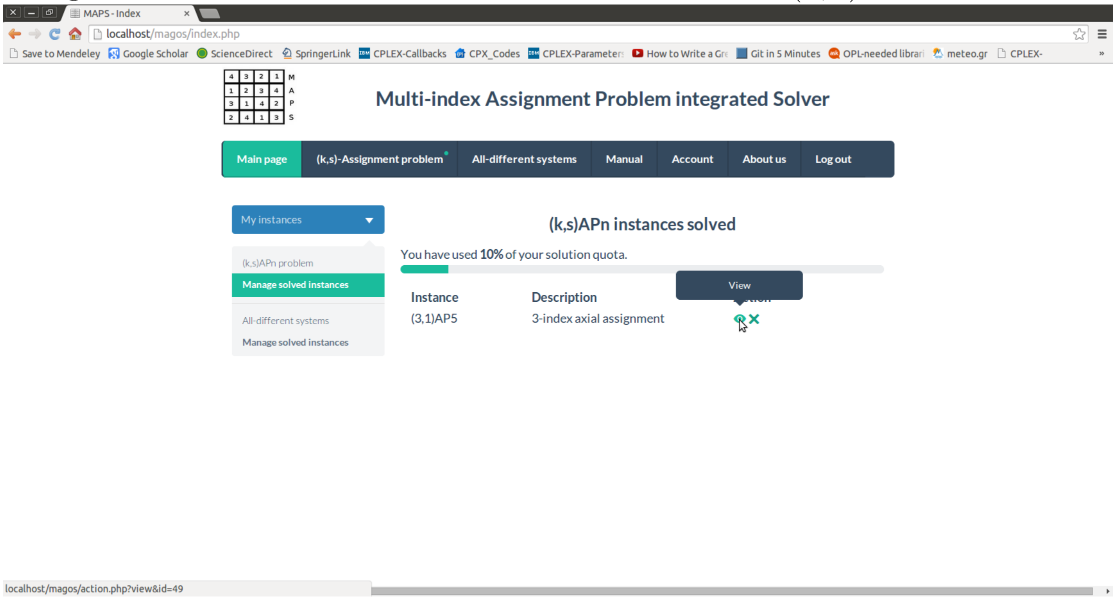


Figure A.8: Use Case 3 - View saved solved instances: $(k, s)AP_n$ instances 2

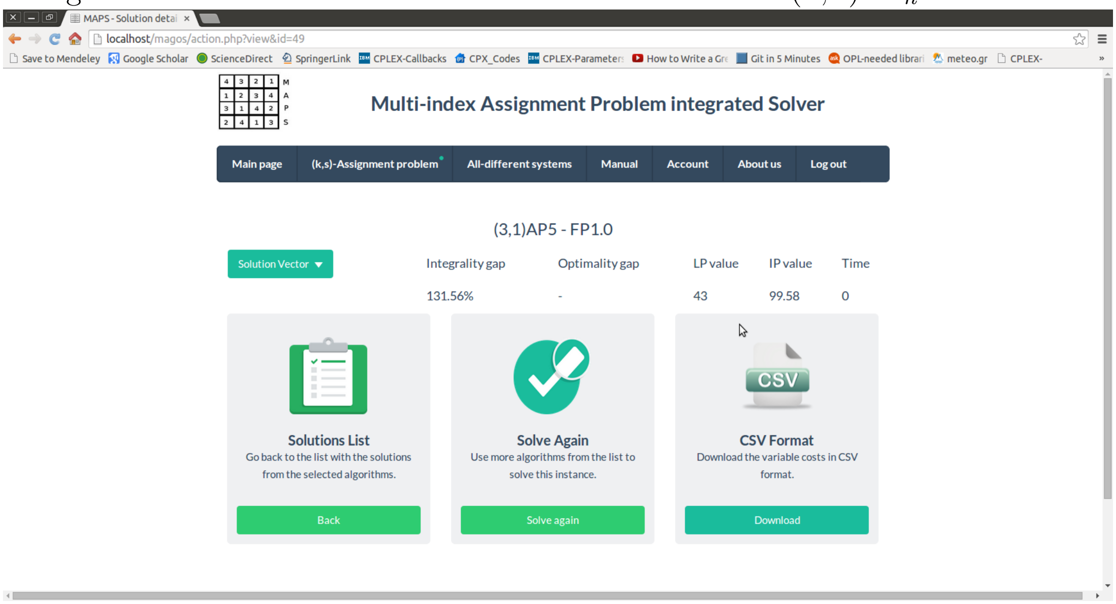


Figure A.9: Use Case 3 - View saved solved instances: *all-different* instances 1

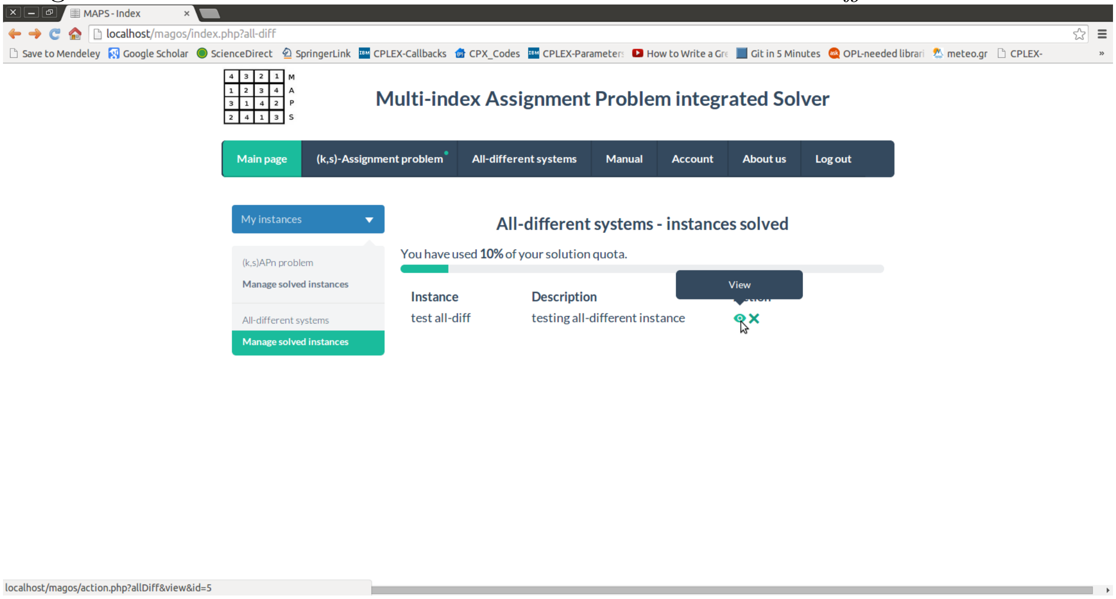


Figure A.10: Use Case 3 - View saved solved instances: *all-different* instances 2

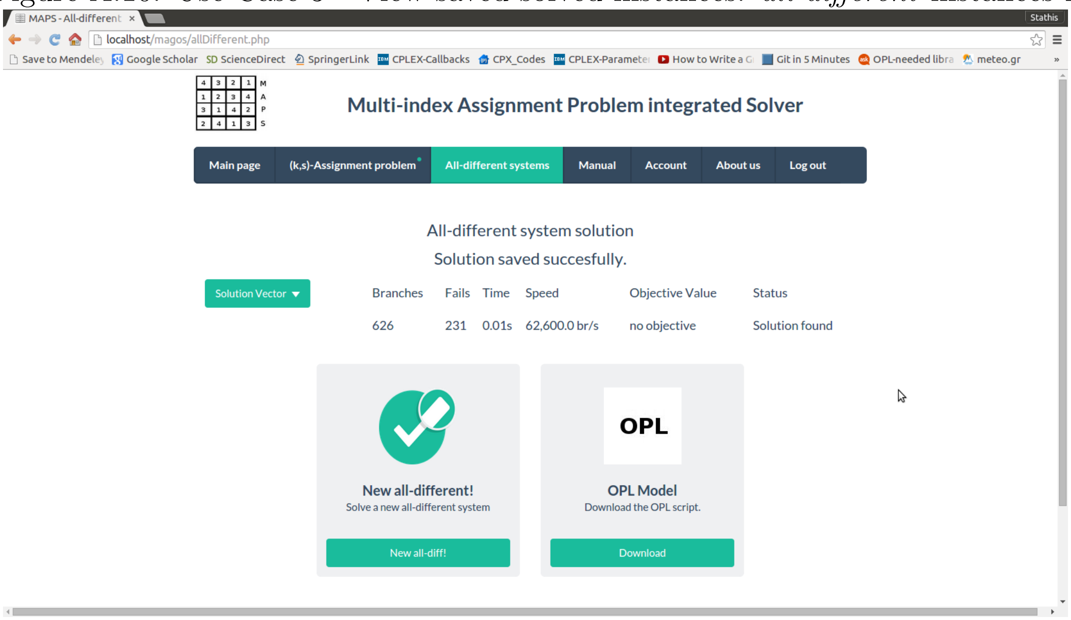


Figure A.11: Use Case 3 - View saved solved instances: Empty list of $(k, s)AP_n$ instances

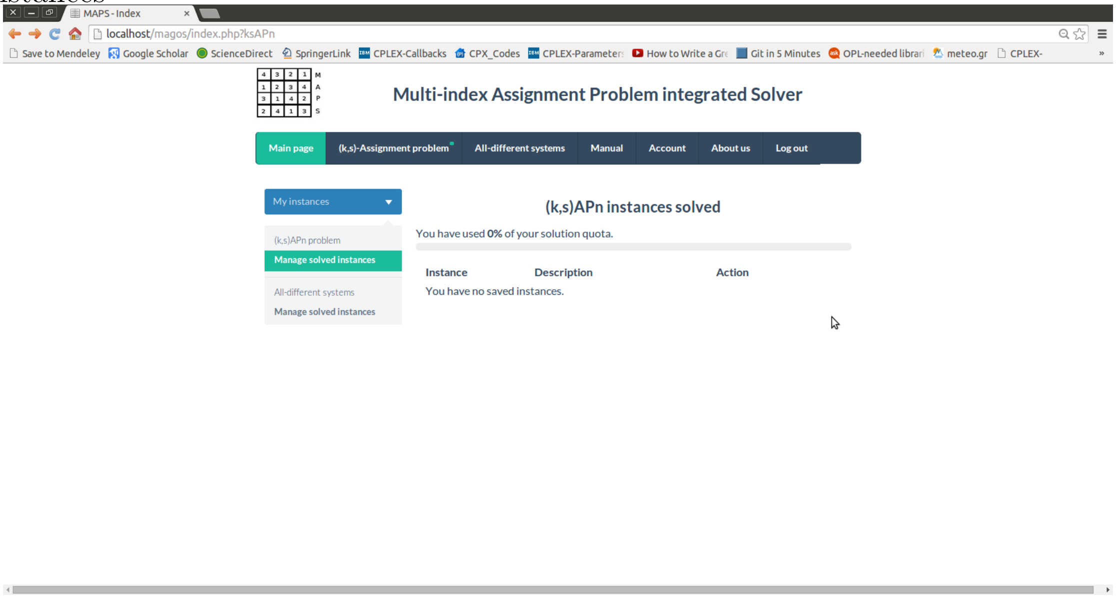


Figure A.12: Use Case 3 - View saved solved instances: Empty list of $all - different$ instances

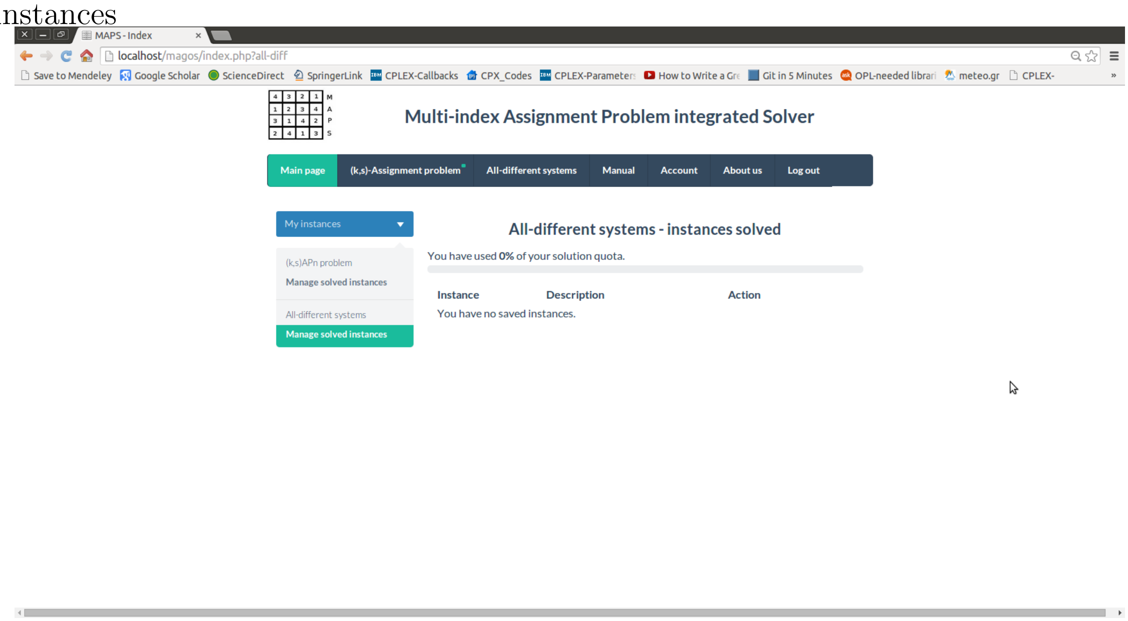


Figure A.13: Use case 7 - Save a $(k, s)AP_n$ instance solution: Solution selection

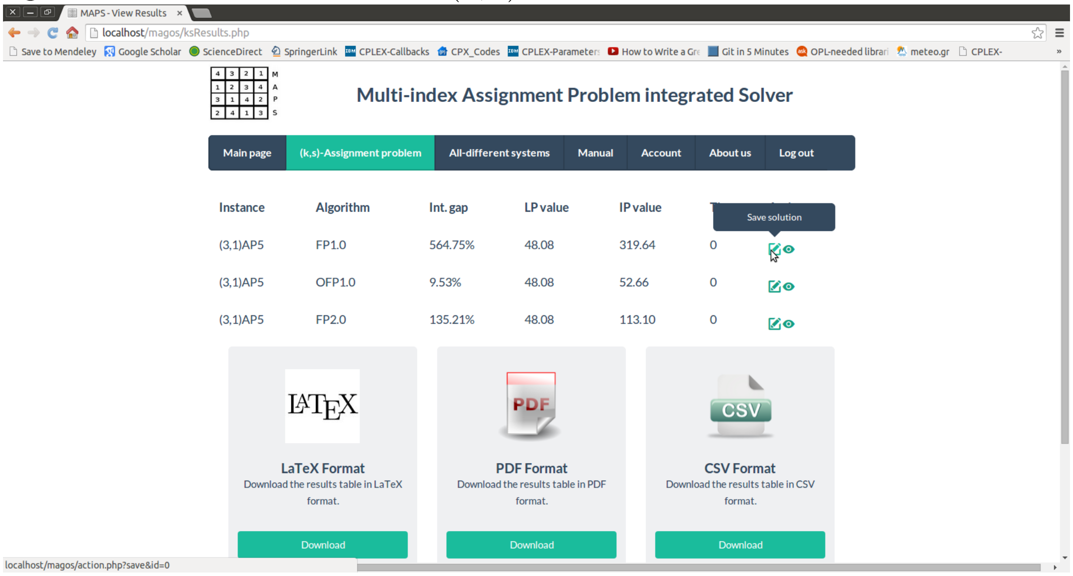


Figure A.14: Use case 7 - Save a $(k, s)AP_n$ instance solution: Saving the solution

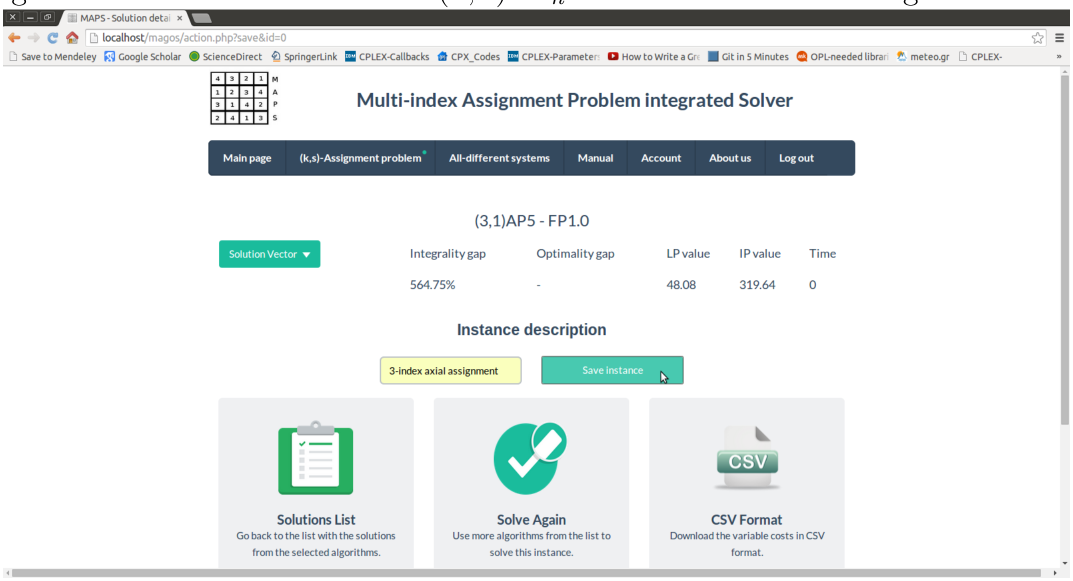
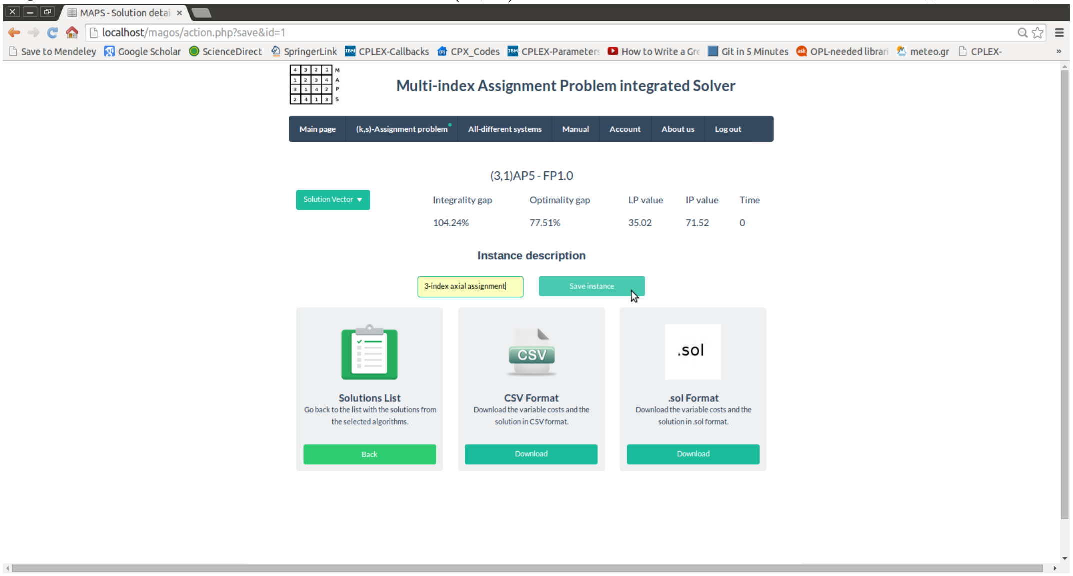
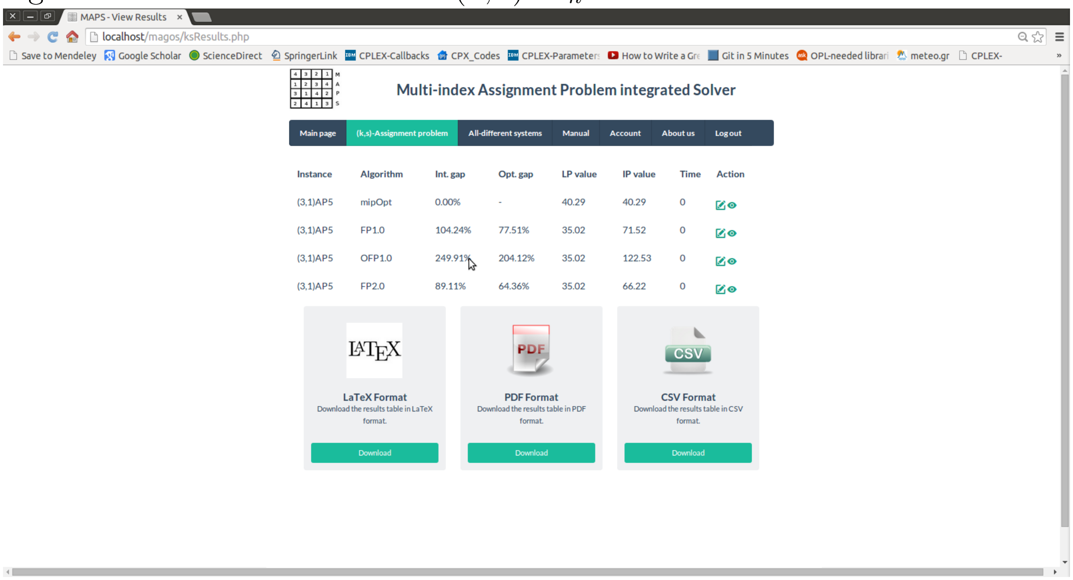


Figure A.15: Use case 7 - Save a $(k, s)AP_n$ instance solution: Description input



Once he saves the solution, he is automatically redirected to the table with the results of this instance.

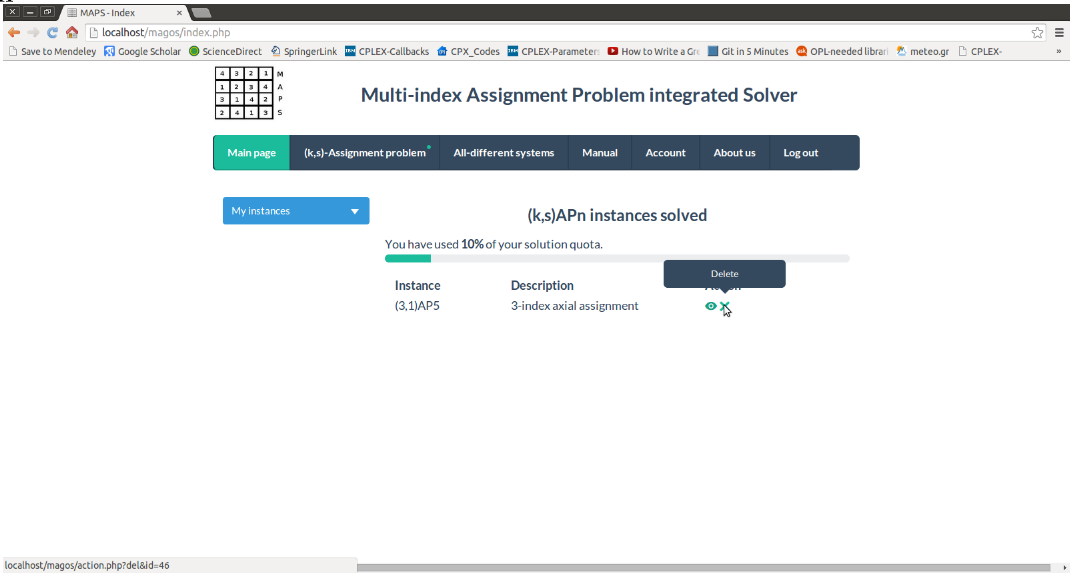
Figure A.16: Use case 7 - Save a $(k, s)AP_n$ instance solution: Solution is saved



A.2.5 Delete a saved $(k, s)AP_n$ instance

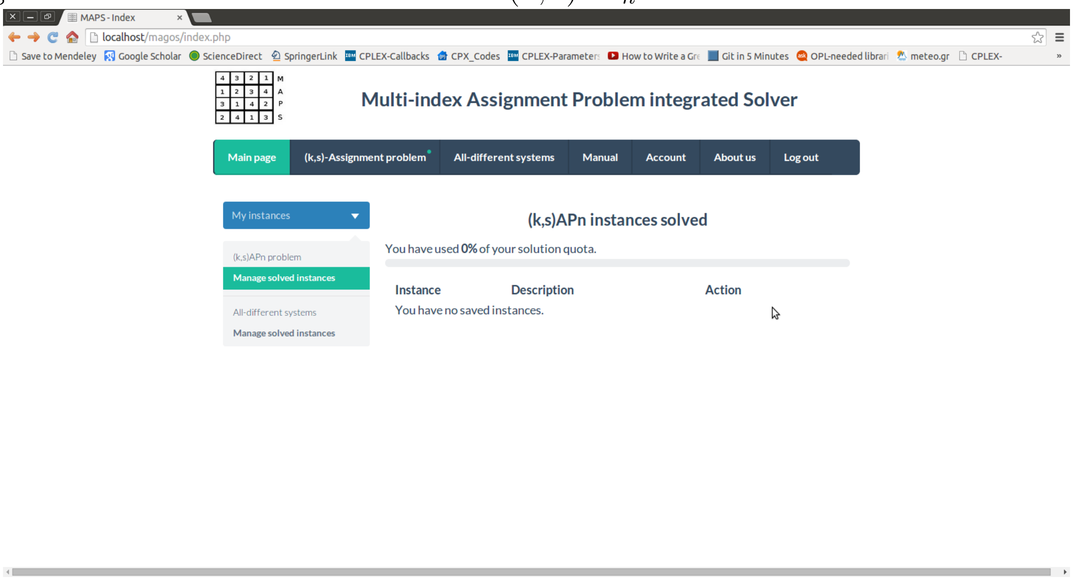
Once the user has solved and saved a $(k, s)AP_n$ instance he can delete it by going to the main page and selecting the instance he wishes to delete.

Figure A.17: Use case 8 - Delete a saved $(k, s)AP_n$ instance solution: Selecting solution



Once he selects it, the instance is deleted, hence removed from the list of the saved solutions.

Figure A.18: Use case 8 - Delete a saved $(k, s)AP_n$ instance solution: Solution deleted



A.2.6 Save an *all-different* instance solution

Once the user has solved an $all - different$ instance he can save it by entering the name and the description of the instance.

Figure A.19: Use case 10 - Save an *all-different* instance solution: Entering instance information

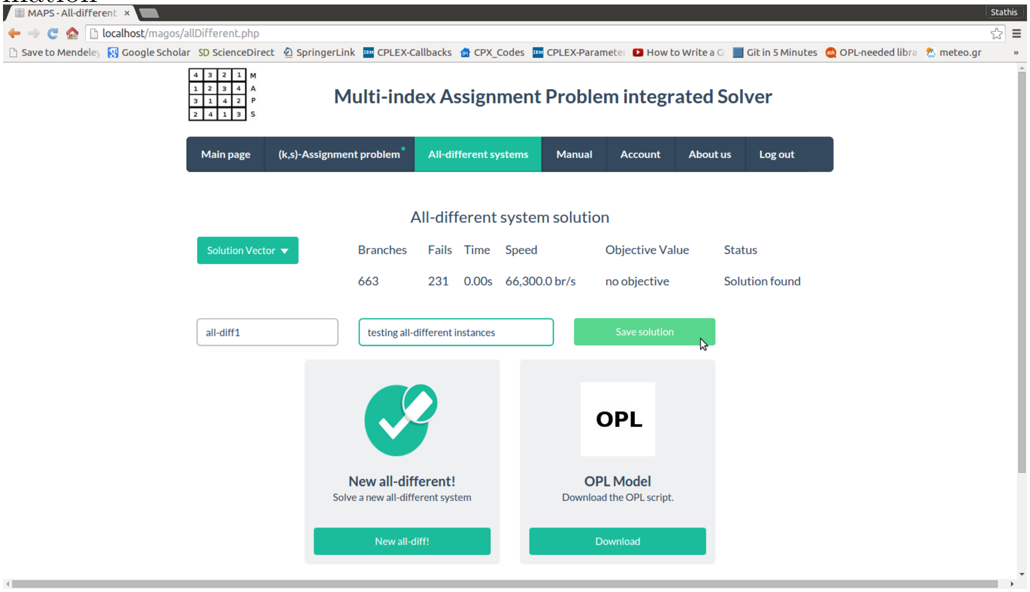
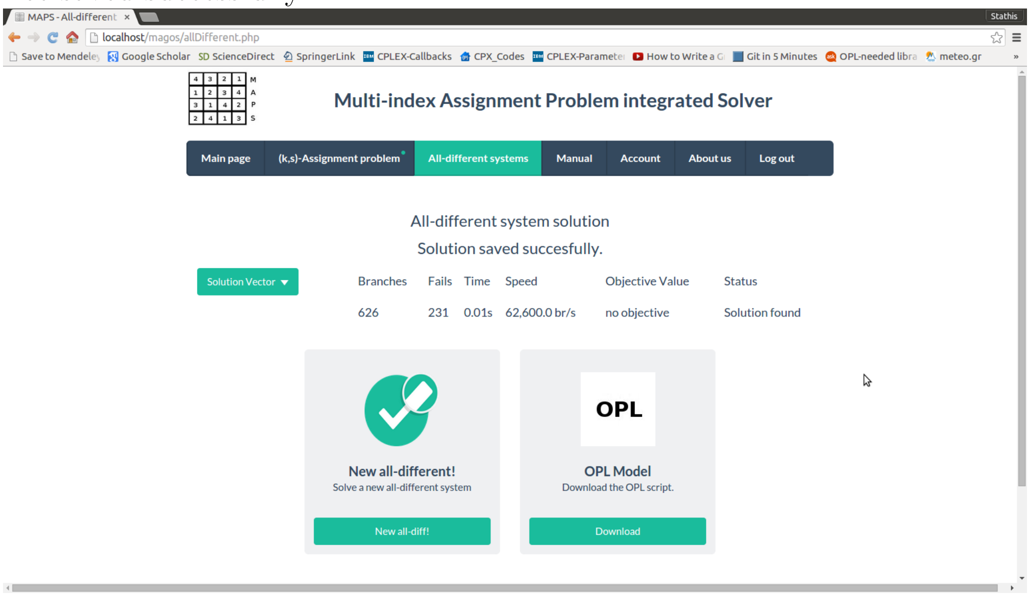


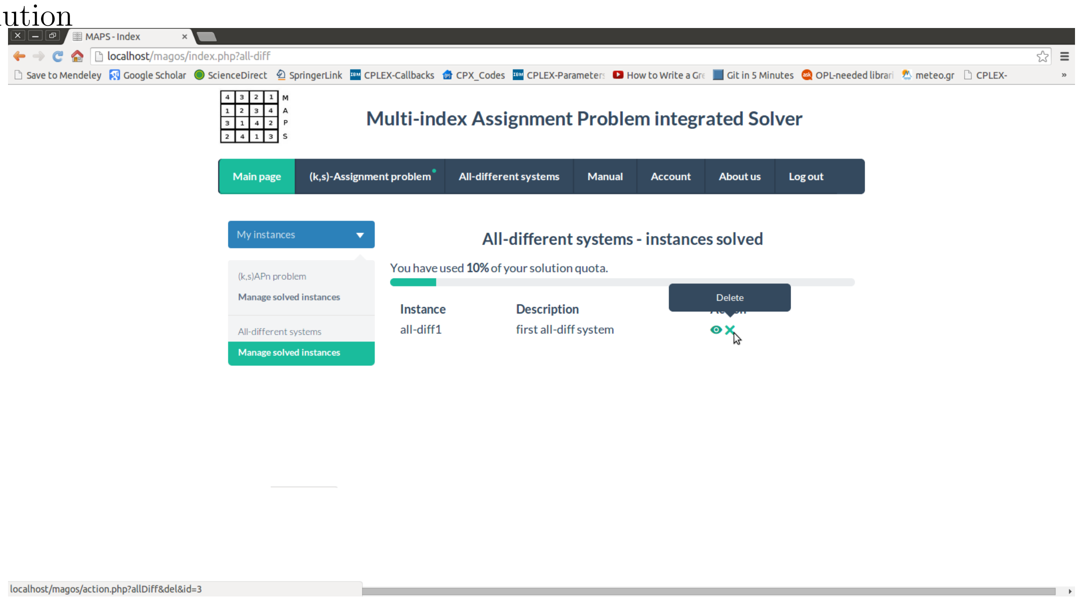
Figure A.20: Use case 10 - Save an *all-different* instance solution: Solution of the instance saved successfully



A.2.7 Delete a saved $all - different$ instance

Once the user has solved and saved an $all - different$ instance he can delete it by going to the main page and selecting the instance he wishes to delete.

Figure A.21: Use case 11 - Delete a saved $all - different$ instance solution: Selecting solution



Once he selects it, the instance is deleted, hence removed from the list of the saved solutions.

A.2.8 View the MAPS manual

The user at any point can view the manual of MAPS by selecting it in the main Menu.

A.2.9 Edit the user account

Once the user has selected to edit his account he views the following screen where he can alter some of his personal information. The fields that are editable are related with the

- password: has to be at least 8 characters;
- first and last name: each one has to be at least 3 characters;
- email: must be of the form *someone@somewhere.something*;
- his personal web-page: must follow the template of a valid URL.

Figure A.22: Use case 8 - Delete a saved $(k, s)AP_n$ instance solution: Solution deleted

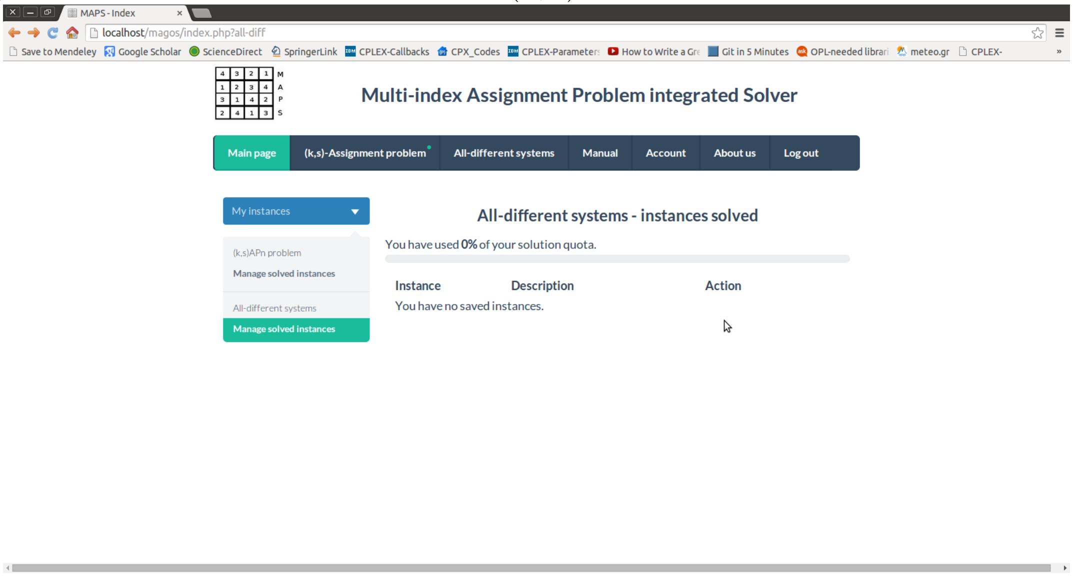


Figure A.23: Use case 12 - View the MAPS manual: Selecting the manual

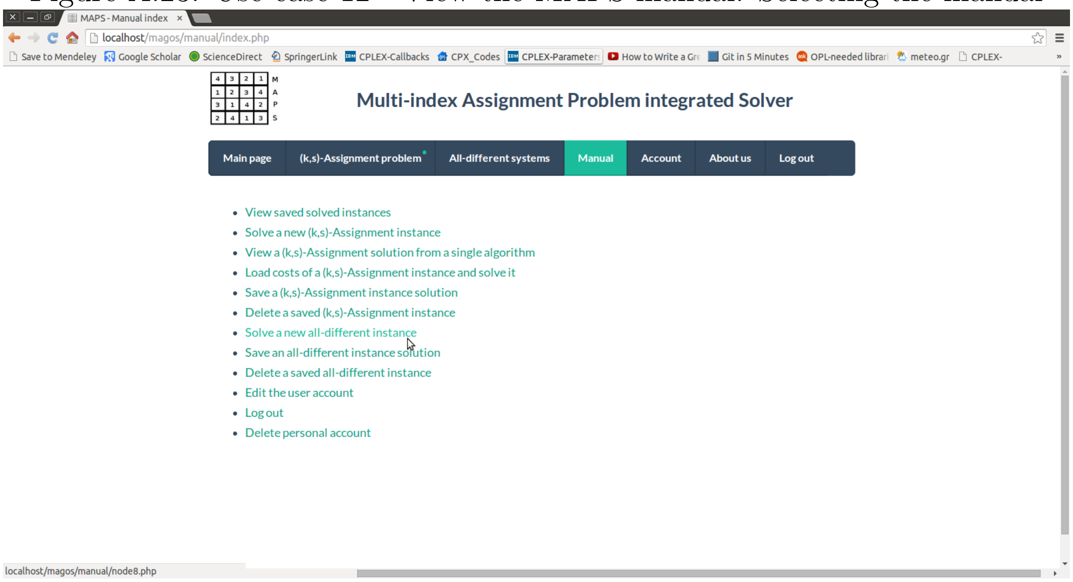


Figure A.24: Use case 12 - View the MAPS manual: Viewing the manual

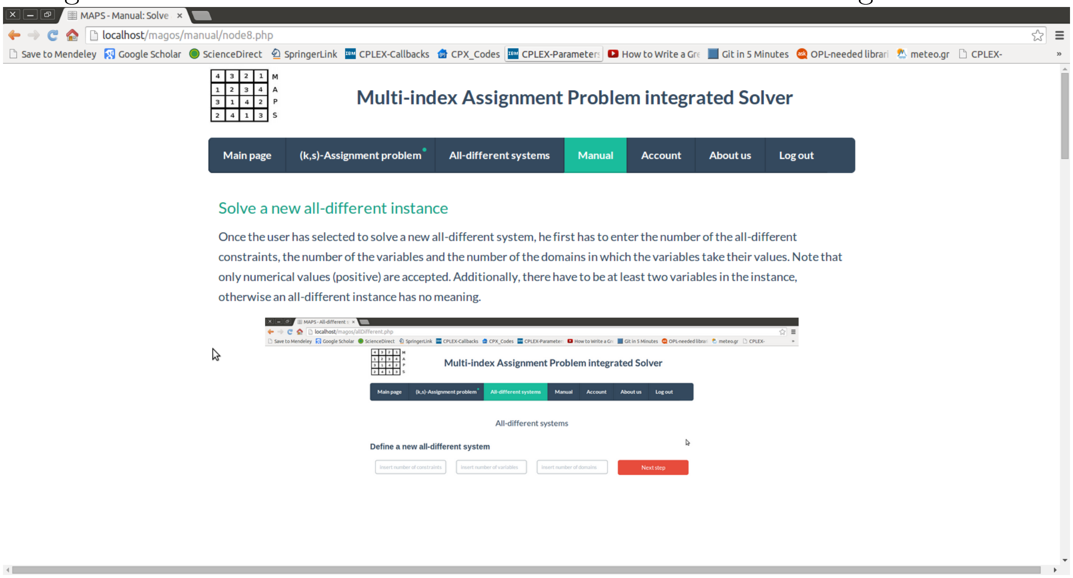


Figure A.25: Use case 12 - View the MAPS manual: Going to the top

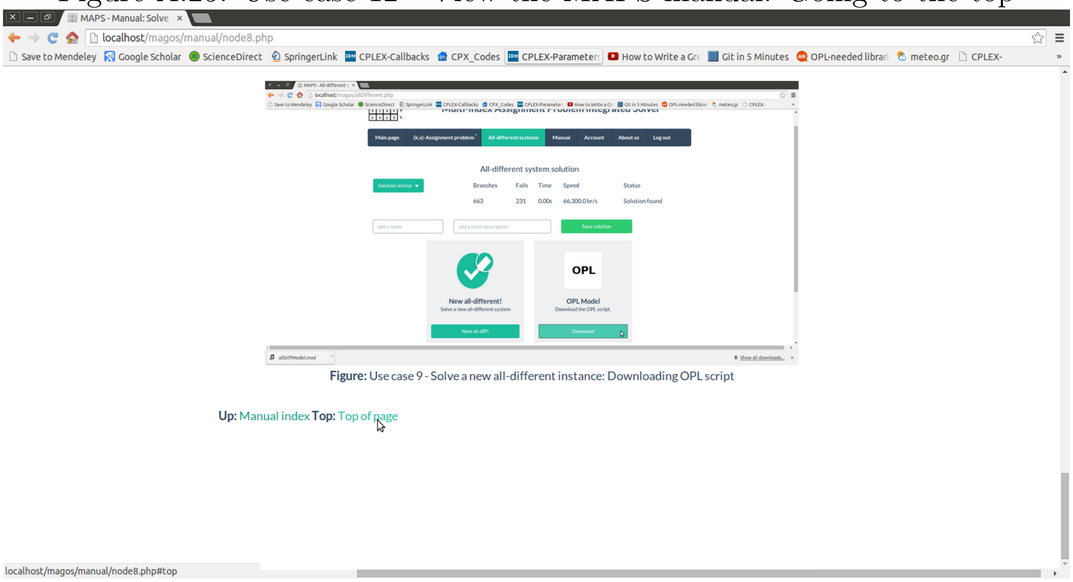
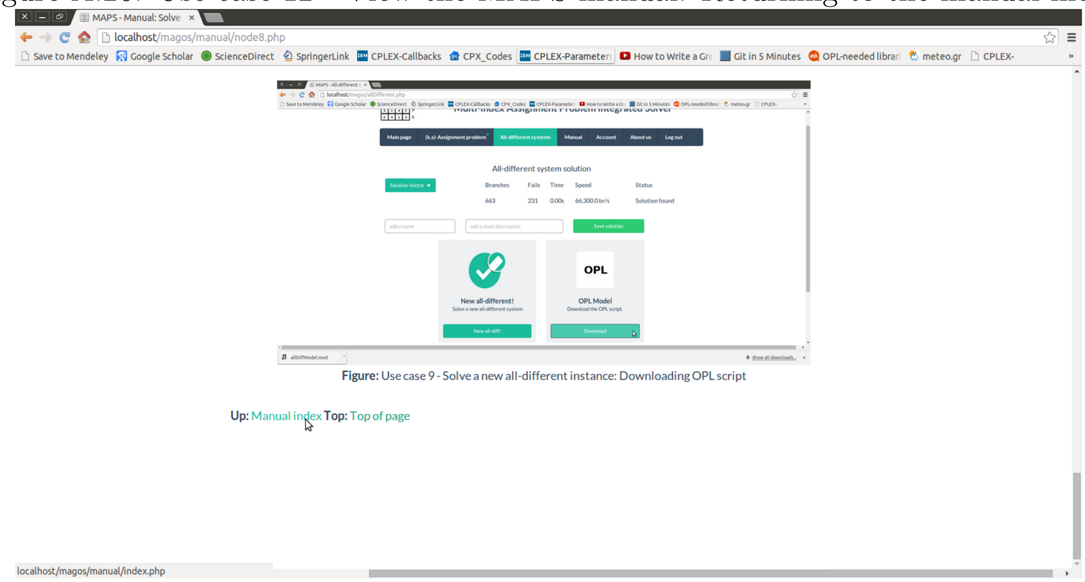
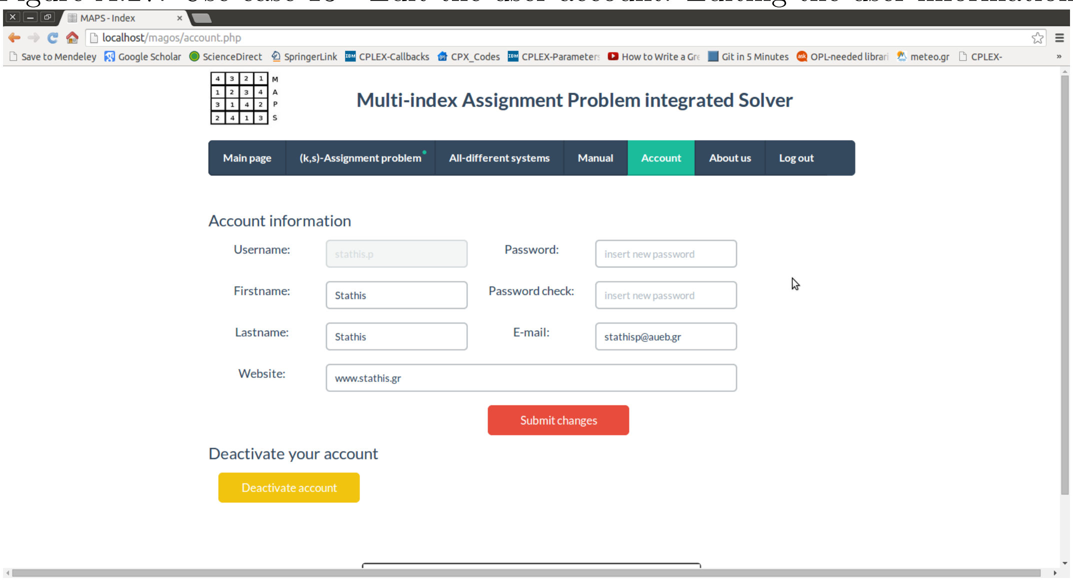


Figure A.26: Use case 12 - View the MAPS manual: Returning to the manual index



If any piece of the required information is not correct, the user is prompted to correct it.

Figure A.27: Use case 13 - Edit the user account: Editing the user information



A.2.10 Log out

The user can at any point log out from the solver. Please note that when that happens any unsaved work is lost. When the user selects to log out from the main menu he is automatically directed to the log in screen.

Figure A.28: Use case 13 - Edit the user account: Error in entry

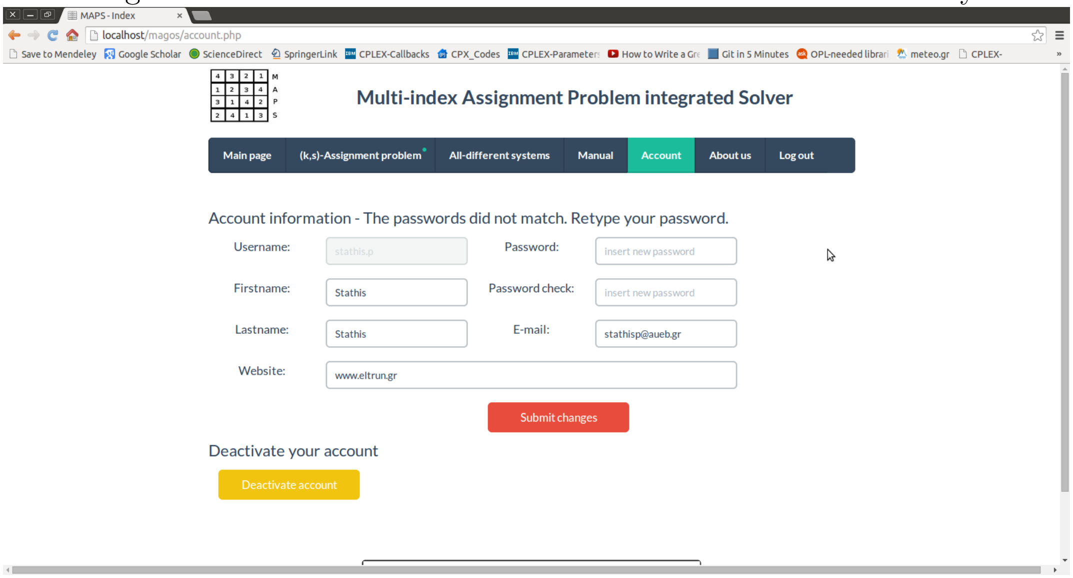


Figure A.29: Use case 13 - Edit the user account: User account info updated

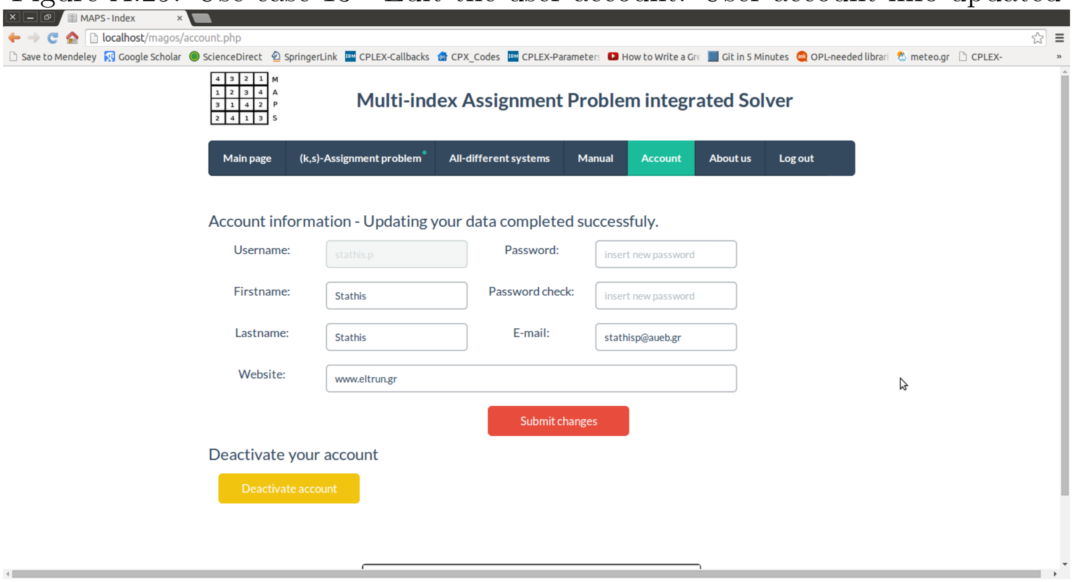


Figure A.30: Use case 14 - Log out: Selecting to log out

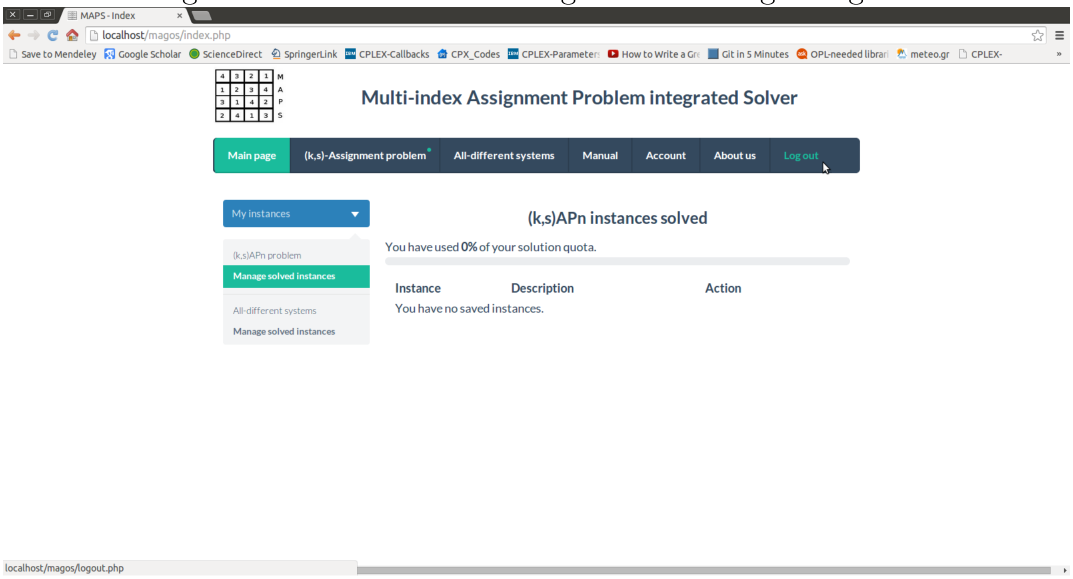
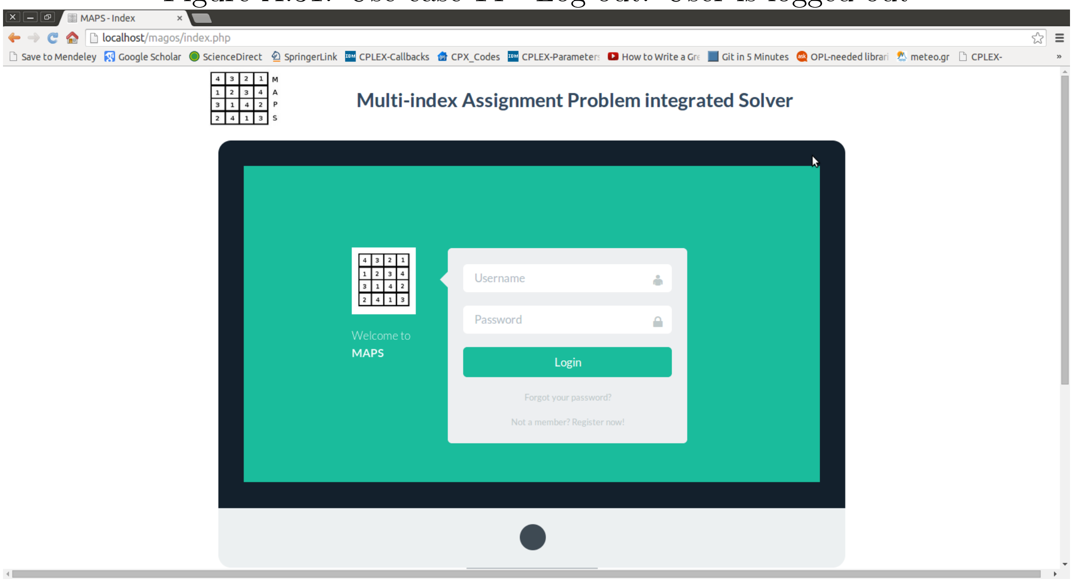


Figure A.31: Use case 14 - Log out: User is logged out



A.2.11 Delete personal account

The user can delete his personal account also from the ‘Edit user account’ screen. Once he selects to delete the account he is prompted to confirm his selection. Please note that if he does so, all saved and unsaved work will be lost. After this selection the user is automatically logged out and redirected to the log in screen.

Figure A.32: Use case 15 - Delete personal account: Selecting to delete personal account

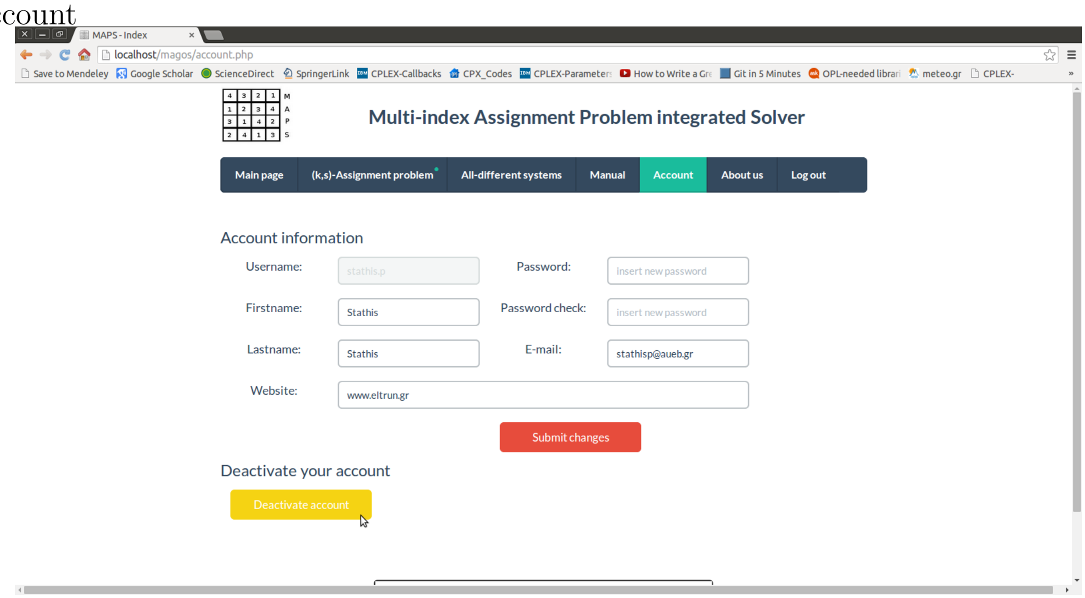
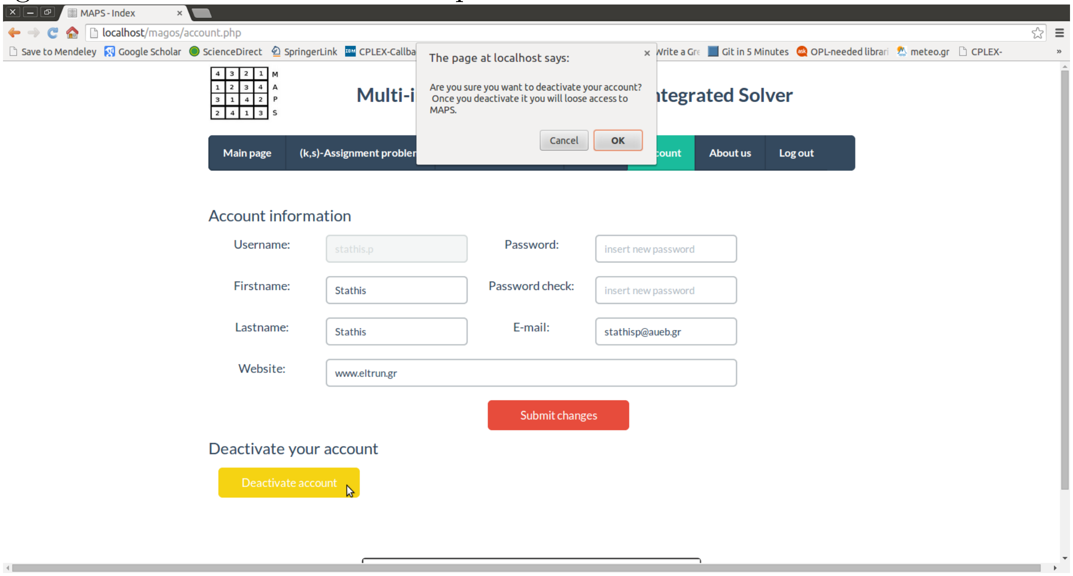


Figure A.33: Use case 15 - Delete personal account: Confirm deletion of account



Bibliography

[1] Artisan project. http://www.artisan-project.eu/.

[2] Or-library. http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

[3] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

[4] Sharlene M Andrijich and Louis Caccetta. Solving the multisensor data association problem. *Nonlinear Analysis: Theory, Methods & Applications*, 47(8):5525–5536, 2001.

[5] G. Appa, D. Magos, and I. Mourtos. Lp relaxations of multiple all_different predicates. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 364–369. Springer, 2004.

[6] G. Appa, D. Magos, and I. Mourtos. On multi-index assignment polytopes. *Linear Algebra and its Applications*, 416(2-3):224–241, 2006.

[7] Gautam Appa, Dimitris Magos, and Ioannis Mourtos. Searching for mutually orthogonal latin squares via integer and constraint programming. *European Journal of Operational Research*, 173(2):519–530, 2006.

[8] Gautam Appa, Dimitris Magos, Ioannis Mourtos, and Jeannette CM Janssen. On the orthogonal latin squares polytope. *Discrete Mathematics*, 306(2):171–187, 2006.

[9] Gautam M Appa, Leonidas S Pitsoulis, and H Paul Williams. *Handbook on modelling for discrete optimization*, volume 88. Springer science & business media, 2006.

[10] Christian Artigues, Pierre Lopez, and Alain Haït. The energy scheduling problem: Industrial case-study and constraint propagation techniques. *International Journal of Production Economics*, 143(1):13–23, 2013.

[11] Egon Balas and Liqun Qi. Linear-time separation algorithms for the three-index assignment polytope. *Discrete Applied Mathematics*, 43(1):1–12, 1993.

[12] Egon Balas and Matthew J Saltzman. Facets of the three-index assignment polytope. *Discrete Applied Mathematics*, 23(3):201–229, 1989.

[13] Egon Balas and Matthew J Saltzman. An algorithm for the three-index assignment problem. *Operations Research*, 39(1):150–161, 1991.

[14] Egon Balas and Eitan Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34(1):119–148, 1978.

[15] Hans-Jürgen Bandelt, Yves Crama, and Frits CR Spieksma. Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49(1):25–50, 1994.

[16] ZM Bi and Lihui Wang. Optimization of machining processes from the perspective of energy consumption: A case study. *Journal of Manufacturing Systems*, 31(4):420–428, 2012.

[17] Pravesh Biyani, Xiaolin Wu, and Abhijit Sinha. Joint classification and pairing of human chromosomes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(2):102–109, 2005.

[18] Natashia L Boland, Andrew C Eberhard, F Engineer, and Angelos Tsoukalas. A new approach to the feasibility pump in mixed integer programming. *SIAM Journal on Optimization*, 22(3):831–861, 2012.

[19] Natashia L Boland, Andrew C Eberhard, Faramroze G Engineer, Matteo Fischetti, Martin WP Savelsbergh, and Angelos Tsoukalas. Boosting the feasibility pump. *Mathematical Programming Computation*, 6(3):255–279, 2014.

[20] Vincent Boyer, Moussa Elkihel, and Didier El Baz. Heuristics for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 199(3):658–664, 2009.

[21] K. Bunse, M. Vodicka, P. Schönsleben, M. Brülhart, and F. O. Ernst. Integrating energy efficiency performance in production management–gap analysis between industrial needs and scientific literature. *Journal of Cleaner Production*, 19(6):667–679, 2011.

[22] Rainer E Burkard, Rüdiger Rudolf, and Gerhard J Woeginger. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65(1):123–139, 1996.

[23] Antonella Certa, Mario Enea, Giacomo Galante, and Concetta Manuela La Fata. Multi-objective human resources allocation in r&d projects planning. *International Journal of Production Research*, 47(13):3503–3523, 2009.

[24] Paul C Chu and John E Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17–23, 1997.

[25] CPLEX. Ibm-ilog-cplex manual, 2013.

[26] Yves Crama and Frits C.R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60(3):273 – 279, 1992.

[27] Harlan Crowder, Ellis L Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[28] H. LF De Groot, E. T. Verhoef, and P. Nijkamp. Energy saving by firms: decision-making, barriers and policies. *Energy Economics*, 23(6):717–740, 2001.

[29] M De Santis, S Lucidi, and F Rinaldi. Feasibility pump-like heuristics for mixed integer problems. *Discrete Applied Mathematics*, 165:152–167, 2014.

[30] Marianna De Santis, Stefano Lucidi, and Francesco Rinaldi. A new class of functions for measuring solution integrality in the feasibility pump approach. *SIAM Journal on Optimization*, 23(3):1575–1606, 2013.

[31] Tom Devoldere, Wim Dewulf, Wim Deprez, Barbara Willems, and Joost R Duflou. Improvement potential for energy consumption in discrete part production machines. In *Advances in Life Cycle Engineering for Sustainable Manufacturing Businesses*, pages 311–316. Springer, 2007.

[32] SA Dichkovskaya and Mikhail Konstantinovich Kravtsov. Investigation of polynomial algorithms for solving the three-index planar assignment problem. *Computational Mathematics and Mathematical Physics*, 46(2):212–217, 2006.

[33] Trivikram Dokka, Yves Crama, and Frits CR Spieksma. Multi-dimensional vector assignment problems. *Discrete Optimization*, 14:111–125, 2014.

[34] I. Dumitrescu, S. Ropke, JF Cordeau, and G. Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305, 2010.

[35] ECR-Europe. Ecr europe blue book-using traceability in the supply chain to meet consumer safety expectations, 2004.

[36] Leonhard Euler. *Recherches sur une nouvelle espece de quarres magiques*. Zeeuwsch Genootschao, 1782.

[37] Reinhardt Euler, Rainer E Burkard, and R Grommes. On latin squares and the facial structure of related polytopes. *Discrete Mathematics*, 62(2):155–181, 1986.

[38] Kan Fang, Nelson Uhan, Fu Zhao, and John W Sutherland. A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. *Journal of Manufacturing Systems*, 30(4):234–240, 2011.

[39] Ross R Farrell and Thomas C Maness. A relational database approach to a linear programming-based decision support system for production planning in secondary wood product manufacturing. *Decision Support Systems*, 40(2):183–196, 2005.

[40] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[41] M. Fischetti and D. Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1(2-3):201–222, 2009.

[42] Arnaud Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

[43] Arnaud Fréville and SaÏd Hanafi. The multidimensional 0-1 knapsack problem-bounds and computational aspects. *Annals of Operations Research*, 139(1):195–227, 2005.

[44] AM Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161–164, 1983.

[45] Armin Fügenschuh and Benjamin Höfler. Parametrized grasp heuristics for three-index assignment. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 61–72. Springer, 2006.

[46] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 58, 1979.

[47] Don A Grundel and Panos M Pardalos. Test problem generator for the multidimensional assignment problem. *Computational Optimization and Applications*, 30(2):133–146, 2005.

[48] Zonghao Gu, George L Nemhauser, and Martin WP Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437, 1998.

[49] Zonghao Gu, George L Nemhauser, and Martin WP Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, 4(1):109–129, 2000.

[50] Zonghao Gu, George L Nemhauser, and Martin WP Savelsbergh. Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, 4(1):109–129, 2000.

[51] Saïd Hanafi and Christophe Wilbaut. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, 183(1):125–142, 2011.

[52] P Hansen and L Kaufman. A primal-dual algorithm for the three-dimensional assignment problem. *Cahiers du CERO*, 15:327–336, 1973.

[53] I. Harjunkoski, C. T Maravelias, P. Bongers, P. M Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, and J. Wassick. Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, 62:161–193, 2014.

[54] Michael James Higgins. *Applications of Integer Programming Methods to Solve Statistical Problems*. PhD thesis, University of California, Berkeley, 2013.

[55] J. N. Hooker. *Integrated methods for optimization*, volume 100. Springer, 2012.

[56] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[57] Konstantinos Kaparis and Adam N Letchford. Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem. *European journal of operational research*, 186(1):91–103, 2008.

[58] Konstantinos Kaparis and Adam N Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical programming*, 124(1):69–91, 2010.

[59] Daniel Karapetyan and Gregory Gutin. Local search heuristics for the multidimensional assignment problem. *Journal of Heuristics*, 17(3):201–249, 2011.

[60] Stamatis Karnouskos, Armando Walter Colombo, Jose L Martinez Lastra, and Corina Popescu. Towards the energy efficient future factory. In *Industrial Informatics, 7th IEEE International Conference*, pages 367–371. IEEE, 2009.

[61] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[62] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack problems. 2004.

[63] Bum-Jin Kim, William L Hightower, Peter M Hahn, Yi-Rong Zhu, and Lu Sun. Lower bounds for the axial three-index assignment problem. *European Journal of Operational Research*, 202(3):654–668, 2010.

[64] Xiangyong Kong, Liqun Gao, Haibin Ouyang, and Steven Li. Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers & Operations Research*, 63:7–22, 2015.

[65] CF Laywine and GL Mullen. *Discrete mathematics using latin squares*. Wiley, New York, 1998.

[66] Adam N Letchford and Andrea Lodi. Strengthening chvátal–gomory cuts and gomory fractional cuts. *Operations Research Letters*, 30(2):74–82, 2002.

[67] Bernhard Lienland and Li Zeng. A review and comparison of genetic algorithms for the 0-1 multidimensional knapsack problem. *International Journal of Operations Research and Information Systems (IJORIS)*, 6(2):21–31, 2015.

[68] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. *Iterated local search*. International Series in Operations Research & Management Science. Springer, 2003.

[69] J. Lysgaard, A. N. Letchford, and R. W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[70] D Magos. Tabu search for the planar three-index assignment problem. *Journal of Global Optimization*, 8(1):35–48, 1996.

[71] D Magos and P Miliotis. An algorithm for the planar three-index assignment problem. *European Journal of Operational Research*, 77(1):141–153, 1994.

[72] D. Magos and I. Mourtos. Clique facets of the axial and planar assignment polytopes. *Discrete Optimization*, 6(4):394–413, 2009.

[73] D Magos and I Mourtos. A characterization of odd-hole inequalities related to latin squares. *Optimization*, 62(9):1169–1201, 2013.

[74] J. Manget, C. Roche, and F. Münnich. Capturing the green advantage for consumer companies. *The Boston Consulting Group*, pages 1–2, 2009.

[75] Silvano Martello and Paolo Toth. Linear assignment problems. *Annals of Discrete Mathematics*, 31:259–282, 1987.

[76] Alexander Martin and Robert Weismantel. The intersection of knapsack polyhedra and extensions. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 243–256. Springer, 1998.

[77] James L McKenney and Morton M Scott. *Management decision systems: computer-based support for decision making.* Harvard Business School Press, 1971.

[78] L. Mundaca. Markets for energy efficiency: Exploring the implications of an eu-wide tradable white certificate scheme. *Energy Economics*, 30(6):3016–3043, 2008.

[79] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[80] Klaus Neumann and Jürgen Zimmermann. Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research*, 117(3):591–605, 1999.

[81] C. Pach, T. Berger, Y. Sallez, T. Bonte, E. Adam, and D. Trentesaux. Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields. *Computers in Industry*, 65(3):434–448, 2014.

[82] Manfred W Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, 1973.

[83] Cheol-Woo Park, Kye-Si Kwon, Wook-Bae Kim, Byung-Kwon Min, Sung-Jun Park, In-Ha Sung, Young Sik Yoon, Kyung-Soo Lee, Jong-Hang Lee, and Jongwon Seok. Energy consumption reduction technology in manufacturing?a selective review of policies, standards, and research. *International Journal of Precision Engineering and Manufacturing*, 10(5):151–173, 2009.

[84] Eduardo L Pasiliao, Panos M Pardalos, and Leonidas S Pitsoulis. Branch and bound algorithms for the multidimensional assignment problem. *Optimization Methods and Software*, 20(1):127–143, 2005.

[85] Jagat Patel and John W Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, 110(3):445–474, 2007.

[86] KT Phelps. A general product construction for error correcting codes. *SIAM Journal on Algebraic Discrete Methods*, 5(2):224–228, 1984.

[87] William P Pierskalla. The multidimensional assignment problem. *Operations Research*, 16(2):422–431, 1968.

[88] Sharma N Pillutla and Barin N Nag. Object-oriented model construction in production scheduling decisions. *Decision Support Systems*, 18(3):357–375, 1996.

[89] Michael Pinedo. *Scheduling: theory, algorithms and systems*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[90] Stathis Plitsos, Panagiotis P Repoussis, Ioannis Mourtos, and Christos D Tarantilis. Energy-aware decision support for production scheduling. *Decision Support Systems*, 2016.

[91] Aubrey B Poore and Sabino Gadaleta. Some assignment problems arising from multiple target tracking. *Mathematical and Computer Modelling*, 43(9):1074–1091, 2006.

[92] Jakob Puchinger, Günther R Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.

[93] Jean-François Pusztaszeri, Paul E Rensing, and Thomas M Liebling. Tracking elementary particles near their primary vertex: a combinatorial approach. *Journal of Global Optimization*, 9(1):41–64, 1996.

[94] Liqun Qi, Egon Ballas, and Geena Gwan. A new facet class and a polyhedral method for the three-index assignment problem. In *Advances in Optimization and Approximation*, pages 256–274. Springer, 1994.

[95] Markus Rager, Christian Gahm, and Florian Denz. Energy-oriented scheduling based on evolutionary algorithms. *Computers & Operations Research*, 54:218–231, 2015.

[96] Pedro Leite Rocha, Martin Gomez Ravetti, Geraldo Robson Mateus, and M. Panos Pardalos. Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35:1250–1264, 2008.

[97] Nancy Ruiz, Adriana Giret, Vicente Botti, and Victor Feria. An intelligent simulation environment for manufacturing systems. *Computers & Industrial Engineering*, 76:148–168, 2014.

[98] Y Sakamoto, Y Tonooka, and Y Yanagisawa. Estimation of energy consumption for each process in the japanese steel industry: a process analysis. *Energy Conversion and Management*, 40(11):1129–1140, 1999.

[99] Fadi Shrouf, Joaquin Ordieres-Meré, Alvaro García-Sánchez, and Miguel Ortega-Mier. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production*, 67:197–207, 2014.

[100] Leonard H Soicher. Optimal and efficient semi-latin squares. *Journal of Statistical Planning and Inference*, 143(3):573–582, 2013.

[101] Ralph H Sprague Jr. A framework for the development of decision support systems. *MIS quarterly*, pages 1–26, 1980.

[102] Corinne Subai, Pierre Baptiste, and Eric Niel. Scheduling issues for environmentally responsible manufacturing: The case of hoist scheduling in an electroplating line. *International Journal of Production Economics*, 99(1):74–87, 2006.

[103] Mottaqiallah Taouil and Said Hamdioui. Layer redundancy based yield improvement for 3d wafer-to-wafer stacked memories. In *European test symposium (ETS), 2011 16th IEEE*, pages 45–50. IEEE, 2011.

[104] A. Valente, E. Carpanzano, A. Nassehi, and S. T. Newman. A step compliant knowledge based schema to support shop-floor adaptive automation in dynamic manufacturing environments. *CIRP Annals-Manufacturing Technology*, 59(1):441–444, 2010.

[105] K. Vikhorev, R. Greenough, and N. Brown. An advanced energy management framework to promote energy awareness. *Journal of Cleaner Production*, 43:103–112, 2013.

[106] Milan Vlach. Branch and bound method for the 3-index assignment problem. *Ekonomicko-Matematicky Obzor*, 3(2):181–191, 1967.

[107] Jose L Walteros, Chrysafis Vogiatzis, Eduardo L Pasiliao, and Panos M Pardalos. Integer programming models for the multidimensional assignment problem with star costs. *European Journal of Operational Research*, 235(3):553–568, 2014.

[108] Nils Weinert, Stylianos Chiotellis, and Günther Seliger. Methodology for planning and operating energy-efficient production systems. *CIRP Annals-Manufacturing Technology*, 60(1):41–44, 2011.

[109] E. Zampou, S. Plitsos, A. Karagiannaki, and I. Mourtos. Towards a framework for energy-aware information systems in manufacturing. *Computers in Industry*, 65(3):419–433, 2014.

[110] Eitan Zemel. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14(4):760–764, 1989.

[111] G. Zobolas, C.D. Tarantilis, and G.Ioannou. A hybrid evolutionary algorithm for the job shop scheduling problem. *Journal of the Operational Research Society*, 60:221–235, 2009.