Data Mining for Enhanced Operations Management Decision Making: Applications in Health Care

Michele Samorani
University of Colorado at Boulder, michael.samorani@colorado.edu

Follow this and additional works at: https://scholar.colorado.edu/opim_gradetds

Part of the Artificial Intelligence and Robotics Commons, Health and Medical Administration Commons, and the Operational Research Commons

Recommended Citation
https://scholar.colorado.edu/opim_gradetds/2

This Dissertation is brought to you for free and open access by Operations & Information Management at CU Scholar. It has been accepted for inclusion in Operations & Information Management Graduate Theses & Dissertations by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.
DATA MINING FOR ENHANCED OPERATIONS MANAGEMENT DECISION MAKING: APPLICATIONS IN HEALTH CARE

BY

Michele Samorani
B.Eng., University of Bologna, 2003
M.Eng., University of Bologna, 2006

A Thesis Submitted to the
Faculty of the Graduate School of the
University of Colorado in Partial Fulfillment
of the Requirement for the Degree of
Doctor of Philosophy
Department of Operations and Information Management
2012
This thesis entitled:

Data Mining for Enhanced Operations Management Decision Making:
Applications in Health Care
written by Michele Samorani

has been approved for the department of Operations and Information Management

________________________________________
Manuel Laguna

________________________________________
Thomas Vossen

Date ________________

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.
ABSTRACT

Samorani, Michele (Ph.D., Operations and Information Management)

Data Mining for Enhanced Operations Management Decision Making: Applications in Health Care

Thesis directed by Professor Manuel Laguna

Data Mining involves the extraction of new knowledge from large data sets. Despite the growing research interest in data mining, however, integrating this extra knowledge into the subsequent decision making processes has received little attention. Within the context of operations management, this integration can occur in two different ways: by providing inputs for an optimization procedure and by analyzing the output of an optimization procedure. In this dissertation, I will begin by introducing a database exploration technique, which is used to improve the drug discovery process of a pharmaceutical company (Samorani et al., 2011). The same procedure is also applied to a mental health clinic’s database to predict whether patients will show up at their scheduled appointments. The knowledge obtained with this procedure is then used to improve patient scheduling procedures (Samorani and LaGanga, 2011). I will finally discuss how data mining can be used to learn useful information about the structure of a problem (Samorani and Laguna, 2012).
ACKNOWLEDGMENTS

This work was possible thanks to a few people that guided me through the doctoral program: Professor Manuel Laguna, for teaching me how to conduct good research and write a scientific paper; Professor Fred Glover, who introduced me to academia; Dr. Linda LaGanga, for making it possible for me to work on a real-world project; Professor Kai Larsen, for teaching me how to teach (effectively and efficiently). I would also like to thank Profs. Stephen Lawrence, Thomas Vossen, and Dan Zhang, for their ongoing support, guidance, availability, and continuous help on the job market.

Further thanks go to some fellow PhD students that have been very influential in my academic and personal life: Marilyn Uy, whose incredibly high academic productivity increased my motivation; Markus Fitza, who taught me how to obtain the most with minimum effort; Subhamoy Ganguly, who taught me all I know about Supply Chain Management; Maria Elizabeth (Lila) Saade Saade, who taught me Design of Experiments and how to use Minitab.

A final thank-you goes to my parents and my brother for their help, encouragement, and unconditional support in helping me achieve my academic goals.
TABLE OF CONTENT

1. Introduction ........................................................................................................ 1
   1.1 Literature Review .......................................................................................... 2
   1.2 Automatic Generation of the Mining Table for Molecule Classification .......... 5
   1.3 Using Data Mining to Improve Appointment Scheduling in the Presence of No-Shows 8
   1.4 Using Classification to Enhance Neighborhood Search .................................. 10

2. A Randomized Exhaustive Propositionalization Approach for Molecule Classification __ 14
   2.1 Introduction .................................................................................................... 14
   2.2 Definitions and Terminology .......................................................................... 19
   2.3 Generation of Attribute Descriptors ............................................................... 22
      2.3.1 Finding the paths ...................................................................................... 22
      2.3.2 Aggregations and Refinements ................................................................. 23
   2.4 Attribute Bound Experiments .......................................................................... 32
   2.5 Generating Attributes when Limiting the Computational Time ....................... 37
   2.6 Additional Attribute Selection Strategies ....................................................... 46
   2.7 Conclusions .................................................................................................... 47

3. Scheduling Appointments in a Multi-Day Scheduling Horizon given Individual Show Probabilities ............................................................ 50
   3.1 Introduction .................................................................................................... 50
   3.2 Clinic Model and Problem Definition .............................................................. 53
   3.3 Appointment Request Scheduling Procedure ................................................... 57
   3.4 Solving the One Day Scheduling Problem ....................................................... 62
   3.5 Experimental Results ...................................................................................... 64
      3.5.1 Full Factorial Design ............................................................................... 66
      3.5.2 A real world example ............................................................................. 70
      3.5.3 Variable Service Times ........................................................................... 75
      3.5.4 Implementation of a Simple Scheduling Policy ........................................ 76
   3.6 Managerial Insights and Conclusions .............................................................. 78

4. Data-Mining-Driven Neighborhood Search .............................................................. 81
   4.1 Introduction .................................................................................................... 81
   4.2 Learning Procedure ......................................................................................... 85
   4.3 Escaping Local Optimality ............................................................................. 95
   4.4 Constrained Task Allocation Problem ........................................................... 99
   4.5 Computational Experiments on the CTAP ....................................................... 107
      4.5.1 Experiment 1 .......................................................................................... 107
      4.5.2 Experiment 2 .......................................................................................... 112
      4.5.3 Experiment 3 .......................................................................................... 117
TABLES

Table 1. For each aggregating function, the type and dimension of the input attributes and output attributes................................................................. 26
Table 2. Refinement operators compatible with each type of Ar........................................ 28
Table 3. Cumulative number of attributes and time to compute one attribute per compound .... 31
Table 4. Average and best individual CV accuracy on the UD tests........................................ 35
Table 5. Comparison between UD-3 and RAND .................................................................. 36
Table 6. Average and best individual CV accuracy on RAND tests ...................................... 36
Table 7. Results on Estrogen ............................................................................................. 38
Table 8. Results on Mutagenesis ...................................................................................... 38
Table 9. Average and best individual CV accuracy on time-limit tests .................................. 38
Table 10. Results obtained by the different molecule representations .................................. 42
Table 11: Notation ............................................................................................................. 56
Table 12. Show rates considered ....................................................................................... 65
Table 13: Factors of the full factorial .................................................................................. 67
Table 14. Best results for each combination of the policies “data mining” (DM), “overbooking “(OB), “same-day scheduling” (SD). ....................................................... 74
Table 15: Profit change due to service time variability ....................................................... 76
Table 16: Target complete sequence of shows and no-shows (sn-sequences) for different combinations of capacity N and appointments booked K ........................................ 77
Table 17: Summary of results of Experiment 1 .................................................................... 110
Table 18: Comparison with results reported in the literature ............................................. 115
Table 19: New best-solutions for 13 CTAP problem instances ........................................ 116
Table 20: Comparison of five solution procedures on eight real-world CTAP instances. ...... 118
Table 21: Steps to generate the attribute .......................................................................... 134
Table 22: Results of sensitivity analysis ............................................................................ 151
FIGURES

Figure 1: Existing research in Data Mining ................................................................. 3
Figure 2. Types Graph for the molecule classification problem ........................................ 21
Figure 3. Pseudo code of the Roll-Up procedure .......................................................... 23
Figure 4. Pseudo code of the GenerateNextDerivedAttribute procedure ....................... 24
Figure 5. Example of attachment .................................................................................. 25
Figure 6. Average accuracy for UD-1, UD-2, UD-3 and UD-4 ......................................... 34
Figure 7. Graphical representations of the results in Tables 7 and 8 ................................. 39
Figure 8. A) Mutagenic compound benz(a)anthracene containing four, fused aromatic rings and exclusively double bonds. B) Mutagenic compound 2-amino-dinitro-phenol, exhibiting two nitro groups. C) Flavone, non-estrogen receptor binding. D) 6-hydroxyflavone, an estrogen receptor binding compound. E) 7-hydroxyflavone, non-estrogen receptor binding ........... 46
Figure 9: Appointment Request Scheduling Procedure .................................................. 57
Figure 10: Impact of the controllable parameters ............................................................ 69
Figure 11: Interactions of sn and sp with other factors ..................................................... 70
Figure 12: Simple Scheduling Policy ............................................................................ 78
Figure 13: Outline of the escape procedure ..................................................................... 96
Figure 14: Data-Mining-Driven Tabu Search ................................................................ 99
Figure 15: Process to test the effectiveness of the escape procedure ............................... 108
Figure 16: DMDTS scores for 50 GRASP iterations ....................................................... 113
Figure 17: DMDTS scores for 50 repetitions .................................................................. 123
Figure 18. Graphical representation of extended example ............................................. 136
Figure 19: Assignment with diagonal structure (a), and without diagonal structure (b) ........ 139
Figure 20: Assignment corresponding to V = (3, 1, 0, 3) ............................................... 142
Figure 21: Assignment corresponding to V = (3, 1, 0, 3) ............................................... 143
Figure 22: Assignment corresponding to V = (3, 1, 0, 3) ............................................... 144
1. INTRODUCTION

Starting from the ‘70s, the increase of the capacity of storage media has resulted in the availability of an enormous amount of information. Although SQL (Beaulieu 2009) allows the retrieval of basic data, this popular programming language fails at individuating high-level information that may be very valuable to decision makers. During the ‘80s and ‘90s, the need of finding patterns in data determined the quick growth of a new field called “data mining”, which precisely aims at extracting patterns from a data set. Obviously, with the knowledge of a valuable pattern, a company can make better decisions than the competitors, and may gain competitive advantage. For example, if the data set of a grocery store showed that “who buys product A, buys also product B”, the store manager could decide to place B close to A in the shelves, in order to maximize its sells. The pattern in this example is found solving a data mining problem called “association rules” (Witten and Frank 2005). There are four classes of data mining problems: “association rules”, “clustering”, “classification”, and “regression”. Of these, “classification” is the most popular and widely studied. It consists of predicting the group membership of an object, given a training set of objects whose group memberships are known. The training set can be represented as a “mining table”, whose rows correspond to the objects and whose columns are their attributes, i.e. the characteristics of the objects. One of these attributes is the “target attribute”, which is a label indicating the group membership. In the training phase, the mining table is analyzed in order to “learn” the classification rule that explains the difference between groups; then, the rule is used to predict the target attribute of a new object, given all its other attributes. The importance of this problem derives from its application to various fields, such as diagnosis, quality control, credit card approval, and fraud
detection. For example, it is possible to diagnose if a patient is sick or healthy from the result of a certain medical test, given the historical results of sick and healthy patients.

1.1 Literature Review

We consider the articles published in the INFORMS journals from 1998 to 2011 that contain the expression “data mining”. First, we exclude the surveys, the articles on web search (this dissertation does not consider unstructured data, such as text, images, etc…), and the articles for which data mining is a very marginal aspect or is casually mentioned. Then, for each article, we record its type and the data mining algorithm that it studies or employs. Articles are classified into three categories: “methodology”, “empirical”, and “analytics”. Methodology papers use mathematical techniques to design a data mining algorithm; for example, Street (2005) proposes a nonlinear-programming-based approach to derive a multi-category decision tree. Empirical papers study the impact of data mining on an aspect of an organization or a company; for example, Fleder and Hosanagar (2009) study the impact of recommender systems (such as Amazon’s) on sales diversity. Analytics papers integrate data mining techniques in the decision making process. Unlike methodology papers, the focus of analytics papers is on the application rather than on the data mining technique; for example, Gal-Or et al. (2006) aim at “understanding the extent to which an advertiser should allocate resources to increase the quality of its targeting”. In this paper, the customers that are most likely to respond to advertisements are retrieved via data mining, and an optimization procedure determines how many to target.
As shown in Figure 1, most of the research on classification has focused on methodology, while very little has focused on the integration of data mining in the decision making process. As shown in Figure 1, while 43 papers on classification belong to the “Methodology” group, only 12 belong to the “Analytics” group. In other words, the effort is on improving the accuracy (i.e., the reliability of the prediction) of classification methods, resulting in the definition of various classification techniques, such as decision trees, Bayesian networks, and neural networks. In practice, most works aim at making the training phase more intelligent so that more reliable predictions can be obtained later. If, on one hand, this trend has produced excellent classification techniques, on the other hand, little effort has been done to integrate this “intelligence” within the business environment where it is used. Therefore, the existing literature presents two gaps, corresponding to the input and the output of the classification task.

On the input side, almost the totality of works assumes that the mining table is given. Suppose that a company is interested in classifying “good” and “bad” customers, in order to predict whether new potential customers will be valuable. This information could be used, for
example, to capture them by offering deals. Suppose also that the real difference between “good” and “bad” customers is simply determined by their age. For example, suppose that all customers in the database that are older than 50 are good customers, while those who are younger than 50 are bad customers. This is the classification rule that we want to find; but, if the attribute “Customer Age” is not included in the mining table, this rule will never be found, no matter how clever the classification technique is. This example suggests that the accuracy of a classification technique strongly depends on the attributes used, which, as a general rule, should contain as much information as possible. If the mining table is not given, then it has to be built from the tables of a database. This critical task has been largely neglected in the existing work. The first paper of the dissertation (Samorani et al. 2010) addresses this problem and applies the methodology developed in the context of Molecule Classification. The results show that an automatic construction of the mining table leads to a higher classification accuracy than a manual construction performed by experts in the field of biology and bioinformatics.

The second, more important aspect that the literature fails to satisfactorily address is how to integrate the output of classification with the operations of an organization. By using classification, an organization gains new (uncertain) information: the predicted group membership of objects. Little effort has been done to individuate the business opportunities that arise from this newly available piece of information. The second and third paper of the dissertation use the classifier’s prediction to set up an optimization problem that aims at improving operations. The second paper (Samorani and LaGanga 2011b) considers the clinic outpatient scheduling problem, and shows how to exploit the prediction on whether patients will show or not for appointments in order to better schedule them. The third paper (Samorani and
Laguna 2011) studies how to use classification to improve the performance of any neighborhood search procedure, by finding better search directions to escape local optimality.

1.2 Automatic Generation of the Mining Table for Molecule Classification

Samorani et al. (2010) addresses the research opportunity individuated above regarding the construction of the mining table for an important problem in bioinformatics, the molecule classification problem, which consists of classifying 2 groups of molecules that exhibit different behavior in order to predict the group to which a new molecule belongs. This problem arises in a variety of situations, especially during the different phases of a drug design process, where it is needed to distinguish toxic from non-toxic molecules, active from non-active molecules, and so on. Weaver (2004) provides a fairly comprehensive overview of the role played by molecule classification within a drug design process.

In the mining table for this problem, rows are molecules and the columns are predefined attributes that describe them. One of the most commonly and successfully used molecular representations is the binary fingerprint. Molecular fingerprints consist of a vector of binary digits which represent the presence or absence of a particular molecular fragment within the molecule. A variation of molecular fingerprint is the MACCS Keys, which are a collection of pre-existing molecular substructures (that have presumably been deemed ‘interesting’ or ‘useful’), each on-bit identifies that fragment as existing within the structure in question (Durant et al. 2002). An alternative approach, called “Daylight”, consists of generating a compendium of molecular fragments from the tested datasets (independently). The simplest method of fragment generation consists of enumeration of unique paths (unique in atom type and connectivity) up to a maximum length within the set of molecules followed by a hash function to assign bits within a
fingerprint of desired length. However, many variations of molecular fingerprinting exist (Hert at al. 2004).

All these representation techniques suffer from the same fundamental drawback: they consider only predefined characteristics or methods of computation and ignore others outside a predefined scope. This leads to two major limitations, one on the accuracy and one on the knowledge discovery. First, King et al. (2001) noted that any predefined attribute representation causes some loss of information because “all the information about a particular example is forced into a single row of a table”, which reduces the upper bound of the classification accuracy. Second, the classification rules cannot involve characteristics other than the ones included in the chosen representation. In other words, a traditional data mining process is incapable of generating attributes that were not included in the data set, limiting the knowledge discovery to expressing the classification rule in terms of a logical expression that involves the predefined attributes.

Relational learning techniques do not have these shortcomings. Instead of starting from a single table, they consider a database and search for patterns that involve more tables. A compounds database, for example, consists of a table containing the molecules, a table containing the atoms, and a table containing the bonds, connected to each other through foreign key (FK) relationships. The patterns found are represented by queries performed on a subset of tables. In practice, these techniques search for classification patterns in the space of the queries, by finding the ones that yield to the best classification accuracy. In this way, they potentially use all information contained in the database and find new patterns that have not been explicitly considered.
There are two families of relational learning techniques: multi relational decision trees (MRDTs) and Propositionalization. In the former, the data set is iteratively split by successively adding refinements, creating a decision tree — as in Atramentov et al. (2003) — where each leaf corresponds to the attribute needed to classify a subset of molecules. In the latter, the database navigation leads to the generation of a set of attributes used to create a tabular data set on which the data mining task is eventually executed. In practice, the goal of Propositionalization techniques is only the generation of new features, after which the classification task is executed by a given classifier.

MRDTs tend to find patterns corresponding to local optima because the refinements are built in a greedy fashion, though variations have been recently proposed by Serrurier and Prade (2008). On the other hand, Propositionalization techniques overcome this problem by generating as many attributes as possible without evaluating their utility for the classification, until they have all been generated. The separation between feature generation and classification highlights a practical advantage of Propositionalization over MRDTs, namely, it allows for the application of any classification technique. This makes it possible to exploit all the existing work that has been done in this field. Using well studied classifiers accelerates the implementation and increases the classification performance.

On the other hand, MRDTs have an important advantage over Propositionalization: the feature space of MRDT techniques is larger. In other words, there are classification rules that can be found by MRDTs and not by Propositionalization. The features generated by Propositionalization have two limitations compared to MRDTs: they are less “deep” and less “expressive”. Both “depth” and “expressivity” of a feature depend on the complexity of the query used to generate the feature. As it will be explained, this complexity can be arbitrarily
large, making the feature space infinite. Obviously, the Propositionalization approach needs the definition of a limit in order to stop, while MRDT approaches do not, because the tree will continue growing — generating deeper and more expressive features — as long as the growth increases the accuracy.

In this dissertation, the traditional Propositionalization approach is extended by making it capable of generating deeper and more expressive attributes. These extra attributes result in a reduction in costs and times of the drug design process and in insights on the chemical mechanisms that determine the behavior of the compounds.

1.3 Using Data Mining to Improve Appointment Scheduling in the Presence of No-Shows

Samorani and LaGanga (2011b) shows the impact of using Classification in appointment scheduling. Efficient scheduling of clinic appointments leads to better resource allocation, timely access to healthcare, and ultimately to lower healthcare costs (White, Froehle, and Klassen, 2011). Access to outpatient healthcare services has been the focus of several decades of appointment scheduling research (Gupta and Denton, 2008). As shown by Cayirli and Veral (2003), the failure of patients to show up for appointments has large effects on the performance of scheduling systems. Appropriate clinic scheduling is important in meeting healthcare demand (Green and Savin, 2008). Thus, there has been increased interest in ensuring that all allocated appointment slots are used by managing the prevalent problem of patient no-shows (Murthuraman and Lawley, 2008).

Two recent approaches that address the problem of the no-shows are overbooking (LaGanga and Lawrence, 2007b) and open-access (Liu et al. 2010; Qu et al., 2007; Robinson and Chen, 2010). Using the overbooking technique proposed by LaGanga and Lawrence (2007b),
the time between appointments is compressed so that more patients can be served; the “compression rate” depends on the no-show rate. For example, if the service time is 30 minutes and the no-show rate is 33%, LaGanga and Lawrence (2007b) suggest to schedule an appointment every 20 minutes. Although this technique results in a larger number of patients seen (patient access), it also results in the introduction of patient waiting time and clinic overtime (low clinic punctuality). Therefore, the decision maker should use a compression that correctly balances patient access with clinic punctuality. On the other hand, open-access simply consists in allowing only same-day appointment, which results in a higher patient access because, according to Liu et al. (2010), the no-show rate tends to decrease with the increase of the scheduling gap, defined as the number of days between the appointment request and the appointment day.

Both approaches consider and model varying levels of no-shows but assume every patient and appointment has the same probability of no-show. Other work (Muthuraman and Lawley, 2008; Zeng et al. 2009) explores the properties of an optimal schedule with heterogeneous show probabilities. Glowacka et al. (2009) apply data mining to predict show probabilities, and the decision of which slot is assigned to a patient is based on his/her show probability. All these works share a few drawbacks. First, they consider only a single day where patients can be scheduled, failing to address the important question of how long in advance patients should be scheduled. Second, the form of overbooking they implement is the “double booking”, which offers a less fair balance of patient wait time and provider utilization (LaGanga and Lawrence 2007a) than overbooking by slot compression. Finally, most of these works assume exponential service times, which is far from true in clinics.
In Samorani and LaGanga (2011b), we overcome these drawbacks. Then, by considering the database of a mental health center, we also show that it is possible to use classification and simulation to define a data-driven clinic design process, whose input is composed of the database of the clinic and whose output is the identification of the optimal clinic setup, in terms of the optimal scheduling horizon, the optimal classification technique to use, and whether or not to adopt overbooking.

1.4 Using Classification to Enhance Neighborhood Search

Heuristic and metaheuristic techniques (Glover and Kochenberger 2003) are widely used for finding high-quality solutions to large combinatorial optimization problems for which identifying the optimal solution through an exact procedure would be computationally impractical. Some metaheuristic approaches are based on maintaining a set of solutions and creating new ones by way of executing combination procedures. Others, however, use a neighborhood search (NS) to move from one point to another in the solution space. The neighborhood of a solution is defined by the move mechanisms that are applied to transform one solution into another. Local search is the simplest form of a NS procedure, which limits the moves to those that direct the search from the current solution to a neighboring solution with a better objective function value. In other words, a local search accepts only improving moves to transform the current solution into one that is in its neighborhood. The process stops when no improving move is available and the resulting solution is called a local optimum (with respect to the defined neighborhood).

Metaheuristic procedures include strategies that aim at escaping from local optima with the goal of improving the best solution found during the search. In tabu search, escaping local optima is achieved by imposing tabu restrictions (Glover and Laguna 1997) or executing moves
that are guided by proxy objective functions, as in path relinking (Glover, Laguna, and Marti 2003). Simulated annealing relies on randomization to allow non-improving moves to be executed in order to escape local optimal points. Greedy Randomized Adaptive Search Procedures (GRASP) restart the search after reaching a local optimum. A common aspect to these strategies is that escape directions are determined by rules that are defined a-priori. For instance, tabu activation rules forbid certain moves by restricting a subset of attributes from assuming certain values for a predetermined number of iterations. In path relinking, only moves that take the current solution “closer” to a guiding solution are considered. In general, escaping from local optima in a NS-based approach requires the execution of a non-improving move and the reduction of the neighborhood. Often, the neighborhood reduction is achieved by enforcing guiding constraints. These guiding constraints are the result of design decisions that may or may not be customized to a particular class of problems. For instance, a tabu search approach for permutation problems may use swaps to explore the neighborhood of a given solution. After a swap of two elements, a tabu restriction could be imposed to forbid the exchange of positions of these two elements for a number of pre-specified iterations. This tabu-activation rule effectively restricts the available moves and hence shrinks the neighborhood of the current solution. In Variable Neighborhood Search (VNS), a metaheuristic methodology suggested by Mladenovic and Hansen (1997), escaping from local optima requires the systematic change from one neighborhood to another. This is achieved by embedding a set of move mechanisms of various degrees of complexity. Simple moves are tried first, followed by more complex moves as the search becomes trapped in local optima corresponding to the neighborhood defined by the moves currently under consideration. Both the set of moves (and therefore neighborhoods) to be
explored and the order in which they will be considered are specified a-priori, a task that often requires a considerable effort in the development phase of the solution procedure.

Some work has already been done to couple machine learning and data mining (DM) techniques in meta-heuristic searches, as reported by Baluja et al. (2000) and Jourdan et al. (2006). In particular, the existing learning methods can be classified into two categories: online and offline methods. Both of these approaches are learning procedures that — after a training phase — are used within a search process in an attempt to improve its effectiveness. The main difference between the two approaches lies on the training: online methods are trained using information collected during the current execution of the search procedure, while offline methods learn from a training set of instances of the same class of problems. Online procedures are used, for instance, to select a branch in a branch-and-bound algorithm given information gathered during the search itself, as in (Glover et al. 1989), or to find patterns useful to enhance the construction phase of GRASP, as proposed by Santos et al. (2008). On the other hand, offline procedures attempt to identify structures and patterns shared by the entire class of problems. In the literature, offline approaches have been further divided into two categories, depending on their goal: 1) algorithm-selection approaches and 2) parameter-tuning approaches. Algorithm-selection approaches use historical performance data with the goal of selecting the most promising procedure from a portfolio of available ones. Parameter-tuning approaches have the goal of finding the set of parameter values that will make a particular procedure perform at the highest level.

In this dissertation, we propose an offline approach that cannot be directly cast as an algorithm-selection or parameter-tuning approach. The learning procedure, which is performed offline by considering different instances from the one that we intend to solve, consists of
identifying search directions to enforce during the solution of the target instance, whenever a local optimum is encountered. We argue that our approach defines a new category of offline methods that may be referred to as *data-mining-driven neighborhood search* (DMDNS). This category refers to search procedures that modify neighborhoods with constraints that have been learned offline by mining data. Padmanabhan and Tuzhilin (2003) point out that this area has been largely neglected in the literature.
2. A RANDOMIZED EXHAUSTIVE PROPOSITIONALIZATION APPROACH FOR MOLECULE CLASSIFICATION

In Samorani et al. (2010), we propose an algorithm that automatically builds the mining table needed to solve the Molecule Classification problem.

2.1. Introduction

The molecule classification problem is a data mining problem (Jiawei and Kamber 2000) that consists of classifying 2 groups of molecules that exhibit different behavior in order to predict the group to which a new molecule belongs. This problem arises in a variety of situations, especially during the different phases of a drug design process, where it is needed to distinguish toxic from non toxic molecules, active from non active molecules, and so on. For a fairly comprehensive overview of the role played by molecule classification within a drug design process, see Weaver (2004). Molecule classification may be addressed in two ways: with a traditional classification technique or with a relational learning technique.

Traditional classification techniques, such as decision trees (Rokach and Maimon 2005) and support vector machines (Cristianini and Shawe-Taylor 2000), operate on data sets represented by a single table — where the rows are molecules and the columns are predefined attributes that describe them. While these methods have enjoyed some success (Svetnik et al. 2003), they require a molecule representation that preserves as much information about the molecule as possible and such representation may not be immediately obvious even for someone with expert knowledge.

A great variety of numerical representations of chemical structures exist within the field of Cheminformatics (Todeschini and Consonni 2000), and they can be classified based upon the
degree of structural information required to compute them. At the simplest level, 0-Dimensional descriptors consist of counts of atoms or summations of the properties of atoms present within a compound. These include such values as molecular weight, number of atoms and number of bonds. 1-Dimensional descriptors require a certain degree of connectivity information and typically consist of counts of particular types of fragments (e.g., the number of amide groups or the number of carboxylic acids), or counts of single atoms with particular bonding patterns. 2-Dimensional descriptors have access to the complete connectivity graph of each molecule. A great many descriptors can be computed from this topological representation and include values such as walk and path counts, graph theoretic values such as minimum or maximum eigenvalues, topological fragment indicators or frequencies, and topological separation indices (e.g., the presence/frequency of carbon and chlorine atoms separated by N bonds), among many others. 3-Dimensional descriptors require not only the topological connectivity matrix but additionally the geometric arrangement of atoms within 3-D space. This leads to additional matrix-based calculations as well as geometrically centered mass, charge, and other property indices, as well as geometric separation indices. Considering that many compounds can adopt multiple 3-dimensional conformations, 4-dimensional descriptors attempt to capture the details of multiple, equally viable arrangements of a molecule's atoms. Interestingly, it has been shown that on average, 2-dimensional descriptors perform as well or better than higher dimensional representations calling into question the need to expend computational effort to predict low-energy 3-dimensional structures (Matter and Potter 1999, Dixon and Merz 2001).

One of the most commonly and successfully used molecular representations is the binary fingerprint. Molecular fingerprints consist of a vector of binary digits which represent the presence or absence of a particular molecular fragment within the molecule. A surprisingly
diverse collection of fingerprints exist, two of which are used here. The MACCS Keys are a collection of pre-existing molecular substructures (that have presumably been deemed ‘interesting’ or ‘useful’), each on-bit identifies that fragment as existing within the structure in question (Durant et al. 2002). While being used very commonly and showing reasonable success across applications, a fundamental drawback of this fingerprint type is that the set of fragments is unchanging and may over time lose its useful or interesting character or may fail to include newly identified fragments emerging from the chemistry space. We also tested an alternative approach, which we call “Daylight”, consisting of generating a compendium of molecular fragments from the tested datasets (independently). The simplest method of fragment generation consists of enumeration of unique paths (unique in atom type and connectivity) up to a maximum length within the set of molecules followed by a hash function to assign bits within a fingerprint of desired length. An example of an alternative fingerprint representation is to generate all fragments consisting of atoms and bonds that extend radially from a central atom the same distance up to a defined maximum (Rogers et al. 2005). However, many variations of molecular fingerprinting exist (Hert at al. 2004).

All these representation techniques suffer from the same fundamental drawback: they consider only predefined characteristics or methods of computation and ignore others outside a predefined scope. This leads to two major limitations, one on the accuracy and one on the knowledge discovery. First, King et al. (2001) noted that any predefined attribute representation causes some loss of information because “all the information about a particular example is forced into a single row of a table”, which reduces the upper bound of the classification accuracy. Second, the classification rules cannot involve characteristics other than the ones included in the

---

1 Daylight Chemical Information Systems, Inc. (http://www.daylight.com)
chosen representation. In other words, a traditional data mining process is incapable of generating attributes that were not included in the data set, limiting the knowledge discovery to expressing the classification rule in terms of a logical expression that involves the predefined attributes.

Relational learning techniques do not have these shortcomings. Instead of starting from a single table, they consider a database and search for patterns that involve more tables. A compounds database, for example, consists of a table containing the molecules, a table containing the atoms, and a table containing the bonds, connected to each other through foreign key (FK) relationships. The patterns found are represented by queries performed on a subset of tables. In practice, these techniques search for classification patterns in the space of the queries, by finding the ones that yield to the best classification accuracy. In this way, they potentially use all information contained in the database and find new patterns that have not been explicitly considered.

There are two families of relational learning techniques: multi relational decision trees (MRDTs) and propositionalization. In the former, the data set is iteratively split by successively adding refinements, creating a decision tree — as in Atramentov et al. (2003) — where each leaf corresponds to the attribute needed to classify a subset of molecules. In the latter, the database navigation leads to the generation of a set of attributes used to create a tabular data set on which the data mining task is eventually executed. In practice, the goal of propositionalization techniques is only the generation of new features, after which the classification task is executed by a given classifier.

MRDTs tend to find patterns corresponding to local optima because the refinements are built in a greedy fashion, though variations have been recently proposed by Serrurier and Prade
(2008). On the other hand, propositionalization techniques overcome this problem by generating as many attributes as possible without evaluating their utility for the classification, until they have all been generated. The separation between feature generation and classification highlights a practical advantage of propositionalization over MRDTs, namely, it allows for the application of any classification technique. This makes it possible to exploit all the existing work that has been done in this field. Using well studied classifiers accelerates the implementation and increases the classification performance.

On the other hand, MRDTs have an important advantage over propositionalization: the feature space of MRDT techniques is larger. In other words, there are classification rules that can be found by MRDTs and not by propositionalization. The features generated by propositionalization have two limitations compared to MRDTs: they are less “deep” and less “expressive”. Both “depth” and “expressivity” of a feature depend on the complexity of the query used to generate the feature. As it will be explained, this complexity can be arbitrarily large, making the feature space infinite. Obviously, the propositionalization approach needs the definition of a limit in order to stop, while MRDT approaches do not, because the tree will continue growing — generating deeper and more expressive features — as long as the growth increases the accuracy. These differences, together with examples, will be explored below.

Our work extends the current approaches for propositionalization in two ways. First, we generate “expressive” attributes that are not generated by existing approaches, making our approach an “exhaustive” propositionalization approach, in the sense that it is capable of generating all the attributes that can be found using a SQL query. Although the attribute search space is exhaustive for obvious practical reasons the approach does not generate all attributes. Second, we randomly choose and generate a few “deep” attributes, which are ignored by existing
approaches, making our approach a “randomized” propositionalization approach. The focus is on the feature generation and not on the classification; therefore we use a set of publicly available classifiers and compare the accuracy obtained with different propositionalization approaches. A statistical analysis shows that our extensions significantly improve the performance of the classification. Interestingly, ours is a general purpose method, and can be used to tackle any classification problem, not only molecule classification.

2.2. Definitions and Terminology

The concepts presented in this section are similar to the ones originally introduced by Knobbe et al. (1999), but we prefer to present a simplified terminology that also allows us to generalize the previous approaches.

A *Types Graph* (TG) is a directed graph that describes the types of attributes and associations in the database (DB). The vertices of the TG, called *elements*, correspond to a physical table in the DB, therefore we alternatively use “element” and table (e.g. rows of an element to indicate rows of the table corresponding to that element); the edges, called *associations*, correspond to the relationship between any two tables.

An element is characterized by the attributes contained in the table (i.e. columns), each of which has a type and a dimension. We consider 3 different types of attribute: id, categorical, and numeric; the type of an attribute is important because it determines the aggregate functions that can be applied to that attribute. The dimension, on the other hand, is a string representing the unit of measurement of an attribute, and it determines if two attributes can be compared. For example, if the dimension of attribute $a_1$ is “#atoms”, i.e. number of atoms, and the dimension of attribute $a_2$ is “#bonds”, i.e. number of bonds, any relation between $a_1$ and $a_2$, such as $a_1 > a_2$, is meaningless and therefore prohibited.
We consider two types of associations, $0 - 1$ and $0 - N$ associations. An association connecting element $E_1$ to element $E_2$, which corresponds to a foreign key relationship between $E_1$ and $E_2$, is:

- $0 - 1$ if every row in $E_1$ is associated to at most one row in $E_2$;
- $0 - N$ if every row in $E_1$ can be associated to any number of rows in $E_2$.

Note that, even if associations $N - N$ may be present in the DB, our method forbids them. Therefore, it is necessary to add an extra table substituting the $N - N$ association. Since the details of the DB design phase are outside the scope of our current development, they will not be treated; instead, they will be shown through examples.

A TG always contains a target table, which has one row per observation and only two attributes, the id of the observation and the class to which it belongs. The target table is connected with all the other tables through outgoing $0 - N$ associations. The specific TG that we consider for our molecule classification problem is depicted in Figure 2.
In Figure 2, the Target table contains the compounds, the Atom table the atoms of all the compounds, the Bond table the bonds of all the compounds. Since $N \times N$ associations are not allowed, a table AtomBond is needed that works as a bridge between the Atom and Bond tables. The fields forming the primary key of each table are underlined.

The feature generation procedure considers all paths up to a determined length and, for each of them, generates all possible attribute descriptors. An attribute descriptor (AD) is the description of a new attribute that is added to the target table. It can be viewed as the SQL query that is used to compute the value of the new attribute for all the rows of the target table. In the molecule classification, an example of an AD is:
select a.idTarget, count distinct a.idAtom
from Target t, Atom a
where a.element = "C"
and a.idTarget = t.id
group by a.idTarget

This AD counts the number of atoms of carbon contained in every compound. The expression “computing an AD” means computing the value of this attribute for all rows of the target table. In our implementation, the tables are in data structures contained in memory; it would be even possible to store them in a database if the amount of data could not be accommodated in memory. Note also that the phases of generating and computing an AD are separated, but we describe them together for presentation purposes. Similarly, the terms AD and attribute are often interchanged. Nevertheless, the distinction between their generation and their computation is important and we will show that the generation is much faster than the computation.

2.3. Generation of Attribute Descriptors

Our propositionalization algorithm consists of generating and computing a set of ADs, and finally adding them to the target table, so that the number of attributes increases. The generation of ADs is performed in two phases. First, paths through the tables are generated; second, a set of ADs is generated given the current path.

2.3.1. Finding the paths

Starting from the Target table, the procedure navigates the TG following the existing associations, up to a certain depth, selecting in this way a sequence of elements. A path of
depth = 2 is, for instance, Target \rightarrow \text{Atom} \rightarrow \text{AtomBond}. Since each element can be encountered more than once, every element in the path is a copy of the original element. Also, for reasons that will be clear in the next section, an association can be navigated only if it does not generate a subpath \( E_1 \rightarrow E_2 \rightarrow E_1 \), where the first association is \( 0 - N \) and the second \( 0 - 1 \). Therefore, a path cannot contain either the subpath \( \text{Atom} \rightarrow \text{AtomBond} \rightarrow \text{Atom} \) or the subpath \( \text{Bond} \rightarrow \text{AtomBond} \rightarrow \text{Bond} \). Note that there is no theoretical limit in the depth of the path under consideration. The longer the path, the richer the information that can be expressed by an attribute, but also the longer the time required to compute each attribute.

### 2.3.2. Aggregations and Refinements

Given a path, we apply the Roll-Up algorithm, which consists of summarizing information and adding it to the target element. Starting from the element preceding the last one in the path and going back to the Target element, a new aggregate attribute is virtually added to each element, using information contained in the following ones. This procedure results in adding a new aggregate attribute to the target element, as shown in the pseudo-code in Figure 3.

\[ \text{Figure 3. Pseudo code of the Roll-Up procedure} \]

```plaintext
1. currentEle := the last element in the path
do {
   2. currentEle := the element preceding currentEle
   3. followingEle := the element following currentEle
   4. GenerateNextDerivedAttribute(currentEle, followingEle)
   5. add the derived attribute to currentEle
} while (currentEle != targetEle)
```

Suppose that we want to generate all possible attributes for the path Target \rightarrow \text{Atom} \rightarrow \text{AtomBond} \rightarrow \text{Bond}. In step 1 of Figure 3, currentEle is set to Bond. Then, in the first do-loop iteration, the element to which we add an attribute (currentEle) is AtomBond, at the second
iteration is Atom, and at the third is Target. The GenerateNextDerivedAttribute function (Figure 4) completes the definition of the entire procedure.

**Figure 4.** Pseudo code of the GenerateNextDerivedAttribute procedure

```
Input: currentEle p, followingEle f

If (the association from p to f is 0-1) {
    Choose an attribute from f (if f contains a derived attribute, select it);
    Attach it to p;
}
Else {
    Choose an attribute $A_g$ from f and a compatible aggregating function $agg$;
    Optionally choose a refinement as follows:
    Choose an attribute $A_r$ from f (if f contains a derived attribute, it must be selected either as $A_g$ or as $A_r$);
    If building a toValue refinement {
        Choose a compatible refinement operator $\rho$;
        Choose a value $v$ with the same dimension as $A_r$;
        Attach the following derived attribute to p:
        Select $agg(A_g)$
        From p
        Where $A_r \rho v$;
    }
    Else If building a comparison refinement {
        Choose a compatible refinement operator $\rho$;
        Choose an attribute $C$ with the same type and dimension as $A_r$ and belonging to p or to an element preceding p;
        Attach the following derived attribute to p:
        Select $agg(A_g)$
        From p
        Where $A_r \rho C$;
    }
}
```

The input to this function is two subsequent elements, currentEle and followingEle, and returns the derived attribute to add to currentEle.
The derived attribute has to summarize, for each row in currentEle, the content of followingEle. The derived attribute can be constructed in two ways, depending on the association connecting currentEle to followingEle:

4. by attaching an attribute of followingEle, if the association is $0 - 1$ (attachment)

4. by aggregating and possibly refining an attribute of followingEle, if the association is $0 - N$ (aggregate-and-refine)

We now show how attachment works using our example. In the first step of our example, currentEle is AtomBond and followingEle is Bond. Since the association is $0 - 1$, we choose an attribute of followingEle and attach it to currentEle. Suppose that the attribute type is chosen. Through a simple join on the composite key [idTarget, idBond] it is possible to retrieve, for each row in AtomBond, the value of type. Note that it would make no sense to choose either idTarget or idBond instead of type, because they are involved in the join and obviously they are already present in the currentEle. The attribute added to currentEle maintains the original dimension and type; then, AtomBond is modified as depicted in Figure 5.

**Figure 5.** Example of attachment
If followingEle contains a derived attribute $X$, then $X$ must be added to currentEle, otherwise an attribute that could be generated by a shorter path would be generated. Enforcing this condition guarantees that no attribute is generated more than once.

If the association is $0 \rightarrow N$, the derived attribute is generated through an aggregate-and-refine process. For each row $r$ in currentEle, there may be many corresponding rows (i.e., the $S(r)$ set) in followingEle. Hence, it is necessary to summarize the $S(r)$ set into one single value that will be the derived attribute for row $r$. Two definitions are necessary to accomplish this:

1. An aggregation of an attribute $A_g$ of followingEle
2. A refinement of an attribute $A_r$ of followingEle

To define an aggregation, we need to choose an attribute $A_g$ of followingEle and a suitable aggregating function. $A_g$ can be any attribute except any used for the join and the aggregating function must be chosen according to the type of $A_g$, as shown in Table 1.

<table>
<thead>
<tr>
<th>Aggregating Function</th>
<th>Input Attribute Type</th>
<th>Input Attribute Dimension</th>
<th>Output Attribute Type</th>
<th>Output Attribute Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>Numeric</td>
<td>D</td>
<td>Numeric</td>
<td>D</td>
</tr>
<tr>
<td>Max</td>
<td>Numeric</td>
<td>D</td>
<td>Numeric</td>
<td>D</td>
</tr>
<tr>
<td>Avg</td>
<td>Numeric</td>
<td>D</td>
<td>Numeric</td>
<td>D</td>
</tr>
<tr>
<td>Sum</td>
<td>Numeric</td>
<td>D</td>
<td>Numeric</td>
<td>D</td>
</tr>
<tr>
<td>MostFrequent</td>
<td>Categorical</td>
<td>D</td>
<td>Categorical</td>
<td>D</td>
</tr>
<tr>
<td>CountDistinct</td>
<td>Categorical or ID</td>
<td>D</td>
<td>Numeric</td>
<td>#D</td>
</tr>
</tbody>
</table>
Alternative aggregate functions may be implemented (e.g., median) which would generate different attributes, however, we chose the ones shown in Table 1 to be consistent with existing propositionalization approaches (Knobbe et al. 2001). Given the set of rows \( S(r) \) in followingEle that corresponds to row \( r \) in currentEle, the aggregating functions compute a single value, as follows:

- **Min** returns the minimum value of \( A_g \) in \( S(r) \)
- **Max** returns the maximum value of \( A_g \) in \( S(r) \)
- **Avg** returns the average value of \( A_g \) in \( S(r) \)
- **Sum** returns the summation of the \( A_g \) values in \( S(r) \)
- **MostFrequent** returns the most frequent \( A_g \) value in \( S(r) \)
- **CountDistinct** returns the number of distinct values in \( S(r) \)

Table 1 also reports the dimension of the output attribute. For example, consider the second step of the algorithm in Figure 3 relative to our example, for which currentEle is Atom and followingEle is the modified element AtomBond. The attributes that can be chosen as \( A_g \) are \( idBond \) and \( type \). Suppose that we choose \( idBond \), whose type is ID, then the countDistinct function must be used, because it is the only aggregating function that allows an input attribute of type ID. Therefore, the derived attribute for Atom is \( \text{countDistinct}(idBond) \), i.e. the number of bonds with which each atom participates. This new attribute is numeric and its dimension is “#bondID” (see Table 1). Note that the dimension of the new attribute may be compared to other existing attributes in order to create refinements. Nevertheless, in the molecule classification
application, this feature is not used because there are no attributes whose type is “#bondID” or “#atomID”.

A refinement is a condition that results in the selection of only a subset of rows in followingEle. When no refinements are present then the derived attribute is built using the entire $S(r)$ set. In this work, we consider two types of refinements: value refinements and comparison refinements. A value refinement has the form $A_r \rho v$, where:

- $A_r$ is an attribute belonging to followingEle
- $\rho$ is a compatible refinement operator
- $v$ is a value with the same dimension as $A_r$ and that is used for the comparison

Table 2 shows that refinement operators that may be used depending on the type of $A_r$.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Not considered</td>
<td>&lt;, &gt;</td>
</tr>
<tr>
<td>Categorical</td>
<td>=, !=</td>
<td>=, !=</td>
</tr>
<tr>
<td>ID</td>
<td>Meaningless</td>
<td>=, !=</td>
</tr>
</tbody>
</table>

Suppose that after selecting the aggregation $\text{countDistinct(idBond)}$ in the second step of the example, we build a value refinement where $A_r$ is type (i.e., the categorical attribute attached to AtomBond in the first step), $\rho$ is “=”, and $v$ is “2” (double bond), then the derived attribute may be represented in a SQL-like notation as $\text{countDistinct(idBond)}$ where $type = “2”$. If we denote as $C2$ this new aggregate attribute, which is added to the element Atom, then its value is the number of double bonds to which the atom participates. Here too, in order to guarantee that
no attribute is generated more than once, if followingEle contains a derived attribute \( X \), we must select \( X \) to be either \( A_g \) or \( A_r \).

Unlike MRDT approaches, we do not consider value refinements on numeric attributes. In the third step of our example, where currentEle is \( Target \) and followingEle is the modified Atom element, we could count the number of atoms (the aggregating function would be \( \text{countDistinct}(\text{idAtom}) \)) with \( C_2 < 2 \). In other words, we could count the number of atoms participating in less than 2 double bonds. This refinement on a numeric attribute is not allowed in our method; if a numeric attribute is aggregated, no value refinement is possible. Allowing this would lead to the generation of a large number of attributes that are similar to one another, with the only difference being the chosen numeric threshold (“2” in our case). Furthermore, similar information (although not identical) contained in these attributes is expressed by other attributes that are also generated (e.g. the average value of \( C_2 \) among all atoms). Note also that a value refinement is meaningless if \( A_r \) is an ID. For example, consider the following derived attribute: \( \text{most frequent (type) where idBond } \rho \text{ “126”} \). For any possible \( \rho \) (=, !=, > or <), the attribute would be clearly useless, because the id’s have no semantic meaning.

The first difference between our approach and existing propositionalization approaches is that existing propositionalization approaches use refinements only when the aggregate function is \( \text{countDistinct} \) and not when it is a numeric aggregate function (min, max, etc.). Consider again the third step of our example, where currentEle is Target and followingEle is the modified element Atom. Existing methods generate attributes such as “Maximum \( C_2 \) among the atoms” or "Number of atoms of Oxygen", but not attributes such as “Maximum \( C_2 \) of atoms of Oxygen”. The first example uses, for each row \( r \) in currentEle (i.e. for each molecule), all rows in \( S(r) \) and aggregates them with the \( \text{max} \) function, with no refinement. The second example has a
refinement (ele = “O”) and counts the rows satisfying this condition. The third example, which is not supported by existing approaches, has both a refinement condition (ele = “O”) and an aggregate function (max) different from count. Our method produces these refinements.

Our approach also considers comparison refinements of the form $A_r \rho C$, where:

- $A_r$ is an attribute belonging to followingEle
- $\rho$ is a compatible refinement operator
- $C$ is an attribute with the same type and dimension as $A_r$ and either belonging to an element preceding currentEle or to currentEle (join key excluded)

Unlike a value refinement, a comparison refinement compares $A_r$ to another attribute $B$ instead of to a fixed numerical value. In this refinement, $B$ may be chosen only among the attributes of the elements preceding followingEle. In fact, for every row in followingEle there is exactly one associated row in a previous element of the path. This is a direct consequence of limiting the navigation of the TG to $0-1$ and $0-N$ associations, and avoiding $N-N$ associations.

Neither existing MRDT nor propositionalization approaches consider comparison refinements, making it a novel feature of our approach. These refinements generate features that embed important information on sub-paths, such as “number of carbon atoms connected to at least one oxygen” or the presence of a particular ring, even when this may require great depths. Hence, the generation of features with comparison refinements differentiates our approach from traditional propositionalization methods. Additional insight on the implementation of aggregation and refinements may be gained by the detailed example included in the appendix.
Table 3 shows, for depths 1 to 7, the cumulative number of attributes generated by our approach (Exhaustive) and the traditional propositionalization approach, when applied to two datasets that represent two significantly different biological processes, both of which are highly relevant to drug discovery. Table 3 also shows the time (measured in milliseconds on an Intel® Xeon® CPU X5355 at 2.66 GHz equipped with 32 GB of RAM and Microsoft Windows Server 2003 R2 Enterprise x64 Edition) required to compute one attribute per compound at each depth. It is important to point out that the time taken to compute an attribute does not depend on whether the attribute belongs to the traditional or the exhaustive space.

<table>
<thead>
<tr>
<th>Depth</th>
<th>Estrogen Traditional</th>
<th>Exhustive</th>
<th>Mutagenesis Traditional</th>
<th>Exhaustive</th>
<th>Comp. Time (ms/compound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56</td>
<td>80</td>
<td>52</td>
<td>74</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>118</td>
<td>238</td>
<td>110</td>
<td>220</td>
<td>0.93</td>
</tr>
<tr>
<td>3</td>
<td>346</td>
<td>1,570</td>
<td>324</td>
<td>1,422</td>
<td>1.71</td>
</tr>
<tr>
<td>4</td>
<td>814</td>
<td>4,934</td>
<td>780</td>
<td>4,534</td>
<td>3.35</td>
</tr>
<tr>
<td>5</td>
<td>2,386</td>
<td>34,962</td>
<td>2,306</td>
<td>30,604</td>
<td>4.48</td>
</tr>
<tr>
<td>6</td>
<td>5,598</td>
<td>197,638</td>
<td>5,474</td>
<td>181,164</td>
<td>6.83</td>
</tr>
<tr>
<td>7</td>
<td>14,514</td>
<td>1,027,570</td>
<td>14,216</td>
<td>948,730</td>
<td>9.02</td>
</tr>
</tbody>
</table>

The estrogen receptor binding data set (Fang, Tong et al. 2001) consists of 232 compounds that have been tested for their ability to bind to the estrogen receptor, expressed as the binding affinity of the compound relative to the natural ligand for the estrogen receptor, 17-β estradiol. This endpoint represents a single biochemical event – small molecule interaction with a protein target – which is ubiquitous in early drug discovery. Early evaluation of binding affinity of compounds that are available or could be synthesized assists in establishing priorities for purchasing or synthesis, thus reducing the resources necessary to identify novel and useful chemical matter.
The mutagenesis dataset (Votano, Parham et al. 2004) consists of 400 compounds that have been evaluated for Ames mutagenicity, a measure of the mutagenic (and thus carcinogenic) potential. While the mutagenicity assay is only a surrogate for the true carcinogenic potential, it is recognized as the de facto standard for evaluating the carcinogenicity of chemical compounds at relative early stages of drug discovery. Relative to interaction with a protein target, Ames mutagenicity is a more complex phenomenon and there exist multiple mechanisms by which a compound could exert a mutagenic effect. The result is that any modeling procedure must be sophisticated enough to isolate multiple mechanisms of action and the compounds that are active/non-active for each mechanism. Again, early identification of potentially problematic chemical matter reduces the resources necessary to develop new drug compounds. Both datasets are publically available and have been evaluated by various authors in the past, although, to the best of our knowledge, they have never been used to test any propositionalization or relation learning technique.

2.4. Attribute Bound Experiments

The times reported in Table 3 show the significant increase experienced with the depth. The reason is that the number of joins needed to retrieve the value of the attributes linearly increases with the depth. This indicates that only a subset of attributes at depths greater than 4 can be computed in most practical settings. In order to assess the classification accuracy of our proposed approach, we used a 10-fold cross validation (CV) performed by the following 10 Weka classifiers (all set at their default parameters): BayesNet, PART, RandomForest, Bagging, MultilayerPerceptron (neural network), J48 (C4.5), ADTree (alternating decision tree), REPTree (fast decision tree), NNge (nearest-neighbor-like algorithm), Ridor (RIpple-DOWN Rule learner). Bagging uses a Fast decision tree as its elementary classifier, which is also Weka’s default
classifier Weka is an open source data mining framework available for download at
http://www.cs.waikato.ac.nz/ml/weka/ (last accessed on 05/18/2010). The software may be
downloaded to access literature references to the classifiers. Witten and Frank (2005) provide
additional details.

We partitioned the data sets into 10 pairs $P_i = (\text{training set } i, \text{test set } i), i = 1, \ldots, 10$. The
classifiers listed above implement well-known classification techniques, such as neural networks,
meta-classifiers, decision trees and rules. They also have the desirable property of accepting
input data with missing values, which is of particular importance to us because many of the
attributes generated are not defined for all molecules. For example, “the most frequent element
different from carbon” is not defined for molecules containing only atoms of carbons, however,
the database contains such molecules because hydrogen atoms are not included in the molecule
representation even if they are actually present in the molecule. Alternatively, it would be
possible to fill the missing values with the average value or the mode, but since many attributes
at low depths are missing for most molecules, this strategy would fail to capture key patterns at
those depths. Since our experiments often involve thousands of attributes, in all experiments of
the paper a supervised feature selection is performed before the execution of the test
corresponding to each fold. Considering the current training set, a set of attributes is selected
using the default feature selection algorithm in Weka (Evaluator: CfsSubsetEval and Search:
BestFirst). Then, the selected attributes are fixed and the classifiers are applied to the current $P_i$.
We have observed that the feature selection improves the average classification accuracy,
particularly when the number of attributes exceeds 1,000. The proportion of selected attributes
varies with the number of initial attributes, but never exceeds 50, even in experiments where the
initial number of attributes is about 8,000. In our experiments, we measure overall classification
accuracy with the average cross validation accuracy obtained by the 10 classifiers. This choice is consistent with the goal of this work, which is to generate attributes that are valuable in a classification process, and not to design new classification or feature selection procedures.

Existing propositionalization approaches compute all attributes up to a predefined depth. Let us refer to this strategy as up-to-depth-$X$ (UD-$X$), where $X$ is the predefined maximum depth. Figure 6 shows the average cross validation accuracy obtained by the 10 classifiers for strategies UD-1, UD-2, UD-3 and UD-4.

Figure 6 suggests that, for both data sets and for both attribute spaces (traditional and exhaustive), the accuracy increases if attributes constructed at deeper levels are included. Also, at depths 3 and 4, the attributes generated by the exhaustive approach yield a higher accuracy than the ones generated by the traditional methods. Interestingly, this is not true for the Estrogen data set at depths 1 and 2. We believe that, at these depths, the information that can be expressed by any exhaustive attribute is also contained in some “simpler” traditional attribute. For
example, consider the exhaustive attribute “average number of bonds connecting an atom of Carbon” and the traditional attribute “number of atoms of Carbon”. Both estimate the molecular mass, but, while the former does it in an involved way, the latter does it in a simple way. Thus, training a classifier on the exhaustive space leads to overfitting and, therefore, the accuracy obtained will be lower. The situation is reversed at lower depths, which contain information that can be expressed by the exhaustive attributes but not by the traditional ones. Table 4 reports the best average cross validation accuracy and the best 2 individual cross validation performances obtained in these UD tests, in terms of accuracy. We report also the second best because its accuracy may have a lower standard deviation ($\sigma$), a desirable property. We use the notation UD-X-Y, where X is the maximum depth and Y is E for the exhaustive attribute space and T for the traditional attribute space. Table 4 shows that, with only one exception, all best performances were obtained with UD-4-E. This indicates the value of using Exhaustive and exploring greater depths.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Average CV accuracy</th>
<th>Two best individual CV accuracies</th>
<th>First</th>
<th>Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estrogen</td>
<td>76.10% — UD-4-E</td>
<td>79.24% ($\sigma = 9.23%$)</td>
<td>78.82% ($\sigma = 8.90%$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UD-4-E</td>
<td>UD-4-E</td>
<td></td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>75.20% — UD-4-E</td>
<td>79.75% ($\sigma = 6.61%$)</td>
<td>79.50% ($\sigma = 10.26%$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UD-3-E</td>
<td>UD-4-E</td>
<td></td>
</tr>
</tbody>
</table>

In order to assess the benefits of including attributes found at deeper levels, we must define a strategy that chooses which ones to compute. The problem of selecting the attributes to compute among a vast set of attributes, such as the ones between depths 4 and 7, is a complex search problem. Generating the ADs representing these attributes is relatively fast, requiring
approximately 3 minutes for the approximately 1 million ADs in the Exhaustive attribute space. However, computing just one AD for all compounds can consume seconds.

We ran an experiment where the set of computed attributes includes all up to depth 3 and 1,000 randomly chosen between depths 4 and 7. We chose to compute all attributes up to depth 3 (instead of 4) because, as shown in Figure 6, the accuracy obtained with UD-3 and UD-4 is similar, but computing only the attributes up to depth 3 is much faster. We denote this randomized strategy with RAND and Table 5 reports the average accuracy obtained and the standard deviation (between parentheses) across all classifiers when compared to UD-3. Table 6 reports the best results using RAND in similar format as Table 4.

<table>
<thead>
<tr>
<th>Table 5. Comparison between UD-3 and RAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set</td>
</tr>
<tr>
<td>Estrogen</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Mutagenesis</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6. Average and best individual CV accuracy on RAND tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Estrogen</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Mutagenesis</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The randomized strategy yields a higher accuracy for both data sets and both attribute spaces. Interestingly, the gap between RAND and UD-3 is larger than the gap between UD-4 and UD-3, even though the number of attributes at depth 4 is more than 1,000. This reveals the usefulness
of the information embedded in attributes derived at depths 5, 6, and 7. The experiment, therefore, suggests the value of developing a mechanism for including attributes at lower depths. When dealing with a computational budget, there is a tradeoff between generating all attributes at higher depths and sampling attributes from lower depths. In our next experiment, we explore this tradeoff.

2.5. Generating Attributes when Limiting the Computational Time

In practical settings, we must consider that there is a time limitation imposed on the generation of attributes from data sets. Since the experiments showed that the complete generation of attributes at lower depths than 3 is extremely time consuming, we now investigate several mechanisms to sample attributes at depths 4 and beyond. In particular, we compare two different strategies:

1. *Scan* — compute all attributes from depth 1 on until reaching the time limit
2. *Scan and Sample* — compute all attributes from depth 1 on until half of the allotted time is reached, then randomly sample from the un-computed attributes at the current depth and any lower depth until the computational budget is exhausted.

We executed both of these strategies in combination with the traditional and exhaustive approaches and time limits of 5, 30, 60, and 120 minutes. All experiments were conducted on an Intel® Xeon® CPU X5355 at 2.66 GHz equipped with 32 GB of RAM and Microsoft Windows Server 2003 R2 Enterprise x64 Edition. Tables 7 and 8 report the average accuracy and the standard deviation (between parentheses) across the classifiers obtained for each data set, time limit, strategy and approach.
Table 7. Results on Estrogen

<table>
<thead>
<tr>
<th>Time Limit (min)</th>
<th>Traditional</th>
<th>Scan</th>
<th>Exhaustive</th>
<th>Traditional</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Scan and Sample</td>
<td>Traditional</td>
<td>Exhaustive</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>72.62% (2.00%)</td>
<td>74.48% (2.65%)</td>
<td>74.12% (2.60%)</td>
<td>71.22% (2.38%)</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>75.07% (2.52%)</td>
<td>76.10% (2.65%)</td>
<td>75.30% (2.87%)</td>
<td>77.06% (2.53%)</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>76.88% (2.12%)</td>
<td>76.97% (2.12%)</td>
<td>77.00% (2.39%)</td>
<td>77.15% (2.00%)</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>77.09% (2.96%)</td>
<td>76.76% (1.52%)</td>
<td>78.03% (1.93%)</td>
<td>78.18% (2.71%)</td>
<td></td>
</tr>
</tbody>
</table>

Table 8. Results on Mutagenesis

<table>
<thead>
<tr>
<th>Time Limit (min)</th>
<th>Traditional</th>
<th>Scan</th>
<th>Exhaustive</th>
<th>Traditional</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Scan and Sample</td>
<td>Traditional</td>
<td>Exhaustive</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>74.25% (2.87%)</td>
<td>72.53% (2.85%)</td>
<td>70.90% (2.46%)</td>
<td>72.20% (3.32%)</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>73.23% (2.21%)</td>
<td>74.60% (4.16%)</td>
<td>73.73% (2.95%)</td>
<td>75.63% (2.73%)</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>73.93% (2.59%)</td>
<td>77.23% (2.65%)</td>
<td>74.45% (3.33%)</td>
<td>77.20% (3.15%)</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>74.40% (3.03%)</td>
<td>77.48% (2.50%)</td>
<td>74.60% (3.41%)</td>
<td>77.90% (2.84%)</td>
<td></td>
</tr>
</tbody>
</table>

Table 9 reports the CV accuracies in the format used in Tables 4 and 6. The notation X-Y-Z is used to indicate the attribute generation strategy (S = scan and SS = scan and sample), the time limit (5, 30, 60 and 120) and the attribute space (T = traditional and E = exhaustive).

Table 9. Average and best individual CV accuracy on time-limit tests

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Average CV accuracy</th>
<th>Two best individual CV accuracies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>First</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Second</td>
</tr>
<tr>
<td>Estrogen</td>
<td>78.18% — SS-120-E</td>
<td>81.83% (σ = 10.65%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS-120-E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81.39% (σ = 8.57%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS-120-E</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>77.90% — SS-120-E</td>
<td>82.75% (σ = 7.95%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS-60-E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>82.50% (σ = 5.77%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S-30-E</td>
</tr>
</tbody>
</table>

38
The results in Tables 7 and 8 show that the Scan strategy should be preferred for relatively short computational times. In the runs with a 5-minute limit, Scan generally outperforms the Scan and Sample strategy. This seems to be another instance of the Occam’s razor effect that suggests that simple classification rules — high level attributes in this case — are preferable to more complex ones (Domingos 1999). The effect does not hold after the procedure is able to generate more attributes at lower levels (as is the case when the time limit is extended). The results show two different patterns, which are more clearly identified when the results in Tables 7 and 8 are shown graphically (see Figure 7). The patterns are such that Scan and Sample tends to be the better strategy, particularly when applied to Estrogen, and Exhaustive tends to produce a better attribute space than Traditional, particularly when applied to Mutagenesis. Table 9 shows that the best results are generally obtained by the Scan and Sample strategy, using the exhaustive attribute space, and with a large time limit.

**Figure 7.** Graphical representations of the results in Tables 7 and 8
Figure 7 shows an advantage of Exhaustive over Traditional on the Mutagenesis data set, while the different methodologies seem to achieve similar performances on the Estrogen data set — though the statistical analysis below shows that our enhancements lead in fact to a higher accuracy.

This result may be explained by the complexity of the mutagenesis process. Compounds tested for mutagenicity must first diffuse into bacterial cells, a non-complex process itself related to molecular size, charge, hydrophobic/hydrophilic balance, and molecular flexibility. Once inside the cell, compounds can then interact with cellular components in a number of ways leading to DNA damage. Direct interaction with DNA, DNA modifying enzymes, DNA replicating enzymes or the replication process itself, DNA repair enzymes, and DNA packaging proteins may all lead to mutations in the bacterial genome and a positive test. Unlike estrogen receptor binding which occurs within a single, defined location of a single protein, mutation may occur through interactions with various targets (DNA and protein, at least) and those targets may provide multiple points of interaction. It is thus not surprising that a more complex set of rules are required to accurately classify a compound as mutagenic positive or negative.

The data generated in this experiment are amenable to statistical analysis. In particular, we have recorded the classification accuracy obtained by the 10 classifiers for each combination of the following variables:

- Data set (Estrogen and Mutagenesis)
- Time (0.5, 1 and 2 hours)
- Attribute space (traditional and exhaustive)
- Scanning strategy (Scan and Scan and Sample)
In our first experiment, we contrast the merit of the approaches to generate the attribute space (i.e., traditional and exhaustive). For each combination of data set, time, and scanning strategy, we record the proportion of classifiers that obtain a higher CV accuracy if the exhaustive attribute space is used. A $t$-test showed that the proportion of classifiers performing better when employing the Exhaustive over the Traditional approach is greater than 0.5 with a $p$-value of 0.0014. In our second experiment, we compare the merit of the two scanning strategies (i.e., Scan and Scan and Sample). Similarly to the first experiment, for each combination of data set, time, and attribute space, we record the proportion of classifiers that obtain a higher CV accuracy if the SS strategy is used. A $t$-test showed that the proportion of classifiers performing better with the SS strategy when compared to the Scan strategy was greater than 0.5 with a $p$-value of 0.0041. Both of these tests show that for runs longer than 5 minutes classifiers tend to perform better when using Exhaustive or Scan and Sample. However, these tests do not provide evidence indicating which classification techniques benefit the most from either approach.

In order to provide a better assessment of the quality of the attributes generated by our method, we consider the accuracy obtained when using Daylight and MACCS Keys, two of the most popular fingerprint representations currently used in drug discovery. For both data sets, the fingerprint representations were generated using the cheminformatics toolkit, RDKit, which is freely available at [http://www.RDKit.org](http://www.RDKit.org) (accessed on 05/18/2010). We performed a cross validation test on the 2 data sets using the same classifiers and the same feature selection technique. Table 10 reports the average CV accuracy and the standard deviation (between parentheses) across the classifiers obtained using Daylight, MACCS Keys, and the Scan-and-
Sample-Exhaustive approach (reported in Tables 7 and 8) when imposing a time limit of 120 minutes.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Daylight</th>
<th>MACCSS Keys</th>
<th>Scan and Sample Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estrogen</td>
<td>79.69% (2.68%)</td>
<td>75.69% (10.50%)</td>
<td>78.18% (2.71%)</td>
</tr>
<tr>
<td>Mutagenesis</td>
<td>76.33% (1.37%)</td>
<td>78.60% (2.29%)</td>
<td>77.90% (2.84%)</td>
</tr>
</tbody>
</table>

We have compared the performance obtained by each Fingerprint representation to the one obtained by the Scan-and-Sample-Exhaustive approach employing a two-proportion test. For both data sets we recorded the proportion of classifiers that perform better with the Fingerprint representation and then tested if the proportion was statistically greater than 0.5. For the Estrogen data set, 5 classifiers out of 10 perform better using the Daylight representation ($p$-value 0.5), whereas 7 out of 10 perform better using the MACCS Keys representation ($p$-value 0.103); for the Mutagenesis data set the proportion was 7 out of 10 for both fingerprint representations ($p$-value 0.103 for both). In all cases, the proportion was not significant, indicating that both fingerprint representations achieve similar performances to our best attribute representation.

Over the years, the molecule classification problem has been tackled by several techniques, all of which require experts’ knowledge to define the attribute representation. Our propositionalization method, however, allows new features to emerge. Therefore, when classifiers are applied to the Fingerprints data, the classification rules employ attributes that have been chosen a priori by domain experts, whereas our propositionalization-based attribute representation may lead to classification rules involving molecule characteristics that experts
have not taken into account beforehand and in this way can be viewed as complementing experts’ knowledge. While it is expected that some attributes will occur which are consistent with pre-existing knowledge of the modeled domain, it is also anticipated that new knowledge may emerge due to the in-process, unbiased methodology. To determine if new information was discovered, we considered the attributes generated by the experiment “Scan and Sample Exhaustive” with a 2 hour time limit. Attributes appearing in each of the 10-cross validation folds for each dataset were evaluated for their biochemical meaning. Following is a list of some of these attributes and an interpretation of their presumed meanings with respect to the modeled endpoint.

**Mutagenicity**

- **Number of bromine atoms.** As mentioned previously, mutagenesis is a complex biological phenomenon which may require more complex rules to fully model the endpoint. Interestingly, the simple count of bromine atoms occurred in each cross-validation fold. Within the dataset used, we found 12 compounds that contain bromine (many with multiple bromine atoms), 11 of which are mutagenic, and thus the number of bromine atoms represents a simple mutagenicity filter based on these data. This finding is consistent with experimental findings that bromine-containing chemical structures are more mutagenic than analogues within which chlorine is substituted for bromine (Finkelstein 1994).

- **Label each atom with the number of double bonds it has. Compute the mean of these labels across the atoms different from sulfur.** By distinguishing compounds that have large numbers of double bonds, this attribute effectively identifies structures that contain a high degree of aromatic ring systems. Compounds in which those rings are fused (in
which some atoms participate in multiple rings rather than two rings joined by a single bond — Figure 7A) further contribute to an increase in the value of this attribute. These types of fused ring systems are inherently planar and are commonly known to be problematic due to their ability to intercalate DNA leading to mutations (Ferguson 2007). The clause “compute the mean of these labels across the atoms different from sulfur” is interesting as it effectively eliminates double bonds to sulfur atoms from impacting the calculation. These substructures are not planar but rather introduce a three-dimensional quality to the molecules — a characteristic that we speculate may prevent DNA intercalation. The electronic nature of such substructures may also prevent direct interaction with DNA via repulsion form the negatively charged phosphate backbone of DNA.

- Label each atom different from carbon with the number of atoms of nitrogen to which it is connected. Compute the mean of these labels. While we are not aware of specific evidence that [non-carbon – nitrogen] atom pairs are generally mutagenic, 49 nitro-containing structures (Figure 7B) are present in the dataset, 46 of which are mutagenic. A further evaluation of the dataset reveals 117 compounds exhibiting the [non-carbon – nitrogen] atom pair, only 19 of which are non-mutagenic, suggesting a relatively robust filter and that atom pairs of this sort would be best avoided in the design of new drug compounds. The example in the appendix shows how this attribute is generated.

**Estrogen**

- Label each bond with the number of oxygen involved in it. Compute the mean of these labels across all bonds of type different from 3. This attribute most likely acts as an oxygen indicator. The clause “Compute the mean of these labels across all bonds of type
different from 3” prevents triple bonds in the molecule from artificially depressing the index as oxygen is not found participating in triple bonds. Generally, binding to the estrogen receptor is influenced by specifically placed oxygen-containing groups placed at either end of the molecule (Brzozowski, Pike et al. 1997), and a lack of these groups tends to decrease the probability of binding to the receptor. Panels C, D, and E of Figure 8 illustrate the importance of oxygen and the impact of oxygen placement on estrogen receptor binding. While this type of indicator appears useful, it is clear that it must be combined with other attributes in order to establish the necessary and sufficient basis for making a prediction.

• Label each atom A in the following way. 1) Consider the atoms connected to it and count the bonds to which they participate (excluding the bond connecting A to each of them). 2) Compute the sum of these labels and obtain the label for A. Label the molecule with the minimum of these labels across all atoms of oxygen. The previous attribute acted as a relatively non-specific oxygen indicator, but this new attribute behaves as an indicator of oxygen placement within the molecular scaffold. Specifically, a high value would represent an oxygen atom that is connected to other atoms participating in a large number of additional bonds - presumably an oxygen atom that is somewhat buried and interacting with highly branched atoms. Low values would represent low branching, and moreover, terminal oxygen atoms would have particularly small values. As illustrated in Panels C, D, and E of Figure 8, all compounds contain oxygen atoms, however, only a specifically placed terminal oxygen atom can impart estrogen binding activity.
In section 2.5, we introduced the Scan and Sample approach for selecting attributes at increasingly lower depths. In addition to this sampling procedure, we attempted two other strategies for selecting attributes. The first strategy, attempted to identify the best mix of attributes from each depth. For this purpose, we set up a search using OptQuest, a commercially available general-purpose optimizer, that is based on the scatter search metaheuristic (Glover, et al. 2003). The optimization model consisted of 4 continuous variables and one constraint. The variables represented the proportion of attributes from levels 4 to 7. The constraint forced the sum of the variables to be equal to one. For each set of proportions suggested during the OptQuest search, a sample procedure was used to sample attributes from each depth, as dictated by the given proportions, and configure a data set. The data set was then used to perform cross
validation with the set of classifiers in order to estimate an average accuracy for the proposed proportions.

The second strategy selected attributes one at a time by considering diversity and expected information gain. The diversity of an AD is the Euclidean distance from its vector representation and the vector representation of other attributes already in the set. The vector representation of an AD contains information such as its depth, the number of refinements and the number of times that aggregating functions appear. The expected information gain is given by a Bayesian Network, which, initially, does not have a way of distinguishing between ADs with high information gain and those with low. However, the Bayesian Network learns, through incremental training, and is then able to discriminate between ADs with potential high information gain from those whose potential is low. Our experiments showed that the estimates given by the Bayesian Network became gradually more accurate.

Unfortunately, neither of these two approaches outperformed the random sampling used within the scan and sample strategy. Our conjecture is that attributes at depth 7 contain discriminant information that is not captured by attributes at higher depths. If this is correct, the sampling strategy has a substantial advantage because most of the attributes that it chooses are from depth 7, due to the large number of attributes generated by both the Exhaustive and Traditional approaches at this depth (see Table 3). Unlike the random sampling, the strategic approaches described above choose a mix of attributes that are not biased toward the lowest depth.

2.7. Conclusions

We improve the current propositionalization procedures that are typically used in the process of drug discovery by creating a method that generates more expressive and complex (deeper)
attributes. Our experimental testing shows that the multi-relational data mining (MRDM) methodology that we employed is competitive with current methods within the field of cheminformatics and quantitative structure activity relationship modeling. MRDM has the advantage that no precalculated molecular representations were required but rather the technique uses the molecular structure itself and derives the necessary descriptors from it during the modeling process. Both datasets utilized are very well characterized and represent two biochemical processes that differ dramatically in their complexity, and in both cases MRDM is competitive with published results and those we have obtained previously using more traditional techniques (not shown). To our knowledge, this represents the first application of the MRDM technology within the field of drug discovery highlighting their potential and opening new avenues of research within this hybrid field. Techniques such as these provide a mechanism to prioritize chemical compounds for purchase or synthesis and increase the probability of bringing a successful drug to market within a shorter time and at reduced costs. Given that current estimates of the cost to bring a single drug to market in the range of $500 million to $2,000 million (DiMasi, Hansen et al. 2003), efforts to reduce costs are necessary to ensure efficient improvements in healthcare.

Future research opportunities based on our work include improving the accuracy and interpretability of the attributes that the procedure identifies. We have identified that embedding a procedure for selecting attributes (that is not based on sampling) is a challenging problem. An improved representation of the ADs may result in more interpretable attributes. We currently represent them by a SQL query, but a graphical representation — where an attribute is represented as the presence or absence of a particular characteristic — may provide additional
insights. Finally, we believe that our propositionalization approach can be applied to other classification problems in fields such as marketing and finance.
In Samorani and LaGanga (2011b), we generalize the traditional scheduling problem for outpatient appointments by considering 1) the possibility of implementing overbooking, 2) individual show probabilities that also depend on the appointment day 3) patient categories that correspond to different waiting times, 4) completely customizable revenues earned for scheduling patients, 5) the possibility of choosing the appointment day within a certain scheduling horizon.

3.1. Introduction

Efficient scheduling of clinic appointments leads to better resource allocation, timely access to healthcare, and to lower healthcare costs (White et al. 2011). Several decades of appointment scheduling research have focused on access to outpatient healthcare services (Gupta and Denton 2008). Cayirli and Veral (2003) show that the failure of patients to show up for appointments has large effects on the performance of scheduling systems. Thus, there has been increased interest in ensuring that all allocated appointment slots are used by managing the problem of patient no-shows (Murthuraman and Lawley 2008).

A popular technique for tackling the no-show problem is “overbooking”, which consists of assigning more than one patient to the same appointment slot (Robinson and Chen, 2010), or reducing the time between two consecutive appointments (LaGanga and Lawrence, 2007b). By overbooking, clinics try to balance the expanded “patient access” obtained (i.e., the extra patients seen) with the patients’ waiting time and clinic overtime which may be introduced. While much of the literature on overbooking assumes that all patients have undifferentiated show probabilities, recent work has considered the individual show probabilities of the patients.
Muthuraman and Lawley (2008), Samorani and LaGanga (2011), and Zeng et al. (2009) propose stochastic overbooking models to dynamically schedule patients in the most appropriate slot, given their show probability; Glowacka et al. (2009) use data mining to estimate the individual show probabilities of the patients, and then test the performance of different scheduling rules. These systems first detect the patients who are likely to show and those who are likely not to show, and then schedule them in the most opportune slots.

Although these works show that the quality of the schedule can be improved by considering individual show probabilities, they present a major shortcoming: they do not consider the problem of choosing the best day to schedule a patient’s appointment. Although other authors have considered dynamic, multi-period scheduling (Gupta and Wang, 2008), we are the first who use this technique in conjunction with individual show predictions. Liu et al. (2010) reported that show probabilities decrease with the “lead time” to the appointment, i.e., the number of days between the arrival of the appointment request and the appointment day. In other words, patients are less likely to show if scheduled far in advance than if scheduled in the near future. This issue has been addressed by using an open access (OA) policy, under which appointments are scheduled only for the same or the next day (Robinson and Chen 2010), in an effort to minimize no shows. Although an open access policy offers attractive benefits for patients who need to be seen very soon after their appointment request, it does not accommodate those who prefer to plan and schedule routine follow-up appointments farther in advance, operationally, open access can be difficult to implement successfully because of the uncertainty and variability of patients calling on any day for same-day service. Thus, it is necessary to balance the patient benefits of immediate access with the provider cost of idle time when the allocation of same-day appointment slots exceeds same day demand.
If individual show probabilities are considered, a scheduling system can exploit their dependence on the appointment lead time, in order to achieve a higher quality schedule. We show that this can be simply achieved by scheduling likely-to-show patients at the last moment (e.g., with a same-day appointment) and unlikely-to-show patients far in advance (e.g., months in the future). In order to validate this simple policy, we develop a general optimization algorithm to create dynamic appointment schedules in outpatient healthcare clinics and study the schedules obtained.

As patients call in over time to make an appointment, the algorithm schedules them to an open slot of the next $h$ days (“scheduling horizon”) or rejects them, according to their predicted individual show outcome. The predicted show outcome, which depends on the patient and on the day, indicates whether the patient is predicted to show or not if scheduled on a certain day. These predictions are obtained by using data mining to find a “classification rule” that effectively discriminates between showing and non-showing appointments. Classification techniques are based on statistical tools, such as regression and Bayesian Networks, or on other computer science algorithms, such as decision trees and support vector machines (Witten and Frank, 2005). The scheduling decision is made by solving a stochastic optimization problem through column generation. We use overbooking by slot compression rather than double-booking, which arguably offers a fairer balance of patient wait time and provider utilization (LaGanga and Lawrence, 2007a,b). Beside day-dependent show probabilities, we include individual waiting time costs, which can be used to allow different patient priorities. By assigning priorities, we try to minimize the waiting time of certain subsets of patients, such as, for instance, very sick people, children, or those who complained in the past about excessive waiting times. Similar
priorities have been considered by Patrick et al. (2008) and Zeng et al. (2010), but they have been neglected in the literature on individual show probabilities.

We test our algorithm through simulation on a dataset of a large outpatient community mental health center (the Mental Health Center of Denver (MHCD)). Throughout our experiments, novel findings are identified that can be of immediate use for clinics. First, we show that if individual show probabilities are used, adopting same-day scheduling is a worse choice than using a longer scheduling horizon. Second, we show how the rule’s ability to correctly classify patients as “shows” and “no-shows” impacts the performance in terms of patient access, overtime, and waiting time. Finally, we use a simulation-optimization approach to find the best scheduling set up for this clinic and test the simple policy reported above.

Our exposition proceeds as follows. In Section 3.2, we define the dynamic appointment scheduling problem. In Sections 3.3 and 3.4, we develop a new appointment request scheduling algorithm and outline a solution procedure. The results of computational experiments are summarized in Section 3.5, including experiments using data taken from MHCD and experiments that extend our results to stochastic service times. In Section 3.6 we present and test a simple scheduling policy. We conclude with managerial insights that emerged through our experiments and define future research directions in Section 3.7.

3.2. Clinic Model and Problem Definition

We model the clinic as a single server, single stage system, which is suitable also for those clinics with multiple providers, each serving a different set of patients. Following the overbooking method of slot compression (LaGanga and Lawrence 2007b), an appointment slot is assigned to at most one patient, who may or may not show up to his or her appointment. We assume constant service times (this assumption is relaxed in section 3.5.3) and patient
punctuality. The clinic session is composed of $K$ slots of length $T$, and the appointment length is $D$ ($D \geq T$). Let $N$ be the capacity of the clinic, i.e., the maximum number of patients that can be serviced in one day without overbooking. By overbooking, the number of appointments increases. For example, if the clinic session length is 6 hours (360 minutes) and $D = 40$ minutes, then the clinic capacity is $N = 360 / 40 = 9$ patients. By setting $K = 12$, it is possible to see up to 12 patients/day. In this case, appointments are scheduled every $T = 30$ minutes, because $360$ minutes / $12$ slots = $30$ minutes/slot. While overbooking increases the number of patients serviced, it introduces clinic overtime and patient waiting time. The overtime cost includes the wages for the staff and the extra use of electricity and other resources, while the waiting time cost of a patient represents monetary or nonmonetary and societal costs, such as the patient’s lost earnings or labor productivity while waiting, or a loss of goodwill and reduced patient satisfaction that the clinic incurs for making a patient wait.

Let us focus on the moment when an appointment request arrives and needs to be scheduled, and the current day is set to $d=0$. The appointment request can be scheduled in any empty slot of any day $d \in \{0, 1, \ldots, h - 1\}$, for the scheduling horizon of $h$ days, or, alternatively, it can be rejected. Therefore, an appointment request is characterized by a vector $s^r$ whose $i$-th component ($i = 0, \ldots, h - 1$) is a binary value indicating if the patient will show (1) or not (0) at an appointment scheduled in day $i$. The superscript $r$ indicates that each component of the vector represents the real outcome (show or no-show), which, obviously, is known only after the appointment. When the appointment request arrives, we can predict this outcome through data mining. In particular, we assume, using data mining terminology, that a “Classifier” has been “trained” with the clinic data set and “has learned” how to predict if an appointment request will result in a show or in a no-show in each of the days of the scheduling horizon. Thus, an
appointment request is also characterized by a vector \( s^p \) whose \( i \)-th component \( (i = 0, \ldots, h-1) \) is a binary value indicating whether the patient is predicted to show (1) or not (0) at an appointment scheduled after \( i \) days.

The clinic accrues a unit of revenue or other benefit for each showing patient. Here, we generalize this concept by characterizing each appointment request with a vector \( r^r \) whose \( i \)-th component \( (i = 0, \ldots, h-1) \) is the revenue obtained by scheduling the patient in day \( i \). For example, the component \( r^r(i) \) can be defined in terms of \( s^r(i) \) by setting it to some positive quantity if \( s^r(i) = 1 \), or in any other arbitrary fashion. This choice allows the user to implement custom policies. In this paper, the revenue is defined as follows:

\[
r^r(i) = \tilde{\pi} - i\delta + \pi \cdot s^r(i)
\]

\( r^r(i) \) depends on two factors: the lead time and whether the patient shows or not. The former factor, expressed by \( \tilde{\pi} - i\delta \), is implemented as a penalty \( i\delta \) proportional to the lead time, which is incurred for both showing and non-showing patients, because we want to minimize the waiting time for obtaining an appointment of any appointment request. The term \( \tilde{\pi} \) is a positive revenue earned when any appointment request is scheduled (i.e., not rejected). The latter factor, expressed by \( \pi \cdot s^r(i) \), represents the revenue \( \pi \) earned in the case of a showing appointment request, where \( \pi \gg \tilde{\pi} \). When an appointment request arrives, the vector of predicted revenues \( r^p \) is built according to \( s^p \):

\[
r^p(i) = \tilde{\pi} - i\delta + \pi \cdot s^p(i)
\]

As mentioned above, two costs are incurred once overbooking is adopted: waiting time cost and overtime cost. Unlike previous works, where the waiting time cost is equal for all patients, we assume that each patient belongs to a waiting cost category, which is characterized
by a waiting time cost $\omega_c$ per unit of waiting time. Finally, an overtime cost $\tau$ is paid for each unit of overtime. The notation is reported in Table 11.

**Table 11: Notation**

<table>
<thead>
<tr>
<th>$D$</th>
<th>Duration of an appointment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Interval between scheduled appointments, $T \leq D$</td>
</tr>
<tr>
<td>$N$</td>
<td>Clinic capacity (i.e. the number of slots obtained by setting $D = T$)</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of appointments scheduled (i.e. number of slots), $K \geq N$</td>
</tr>
<tr>
<td>$h$</td>
<td>Scheduling horizon, in days</td>
</tr>
<tr>
<td>$s^r(v, i)$</td>
<td>1 if appointment request $v$ shows on day $i$, 0 otherwise</td>
</tr>
<tr>
<td>$s^p(v, i)$</td>
<td>1 if appointment request $v$ is predicted to show on day $i$, 0 otherwise</td>
</tr>
<tr>
<td>$r^r(v, i)$</td>
<td>Actual revenue earned if appointment request $v$ is scheduled in day $i$</td>
</tr>
<tr>
<td>$r^p(v, i)$</td>
<td>Predicted revenue earned if appointment request $v$ is scheduled in day $i$</td>
</tr>
<tr>
<td>$\bar{\pi}$</td>
<td>Benefit of scheduling any appointment request</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Penalty for each day of delay in scheduling any appointment request</td>
</tr>
<tr>
<td>$\pi$</td>
<td>Additional benefit of scheduling a showing appointment request</td>
</tr>
<tr>
<td>$\omega_c$</td>
<td>Cost for each unit of waiting time of a patient of category $c$</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Cost for each unit of overtime</td>
</tr>
</tbody>
</table>

Extending the model proposed by LaGanga and Lawrence (2007b), the objective of the scheduling problem is to maximize the profit of the clinic of the next $h$ days:

$$P = \sum_{d=0}^{h-1} P(i) = \sum_{d=0}^{h-1} [R_d - O_d - W_d]$$

$R_d$ is the total revenue made in day $d$, $O_d$ is the overtime cost experienced by the clinic in day $d$, and $W_d$ is the average waiting time cost experienced by the showing patients in day $d$. 

56
Instead of total patient wait time, we use the average wait time because it evaluates the waiting time that customers expect by showing up at their appointment.

3.3. Appointment Request Scheduling Procedure

We now define how to schedule a given appointment request $q$ within the scheduling horizon, composed of the days $d = 0, ..., h - 1$ (the current day is $d = 0$). The Appointment Request Scheduling (ARS) procedure, which is summarized in Figure 9, is executed every time a new appointment request $q$ arrives, and results in the assignment of $q$ to an empty slot or in the rejection of $q$. The input of the procedure is composed of the predicted characteristics of $q$ ($s^p, r^p, \omega_c$) and those of the appointment requests that are already present in the existing schedule of days $d = 0, ..., h - 1$. First, a set of scenarios $S$ is generated to describe the forecasted arrival of future appointment requests. Then, we solve a robust optimization problem that simultaneously schedules the generated appointment requests and $q$ to the empty slots of the existing schedule, by ensuring that $q$ is scheduled in the same slot across the scenarios, to ensure a unique solution.

**Figure 9:** Appointment Request Scheduling Procedure

1. Offline Procedure. Find the non-dominated sequences for each day $d$ and slot $j$
2. Appointment request $q$ arrives
3. Generate scenarios
4. Set up the ARS model with continuous variables
5. Solve the ARS problem and obtain values of the dual variables
6. Online Procedure. Solve (6) and possibly add columns to the ARS model
7. If columns are added to the ARS model, go to 5
8. Transform the variables to binary while keeping the columns added in steps 5-7
9. Solve the IP version of the ARS problem obtained and return the solution
Step 1, which is described in Section 3.4, has the goal of speeding up the computations needed in step 6. In step 2, the appointment request \( q \) arrives. In step 3, we generate \( S \) forecast scenarios, each associated to a set of fictitious appointment requests that may arrive in the future. Mathematically, each scenario \( s \) is described by \( h \) sets \( M^s(d) \), composed of the appointment requests that are forecasted to arrive during day \( d \) \((d = 0, \ldots, h-1)\) under scenario \( s \) \((s = 1, \ldots, S)\). The generation of these scenarios is outlined in section 3.5. While \( q \) can be scheduled in any empty slot of the current day or of the future days, each generated appointment request can only be scheduled in the day of its arrival or in a following one. We define \( F^s(d) \) as the set of available appointment requests that can be scheduled in day \( d \) under scenario \( s \):

\[
F^s(d) = M^s(0) \cup \ldots \cup M^s(d)
\]

Finally, any appointment request may be unassigned; in particular, if \( q \) is unassigned in the optimal solution, then \( q \) should be rejected. We refer to the generated appointment requests as “available” appointment requests, to denote that they can be scheduled in any empty slot.

In step 4, a robust optimization problem is set up, which we call ARS problem, whose objective is to maximize the expected clinic profit obtained across the scenarios. The ARS problem could be modeled by using one binary variable for each day and slot for each available appointment request, to indicate the slot and day assigned to that appointment request. The constraints would have to compute the overtime cost, the waiting time cost, and the revenue of each day under each scenario. Obviously, this would require a large number of variables and constraints, and would therefore lead to very long computational times. We decide to tackle this problem with column generation, a technique that can solve large problems in short time, which is commonly used to solve large LP problems (Barnhart et al. 1998). Under this framework, the problem formulation includes a large number of variables (column variables), which are
iteratively added during the execution of the Simplex method, so that, usually, only a small fraction is explicitly considered. In steps 4-7, the linear relaxation of the ARS model is solved as described below. Then, in steps 8-9, we generate and solve the IP version with the set of columns built in the first phase. In our case, the column variables are binary variables which indicate what subset of available appointment requests is assigned to each day under each scenario. The introduction of these variables allows us not to include in the model all the constraints that would be needed to compute the profit of each day under each scenario. Naturally, only a very small subset of column variables is included in the model built at step 4. Then, through repeated iterations of steps 5-7, column variables are added to the model until the optimal solution is found. Which column variables to include at each iteration is determined at step 6 by the solution of the One-Day Scheduling Problem, which consists of finding the optimal assignment of a set of appointment requests to the empty slots of a single day. Let $A^s(d, a)$ be the $a$-th element of the powerset of $\mathcal{F}^s(d)$, with $a = 0, ..., Z^s(d)$, $Z^s(d) = 2^{\mid \mathcal{F}^s(d) \mid} - 1$, and $A^s(d, 0) = \emptyset$. Practically, $a$ is the index over all possible subsets of $\mathcal{F}^s(d)$ pointing to subset $A^s(d, a)$.

**Definition** One-Day Scheduling Problem. Input: scenario $s$, day $d$, slot $j$

Find the optimal schedule that can be obtained by assigning $q$ to slot $j$ (if $j=0$, $q$ is not scheduled in day $d$) and some appointment requests of $\mathcal{F}^s(d)$ to the empty slots of day $d$. Solving this problem provides the index $a^* \in \{0, ..., Z^s(d)\}$ of the set of available appointment requests actually assigned to the empty slots, and the profit obtained, $profit^s(d, j, a^*)$. 
This problem can be solved in polynomial time, provided that a certain data structure has been built offline (step 1). The binary variables that identify the optimal schedule are \( q(d,j) \), which indicate if \( q \) is scheduled in slot \( j \) of day \( d \) (if \( q(d,0)=1 \), \( q \) is not scheduled in day \( d \)), and \( x^s(i,d) \), which indicate if the available appointment request \( i \in \mathcal{F}(d) \) is assigned to day \( d \) under scenario \( s \). Finally, the column variables are \( y^s(d,j,a) \), which we call “Scheduling variables”, and which are equal to 1 if and only if:

1. \( q \) is scheduled in slot \( j \) of day \( d \), and
2. \( A^s(d,a) \subseteq \mathcal{F}(d) \) is the set of available appointment requests that are scheduled in day \( d \) under scenario \( s \).

The mathematical model for the ARS problem is the following:

\[
\max \sum_{s \in S} \sum_{d=0}^{h-1} \sum_{j=0}^{K} \sum_{a=0}^{Z(d)} y^s(d,j,a) \cdot \text{profit}^s(d,j,a)
\]

s.t.

\[
\sum_{d'=a} x^s(i,d') \leq 1 \quad \forall s \in S, d = 0, \ldots, h-1, i \in M^s(d)
\]

\[
\sum_{d=0}^{h-1} \sum_{j=1}^{K} q(d,j) \leq 1
\]

\[
\sum_{j=0}^{K} \sum_{a=0}^{Z(d)} y^s(d,j,a) = 1 \quad \forall s \in S, d = 0, \ldots, h-1
\]

\[
x^s(i,d) - \sum_{j=0}^{K} \sum_{a=0}^{Z(d)} y^s(d,j,a) = 0 \quad \forall s \in S, d = 0, \ldots, h-1, i \in \mathcal{F}(d)
\]

\[
q(d,j) - \sum_{a=0}^{Z(d)} y^s(d,j,a) = 0 \quad \forall s \in S, d = 0, \ldots, h-1, j = 0, \ldots, K
\]

\[
x^s(i,d) \in \{0,1\} \quad \forall s \in S, d = 0, \ldots, h-1, i \in \mathcal{F}(d)
\]

\[
y^s(d,j,a) \in \{0,1\} \quad \forall s \in S, d = 0, \ldots, h-1, j = 0, \ldots, K, a = 0, \ldots, Z(d)
\]

\[
q(d,j) \in \{0,1\} \quad \forall d = 0, \ldots, h-1, j = 0, \ldots, K
\]
Constraint set (1) ensures that any appointment request is assigned to at most one of the days that follow its arrival. Constraint set (2) enforces \( q \) to be assigned to at most one slot. Constraint set (3) forces the selection of exactly one set \( A^s(d,a) \subseteq \mathcal{F}^s(d) \) for each day under each scenario. The selection of \( j = 0 \) indicates that \( q \) is not scheduled in day \( d \). Constraint set (4) sets \( x^s(i,d) \) to 1 if and only if the set \( A^s(d,a) \subseteq \mathcal{F}^s(d) \), scheduled in day \( d \) under scenario \( s \), contains \( i \). Constraint set (5) states that \( q \) is assigned to slot \( j \) of day \( d \) if and only if the scheduling variable selected for day \( d \) and scenario \( s \) assigns \( q \) to the \( j \)-th slot. When the model is initialized (step 4), it includes only the scheduling variables corresponding to scheduling no patient in day \( d \) under scenario \( s \), i.e., \( y^s(d,j,0) \).

Let us consider the solution of the linear relaxation found during step 5. Let \( \beta(s,d) \) be the dual variables corresponding to constraint set (3), \( \gamma(s,d,i) \) the dual variables corresponding to constraint set (4), and \( \xi(s,d,j) \) the dual variables corresponding to constraint set (5). It is easy to verify that the dual constraints corresponding to the primal variables \( y^s(d,j,a) \) are satisfied if and only if:

\[
\forall s \in S, d = 0, \ldots, h - 1, j = 1, \ldots, K: \max_{a=0, \ldots, z^s(d)} \{ \text{profit}^s(d,j,a) + \sum_{i \in A} \gamma(i,d,s) \} \leq \beta(s,d) - \xi(s,d,j)
\]  

(6)

Note that \( \max_{a=0, \ldots, z^s(d)} \{ \text{profit}^s(d,j,a) + \sum_{i \in A} \gamma(i,d,s) \} \) is the solution value of a One-Day Scheduling Problem where the original revenues are modified by the parameters \( \gamma(i,d,s) \). At step 6, we check if (6) is satisfied. If this is not the case, we add to the model the variables \( y^s(d,j,a) \) corresponding to the violated dual constraints (at most one for each \( s,d,j \)) and go back to step 5. If, on the other hand, (6) is satisfied, then the optimal solution to the linear
relaxation is found. At this point, at steps 8 and 9, we solve the IP obtained by converting the variables into binary and by keeping the columns added in the executions of steps 5-7. In this way, we obtain the solution to the ARS problem. If there is a variable $q(d,j)$ that is equal to 1, then $d$ and $j$ respectively represent the day and slot where $q$ should be scheduled. If, on the other hand, $q(d,j) = 0$ for all values of $d$ and $j$, then the optimal decision is to reject $q$. Note that it is straightforward to allow patients' preferences about appointment time by including simple constraints on the variables $q(d,j)$.

### 3.4. Solving the One Day Scheduling Problem

At each iteration of the column generation procedure, the One Day Scheduling Problem

$$\max_{a=0,\ldots,Z^s(d)} \{profit^s(d,j,a) + \sum_{i \in A} \gamma(i,d,s)\}$$

must be solved for each scenario $s = 1,\ldots,S$, for each day $d = 0,\ldots,h-1$, and for each slot $j = 0,\ldots,K$. The value of $P^s(d,j)$ is found by optimally scheduling the appointment requests of $F^s(d)$ in the available slots of day $d$, fixing the appointment request $q$ in slot $j$ ($j = 0$ corresponds to the case when $q$ is not scheduled in day $d$). At step 1, we run an “offline procedure” to build a data structure that speeds up the solution of the problem. At step 6, we execute a polynomial time algorithm, called “online procedure”, several times during the column generation procedure.

The existing schedule of a day can be viewed as a string of ‘s’, ‘n’, and ‘_’ characters, where each character denotes if the appointment request in that position is predicted to be a show (‘s’), a no-show (‘n’), or if the slot is still empty (‘_’). For example, s_n_n indicates that the first slot has already been assigned to a predicted show, the third and the fifth slots to predicted no-shows, while the second and fourth slots have not been assigned. We refer to such strings as “sn-sequences” or, more simply, “sequences”. When a sequence contains a ‘_’, it is a partial
sequence; if, on the other hand, it does not contain any ‘_’, it is a complete sequence. Note that a complete sequence $S$ determines the waiting times that will be experienced by the patients, as well as the clinic overtime. For example, assuming $D = 30$ and $T = 20$, in the sequence ssnsss, the patients in the $1^{st}$ and $4^{th}$ slot wait 0 minutes, the ones in the $2^{nd}$ and the $5^{th}$ slot wait 10 minutes, the $6^{th}$ patient waits 20 minutes, and the overtime is 30 minutes. The offline procedure completes the partial sequence of each day by filling the empty slots (‘_’) with ‘s’ and ‘n’, in order to identify the non-dominated complete sequences, i.e. the ones that result in the largest profit. To this end, starting from the left of the sequence, the algorithm fills the ‘_’ with ‘s’ or ‘n’ in order to generate a longer sequence that is not dominated by already generated other sequences.

The complexity of the offline procedure is exponential, but this does not necessarily have any impact on the clinic operations. In fact, it can be executed after an appointment is scheduled; therefore, the computing time does not need to be very short, as long as it is less than the time between two appointment requests’ arrivals.

The online procedure is executed for each scenario $s = 1,\ldots, S$, for each day $d = 0,\ldots, h-1$, and for each slot $j = 0,\ldots, K$, and for each non-dominated sequence $e = 1,\ldots, E$ found by the offline procedure. The problem consists of finding the optimal assignment of the shows and the no-shows of $F^s(d)$ to the ‘s’ and ‘n’ of $e$, with the constraint of $q$ being assigned to slot $j$ (or not scheduled in day $d$, if $j = 0$). This problem can be solved in $O(cn)$, where $n$ is the number of appointment requests in $F^s(d)$ and $c$ the number of waiting time cost categories. Therefore, the overall complexity of step 6 is $O(cnShKE)$. The details of the online procedure are included in Appendix B.
3.5. Experimental Results

In this section, we test our method through a simulation procedure. We simulate 100 days of clinic sessions, where appointment requests arrive in the mornings and patients are visited in the afternoons. We fix the clinic capacity to $N = 8$, which is a realistic value if the appointment duration $D$ is, for example, 30 minutes. Larger values of $N$ result in larger revenues, because more patients would be seen, but also in larger costs, because longer sequences of consecutive shows would be experienced. The input of the procedure is composed by several parameters; some can be fixed or controlled by the clinic management and some cannot be controlled. The controllable parameters are the scheduling horizon $h$, the number of slots $K$, the sensitivity $sn$, the specificity $sp$. In this paper, the sensitivity is the probability of correctly predicting a no-show, while the specificity is the probability of correctly predicting a show. The non-controllable parameters are the arrival rate of appointment requests $\lambda$, the show rate $R$, the profit for scheduling a show $\pi$, the overtime cost per time unit $\tau$, the average waiting time cost per time unit $\bar{\omega}$. The output of the procedure is the clinic performance, measured by the expected clinic profit $\bar{p}$, the expected average waiting time $\bar{wt}$, the expected overtime $\bar{\sigma}$, and the expected number of shows $\bar{s}$. Any given combination of simulation parameters results in a certain clinic performance. Although the clinic performance could be simply represented by the clinic profit $\bar{p}$, which implicitly takes into account $\bar{wt}$, $\bar{\sigma}$, and $\bar{s}$, we prefer to consider the other measures too. In fact, $\bar{p}$ is an accurate measure of the clinic profit only if the simulation parameters $\pi$, $\tau$, and $\bar{\omega}$ are accurate, but they are often hard to set. In Section 3.5.2, we show that, by considering also the other measures, the user can choose between clinic designs that result in similar values of $\bar{p}$, but have very different values of $\bar{wt}$, $\bar{\sigma}$, and $\bar{s}$. 
In the mornings, appointment requests arrive according to a Poisson process with parameter $\lambda$, and need to be scheduled in the afternoon of the same day or in one of the following $h-1$ days. We assume that their show probability tends to decrease with the increase of the lead time (Liu et al., 2010), which occurs at the clinical system we studied. The clinics’ show rates are reported in the second row of Table 12, together with their average (.76). Our analysis, though, is not limited to these values, but also considers a case where the show rates are lower and a case where they are higher, as indicated in Table 12. The “high show rates” are computed by multiplying the clinics’ show rates by a factor such that the show rate for same day appointments is equal to 1.00, which is the assumption of most works on open access (see, for example, Robinson and Chen 2010). The “low show rates” are computed by dividing the show rates by a factor, in order to obtain an average show rate of .65. In this way, the three average show rates $R = .65$, $R = .76$, and $R = .87$ are “equidistant”. Although we consider values of $h \leq 5$ throughout Section 3.5, we show in Section 3.6 that a longer scheduling horizon does not change the “behavior” of the scheduling algorithm.

<table>
<thead>
<tr>
<th>Show rate</th>
<th>Same day</th>
<th>1 day</th>
<th>2 days</th>
<th>3 days</th>
<th>4 days</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>.74</td>
<td>.64</td>
<td>.65</td>
<td>.62</td>
<td>.61</td>
<td>.65</td>
</tr>
<tr>
<td>MHCD</td>
<td>.87</td>
<td>.74</td>
<td>.75</td>
<td>.72</td>
<td>.71</td>
<td>.76</td>
</tr>
<tr>
<td>High</td>
<td>1.00</td>
<td>.85</td>
<td>.87</td>
<td>.83</td>
<td>.81</td>
<td>.87</td>
</tr>
</tbody>
</table>

When an appointment request arrives, the “real show” vector $s^r$ is randomly generated according to Table 12 depending on the value of $R$ considered. The “real revenue” vector $r^r$ is generated from vector $s^r$ as explained in section 3.2, using $\pi$ as revenue for scheduling a show.
The components of the “predicted show” vector $s^p$ are provided by a given classifier, which predicts if the appointment request shows or not for each day of the scheduling horizon. The predicted revenue vector $r^p$ is generated from vector $s^p$, as outlined in section 3.2. We consider two categories of patients, whose waiting time costs are $\omega_1 = 0.5\bar{\omega}$ and $\omega_2 = 1.5\bar{\omega}$. An arriving appointment request is equally likely to belong to category 1 or 2. Finally, $\bar{\pi}$ is fixed at 0.05 and $\delta$ to 0.005, so that the goal of scheduling a patient early is secondary to the goals of maximizing the shows and minimizing overtime and waiting time costs.

After the arrival of an appointment request, the scheduling problem is solved. We generate three forecast scenarios, in which the arrival rate of the appointment requests is the real arrival rate $\lambda$, and the characteristics of each appointment request are generated in the same way as the real appointment requests, as described above. After solving the scheduling problem, if the appointment request is assigned to a slot, the clinic schedule is updated.

At the end of each day $d$ of the simulation, we consider the real characteristics of the appointment requests scheduled in $d$, and compute the real clinic profit $p_d$, the average waiting time $wt_d$, the overtime $o_d$, and the number of shows $s_d$. The results of the first 5 days of the simulation are excluded because during this start-up period the schedule is mostly empty; the last 5 days are excluded because the scheduling problem solved may have a shorter scheduling horizon than $h$. At the end of the simulation, we compute the average of the values recorded in each day: $\bar{p} = \frac{\sum_{d=6..95} p_d}{90}$, $\bar{o} = \frac{\sum_{d=6..95} o_d}{90}$, $\bar{wt} = \frac{\sum_{d=6..95} wt_d}{90}$, $\bar{s} = \frac{\sum_{d=6..95} s_d}{90}$.

### 3.5.1 Full Factorial Design

In this section, we consider the impact of the simulation parameters on the clinic performance. To this end, we set up a full factorial design where we analyze the results of the simulation for
different combinations of parameters (or factors of the experiment). Table 13 reports the factors and their levels.

We use 3 levels for the controllable factors $h$, $sn$, and $sp$, because their impact on the clinic performance has not been studied in previous works, while we use 2 levels for all other factors, because their impact has already been assessed (LaGanga and Lawrence, 2007b). So, for every combination of parameters ($2^6 \times 3^3 = 1728$ combinations), we run a 100-day simulation and record the measures $\bar{p}$, $\bar{wt}$, $\bar{\sigma}$, and $\bar{s}$. The same random seed is used, so that the appointment request characteristics and their arrival time are the same for any parameter combination, as well as the scenarios used to solve the scheduling problems. The software package used, Minitab 16, is limited to the analysis of the interactions of up to 2 factors. Although all main effects and the majority of the 2-factor interactions are significant at $\alpha=0.01$, we discuss only the effects involving at least one controllable parameter. Furthermore, we consider the impact of overtime but not that of waiting time because they are very similar.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>Number of Levels</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>Arrival rate</td>
<td>2</td>
<td>12, 15</td>
</tr>
<tr>
<td>$R$</td>
<td>Show rate</td>
<td>2</td>
<td>.65, .87</td>
</tr>
<tr>
<td>$\pi$</td>
<td>Revenue for shows</td>
<td>2</td>
<td>1, 3</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Overtime cost</td>
<td>2</td>
<td>1, 3</td>
</tr>
<tr>
<td>$\bar{\omega}$</td>
<td>Average waiting time cost</td>
<td>2</td>
<td>1, 3</td>
</tr>
<tr>
<td>$h$</td>
<td>Scheduling horizon</td>
<td>3</td>
<td>1, 3, 5</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of slots</td>
<td>2</td>
<td>9, 12</td>
</tr>
<tr>
<td>$sn$</td>
<td>Sensitivity</td>
<td>3</td>
<td>.4, .7, .9</td>
</tr>
<tr>
<td>$sp$</td>
<td>Specificity</td>
<td>3</td>
<td>.4, .7, .9</td>
</tr>
</tbody>
</table>
Figure 10 reports the impact of the controllable factors on overtime (dashed line), shows (continuous line), and clinic profit (bars and bold numbers). As expected, $K$ increases patient access (10a), whereas the scheduling horizon $h$ does not have a strong impact on any of the measures of clinic performance (10b). Nevertheless, we note below that its interaction with other factors more strongly affects the performance. Like $K$, also the sensitivity $sn$ increases patient access (Figure 10c). Unlike in the case of $K$, the extra profit obtained here by the increase of shows offsets the extra cost caused by the overtime and waiting time increase, resulting in a higher profit. In case of high sensitivity, when an appointment request is predicted to be a “show”, it rarely happens to be a “no-show” (i.e., the prediction is correct), because the “no-shows” are mostly classified correctly. Thus, it will be a “show” in most cases. Therefore, a highly sensitive classifier is good at “retrieving” the showing appointment requests. This task is particularly important when the show rate $R$ is low, when $\tau$ is low, or $\pi$ is large. For example, Figure 10a shows that, when $R$ is low, the profit (bars) increases when $sn$ increases. This trend, which is caused by the extra number of shows (lines) retrieved by the classifier, is much less visible when $R$ is large. Also, a large $sn$ makes $h$ positively correlated with the profit. Figure 10b suggests that, when $sn$ is large, the profit increases when $h$ increases. In fact, same-day scheduling ($h = 1$) is the best strategy only when $sn = 0.4$; otherwise, larger values of $h$ are preferred.

The specificity $sp$ decreases overtime (Figure 10d). Unlike a highly sensitive classifier, a highly specific classifier is very good at retrieving no-shows, which can be effectively used to reduce waiting times and overtime. This task is particularly important if $\tau$, $R$, or $\bar{\omega}$ are large, or if $\pi$ is low. For example, Figure 11c shows that, if overtime cost $\tau$ is large, increasing $sp$, the accuracy in predicting shows, leads to a dramatic reduction in the number of shows (and
overtime and waiting time), eventually resulting in a higher profit. Figure 11d shows the interaction between $sp$ and $h$. Although the interaction is weaker than the one between $sn$ and $h$, we note that, also in this case, same-day scheduling ($h = 1$) leads to the highest profit only if $sp = 0.4$, whereas a larger value of $h$ is preferred if $sp$ is large. Our results suggest that same-day scheduling is the best policy only if no effort is made to predict if patients show or not.

**Figure 10:** Impact of the controllable parameters

![Figure 10](image)
3.5.2 A real world example

In this section, we use a simulation optimization approach to find the best clinic design for the clinical system we studied. By considering their appointment data and by fixing certain parameter values, we aim at individuating optimal values for the number of slots $K$ and for the scheduling horizon $h$, and an optimal classification technique to use (which defines $sn$ and $sp$).

We consider $N = 8$, $D = 30$ minutes, $R = 0.76$ (the actual show rate). We also assume that the arrival rate is $\lambda = 15$ appointment requests per day. In this way, the clinic receives many more appointment requests than it is able to serve, which is often the case. Finally, we assume $\pi = 1$, $\tau = 1.2$, $\bar{\omega} = .5$, as proposed by LaGanga and Lawrence, (2007b).
Note that, among the controllable parameters, $sn$ and $sp$ cannot be set arbitrarily. In fact, they depend on the level of difficulty to distinguish a showing appointment request from a non-showing appointment request. If this problem was trivial, then we could reach a perfect accuracy (i.e. $sn = sp = 1$); if, on the other end of the spectrum, the two types of appointment requests were absolutely undistinguishable, we could only reach a random accuracy, such as $sn = sp = 0.5$. Although they cannot be set arbitrarily, it is possible to tweak them within certain limits. Therefore, let us test our method in two phases. First, we analyze the data in order to estimate what combinations of $sn$ and $sp$ can be obtained on a new unseen appointment. Then, we run a simulation for many combinations of $K$, $h$, $sn$, and $sp$, and we find the best values of these parameters.

The center’s database includes the details of about 50,000 appointments and 6,700 clients. All protected health information was removed or coded to protect patients’ rights to privacy. Each appointment is performed by a certain staff and a certain service team and can result in a show or in a no-show. The staff is a provider such as a doctor or a nurse; the service team is the program organized to deliver appropriate levels and types of services. Each client is periodically evaluated through two types of outcome indicators proprietary to MHCD: the Recovery Marker Inventory (RMI) and Consumer Recovery Measures (CRM). RMI scores are evaluations of consumer progress, assessed by the provider, on different aspects of the client’s life, such as his/her job, housing situation, and so forth. On the other hand, CRM scores are self-evaluations of the clients, who are asked how they cope with symptoms, what their level of hope is, and so on.

In order to use a classification algorithm, we need to build a “mining table”, which has one row for each appointment and one column for each characteristic (or attribute) of the
appointment. One more column is the target attribute, which, in our case, is a Boolean value that indicates whether the appointment was a no-show (true) or a show (false). In the training phase, the classifier “learns” the difference shows and no-shows; then, it predicts the target attribute of a new appointment whose remaining attributes are known. We build the mining table by using the Propositionalization algorithm proposed by Samorani et al. (2010), which explores the database and automatically generates attributes for Appointments.

Now, we assess the classification performance that can be obtained on the center’s data. To this end, half of the data set is used as training set and half as test set. Our experiments are run on Weka (Hall et al. 2007), an open source data mining software available for download at http://www.cs.waikato.ac.nz/ml/weka/ (as of October 19th, 2011). Considering only the training set, we run several 3-fold cross validations using different classification techniques, with the goal of finding a suitable one. Through a trial-and-error approach, we find that the Bayesian Network classifier called BayesNet can obtain any of the following performances on the test set: $(sn = .6, sp = .8), (sn = .9, sp = .5), \text{ and } (sn = sp = .7)$. These configurations, which correspond to a “high-specificity”, a “high-sensitivity”, and a “balanced” classifier, are obtained by varying the weight of misclassifying shows and no-shows, so that the classifier favors one type of prediction, the other, or none. These three classifiers are the output of the first phase. Note that the classification rule does not need to be obtained through data mining, but could be obtained by using any statistical tool. In general, a condition that is satisfied by $x\%$ of the no-shows and $y\%$ of the shows corresponds to a classification rule (or classifier) having $sn = x\%$ and $sp = y\%$.

The goal of the second phase is to use these classifiers to establish which parameter values obtain the best clinic performance. We test our method by running the same simulation procedure as in section 3.5.1. We run a simulation procedure for different combinations of the
parameters that have not been fixed: $k$, $h$, $sn$, and $sp$. The parameter values that we consider are: $k \in \{8, 9, 10, 11, 12\}$, $h \in \{1, 2, 3, 4, 5\}$, $(sn, sp) \in \{(0.9, 0.5), (0.7, 0.7), (0.6, 0.8), (1, 0)\}$. We include $(sn = 1, sp = 0)$ because, as noted below, it represents the case where data mining is not used. Some “limit cases” arise when these parameters have certain “limit” values:

- **Same-day scheduling ($h = 1$)**. If $h = 1$, each appointment request is either scheduled on the same day or rejected, resulting in an “same-day scheduling” policy.

- **Data mining ($sn \neq 1$ or $sp \neq 0$)**. If $sn = 1$ and $sp = 0$, the classifier predicts every appointment request to be a no-show. Thus, appointment requests will be scheduled without a particular order, because the predicted overtime and waiting time costs are 0. In all the other cases ($sn \neq 1$ or $sp \neq 0$), the scheduling order of appointment requests is taken into account. Let us label this case simply with “data mining” (DM).

- **Overbooking ($K > 8$)**. If $K > 8$, then $K > N$; therefore, we are overbooking.

For all combinations of $k$, $h$, $sn$, $sp$, we execute a 100-day simulation. In view of the limit cases reported above, depending on the values of $k$, $h$, $sn$, and $sp$, there are three possible features that the clinic may implement: using DM, using overbooking (OB), using same-day scheduling (SD). Therefore, there are 8 possible policies that the clinic may adopt, depending on which features are implemented. For each of the 8 policies defined above, Table 14 reports the values $K^*$ and $h^*$ that result in the best clinic performance. Since 3 classification performances are available, the policies that make use of data mining include 3 different sub-cases.
Table 14. Best results for each combination of the policies “data mining” (DM), “overbooking” (OB), “same-day scheduling” (SD).

<table>
<thead>
<tr>
<th>Policy</th>
<th>DM</th>
<th>OB</th>
<th>SD</th>
<th>$R = 0.76$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\bar{p}$</td>
</tr>
<tr>
<td>1</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>7.28</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>7.27</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>7.47</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>7.52</td>
</tr>
<tr>
<td>.6, .8</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>7.49</td>
</tr>
<tr>
<td>5</td>
<td>.7, .7</td>
<td>No</td>
<td>No</td>
<td>7.56</td>
</tr>
<tr>
<td>.9, .5</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>7.85</td>
</tr>
<tr>
<td>.6, .8</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>7.56</td>
</tr>
<tr>
<td>6</td>
<td>.7, .7</td>
<td>No</td>
<td>Yes</td>
<td>7.59</td>
</tr>
<tr>
<td>.9, .5</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>7.52</td>
</tr>
<tr>
<td>.6, .8</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>7.60</td>
</tr>
<tr>
<td>7</td>
<td>.7, .7</td>
<td>Yes</td>
<td>No</td>
<td>7.65</td>
</tr>
<tr>
<td>.9, .5</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>7.86</td>
</tr>
<tr>
<td>.6, .8</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>7.62</td>
</tr>
<tr>
<td>8</td>
<td>.7, .7</td>
<td>Yes</td>
<td>Yes</td>
<td>7.64</td>
</tr>
<tr>
<td>.9, .5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>7.57</td>
</tr>
</tbody>
</table>

First, note that each policy that uses DM leads to a larger profit than the corresponding policy that does not use it (i.e, policy 5 is better than policy 1, policy 6 is better than policy 2, etc…). The largest profit among the cases where DM is not used (policies 1-4) is obtained by adopting SD and OB (policy 4). This profit is only 3.4% larger than that obtained with SD only (policy 2). The improvement obtained by OB is small because the show rate considered is quite high. On the other hand, the largest profit among the cases where DM is used (policies 5-8) is obtained by adopting OB but not SD (policy 7). This reflects the findings of section 3.5.1 (i.e.,
SD is the worst choice if we use DM). This profit is 8.0% larger than that obtained by policy 2, which is one of the most popular policies in appointment scheduling. There are two reasons for this. First, although SD guarantees a show rate of 87% (Table 12, “Same-day” column), a DM-based algorithm can exploit the longer scheduling horizon by individuating a higher proportion of showing patients. For example, if a patient is predicted not to show today but to show tomorrow, he or she will be likely to be scheduled tomorrow. Second, if OB is used, then using also DM leads to a smaller overtime and waiting time than not using DM, because the sequence of shows and no-shows obtained is the one that minimizes the chance of long overtime and waiting times. Note that a very similar profit to that of policy 7 can be obtained by adopting DM only, with no OB and no SD (policy 5). Although the number of patients seen daily decreases by 8.7%, the overtime and waiting time are eliminated. Note that the profit obtained by policy 5 is 7.8% higher than that of the corresponding policy that does not use DM (policy 1). This example shows that the use of DM leads to a significantly higher clinic profit, both with and without overbooking. MHCD should therefore choose between adopting policy 5 and policy 7.

The average time taken to schedule the current appointment request is 0.19”, while the average time taken to update the data structure (offline procedure) is 0.04”. Neither procedure takes more than 1 second in our experiments. The average gap between the optimal solution of the LP relaxation and the integer solution is 3.7% with standard deviation = 2.8%.

### 3.5.3 Variable Service Times

In order to relax the assumption of constant service times, we performed a sensitivity analysis which consists of running the simulations in Table 14 with variable service times. To this end, we model the service times with a Gamma distribution ($\alpha, \beta$) with parameter pairs of (8.0, .125),
(3.0, .333), (2.0, .5), and (1.0, 1.0), as in LaGanga and Lawrence (2007b), in order to study the impact of different levels of uncertainty.

Table 15 shows the average difference between the profits reported in Table 14 and the profits made in case of service time uncertainty. The rows of Table 15 correspond to the subset of experiments where DM is used, the one where OB is used, and the one where SD is used. The columns correspond to the different levels of uncertainty, from the least uncertain (left) to the most uncertain configuration (right). Unsurprisingly, the clinic profit is reduced proportionally to the level of uncertainty in service time; but, for the same level, this reduction is similar across all policies. Therefore, the advantage of DM and OB discussed above are the same even in the case of variable service times.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Service Time Variability (α, β)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8.0, .125</td>
</tr>
<tr>
<td>DM</td>
<td>-6.5%</td>
</tr>
<tr>
<td>OB</td>
<td>-4.6%</td>
</tr>
<tr>
<td>SD</td>
<td>-6.4%</td>
</tr>
</tbody>
</table>

3.5.4. Implementation of a Simple Scheduling Policy

In this section, we define and test a simple scheduling policy (SSP) that can be used in place of the ARS procedure (described in the section 3.3) and that leads to a similar performance. An analysis of the schedules obtained throughout our experiments reveals that the ARS procedure tries to achieve the same best complete sn-sequence every day. Table 16 shows the sequences targeted by the algorithm for different values of N and K. If, for example, N =8 and K =12
(highlighted in Table 16), an appointment request that is predicted not to show will be scheduled in slot 3, 6, 9, or 12.

**Table 16:** Target complete sequence of shows and no-shows (sn-sequences) for different combinations of capacity $N$ and appointments booked $K$

<table>
<thead>
<tr>
<th></th>
<th>$K = N$</th>
<th>$K = N+1$</th>
<th>$K = N+2$</th>
<th>$K = N+3$</th>
<th>$K = N+4$</th>
<th>$K = N+5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N=4$</td>
<td>ssss</td>
<td>ssssn</td>
<td>ssnsn</td>
<td>ssnsn</td>
<td>ssnsn</td>
<td>ssnsn</td>
</tr>
<tr>
<td>$N=5$</td>
<td>sssss</td>
<td>sssssn</td>
<td>ssnsn</td>
<td>ssnsn</td>
<td>ssnsn</td>
<td>ssnsn</td>
</tr>
<tr>
<td>$N=6$</td>
<td>sssssss</td>
<td>ssssssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
</tr>
<tr>
<td>$N=7$</td>
<td>ssssssss</td>
<td>sssssssn</td>
<td>sssnsnssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
</tr>
<tr>
<td>$N=8$</td>
<td>sssssssss</td>
<td>ssssssssn</td>
<td>sssnsnsssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
<td>ssnsnssn</td>
</tr>
</tbody>
</table>

Note that the sequences in Table 16 depend on the non-controllable parameters and will likely change if these parameters change.

Our analysis also suggests that predicted no-shows tend to be scheduled far in advance, while predicted shows tend to be scheduled at the last moment. This counterintuitive behavior of our procedure is simply explained by considering that, to maximize the likelihood of obtaining a no-show in a certain slot, it is advisable to schedule patients far in advance. Conversely, to maximize the likelihood of obtaining a show in a certain slot, it is advisable to schedule patients at the last moment, for example by using a same-day or a next-day appointment. The practical implication is that same- or next-day slots, which are highly desirable, should be assigned to those who will utilize them because the chance of filling such allocated slots goes down as the day progresses. The clinic should avoid having reserved same-day slots go unused because they were assigned to patients who didn’t show up. On the other hand, there are more days and hence scheduling opportunities to fill up appointment schedules for days farther in advance, so the effect of long-term no-shows can be mitigated by scheduling more patients.
To test the validity of these observations, we repeat the experiments of section 3.5.2 by considering the target sequence corresponding to the value of $K$ used and by scheduling the current appointment request $q$ according to policy in Figure 12, which aims at scheduling the shows at the last moment and the no-shows as far in advance as possible:

![Figure 12: Simple Scheduling Policy](image)

Surprisingly, this simple policy leads to an average performance which is only 2.9% smaller than that obtained by the ARS procedure. This is also true for large values of the scheduling horizon ($h = 30, 60$). In conclusion, a clinic can use the SSP in place of the ARP at the cost of a modest decrease in performance.

3.6. Managerial Insights and Conclusions
In this paper, we present a new solution method to schedule appointments based on the prediction, obtained through data mining, of whether the appointment will result in a show or in a no-show. Our procedure allows the decision maker to independently adopt different strategies to tackle the no-show problem: same-day scheduling, overbooking, and data mining. We embed
our method in a simulation procedure and test it on the data set of a real world organization. The simulation optimization approach allows the decision maker to identify the best set of strategies (DM, OB, SD) that should be used to maximize the clinic profit. Our framework can therefore be used by clinics as a “data-driven design tool”, which, given the data, finds the optimal clinic setup. The findings, which emerged through our experimental studies, are summarized here.

First, using data mining to obtain individual show predictions always improves the clinic performance. At the clinic system we studied, the adoption of DM can improve the clinic performance by up to 7.8%. Second, using data mining on top of overbooking tends to reduce overtime and waiting time. Third, same-day scheduling is the worst policy if individual show predictions are considered. Fourth, appointment classifiers that are good at predicting no-shows but bad at predicting shows lead to high patient access, but incur high overtime and high waiting time. On the other hand, classifiers that are good at predicting shows but bad at predicting no-shows lead to low patient access, low overtime, and low waiting time – in other word, they incur little cost but add little value in increasing patient access. Fifth, a clinic can improve the performance significantly by implementing the ARS procedure (Sections 3.3-3.4), or by applying our SSP as outlined in Section 3.6. After obtaining the optimal sn-sequence from Table 16, SSP is implemented by scheduling the predicted shows in ‘s’ slots as close as possible to the appointment date (e.g., with a same-day or next-day appointment) and the predicted no-shows in the ‘n’ slots as far ahead as possible (e.g., months in advance).

Finally, the problem that we solve is more general than the traditional appointment scheduling problem because it allows different approaches such as data mining, along with policies such as same-day scheduling and overbooking, used singly or in combination. A unique contribution of our model is its consideration of individual waiting costs along with custom
revenue contributions. These realistic and generalizable assumptions make our solution method suitable for use in a variety of clinical and general service settings.

We are in the process of piloting and implementing our scheduling procedure in an actual MHCD clinic to begin the study of its impact on actual operations. Other future research includes the adoption of techniques to improve the method, such as dynamic programming to allow individual show probabilities, rather than binary show outcomes. The expanding use of health information technology, especially as the American Reinvestment and Recovery Act provides incentives for providers to make meaningful use of electronic health records, leads to growing numbers of clinics developing more complete and useful databases of patient and service information. This provides fertile ground to identify meaningful performance-predicting attributes to improve schedule performance and to further develop our data mining approaches. The higher accuracy that can be achieved will lead to higher benefits of data mining.
4. DATA-MINING-DRIVEN NEIGHBORHOOD SEARCH

While the second paper shows how classification can improve operations in a particular problem, in Samorani and Laguna (2011) we propose a general approach that uses classification to improve the solution quality to any optimization problem.

4.1. Introduction

Heuristic and metaheuristic techniques (Glover and Kochenberger 2003) are widely used for finding high quality solutions to large combinatorial optimization problems, for which identifying the optimal solution through an exact procedure would be computationally impractical. Some metaheuristic approaches are based on maintaining a set of solutions and creating new ones by way of executing combination procedures. Others, however, use a neighborhood search (NS) to move from one point to another in the solution space. The neighborhood of a solution is defined by the move mechanisms that are applied to transform one solution into another. Local search is the simplest form of a NS procedure, which limits the moves to those that direct the search from the current solution to a neighboring solution with a better objective function value. In other words, a local search accepts only improving moves to transform the current solution into one that is in its neighborhood. The process stops when no improving move is available and the resulting solution is called a local optimum (with respect to the defined neighborhood).

Metaheuristic procedures include strategies that aim at escaping from local optima with the goal of improving the best solution found during the search. In tabu search, escaping local optima is achieved by imposing tabu restrictions (Glover and Laguna 1997) or executing moves that are guided by proxy objective functions, as in path relinking (Glover, Laguna, and Marti
Simulated annealing relies on randomization to allow non-improving moves to be executed in order to escape local optimal points. Greedy Randomized Adaptive Search Procedures (GRASP) restart the search after reaching a local optimum. A common aspect to these strategies is that escape directions are determined by rules that are defined a-priori. For instance, tabu activation rules forbid certain moves by restricting a subset of attributes from assuming certain values for a predetermined number of iterations. In path relinking, only moves that take the current solution “closer” to a guiding solution are considered. In general, escaping from local optima in a NS-based approach requires the execution of a non-improving move and the reduction of the neighborhood. Often, the neighborhood reduction is achieved by enforcing guiding constraints. These guiding constraints are the result of design decisions that may or may not be customized to a particular class of problems. For instance, a tabu search approach for permutation problems may use swaps to explore the neighborhood of a given solution. After a swap of two elements, a tabu restriction could be imposed to forbid the exchange of positions of these two elements for a number of pre-specified iterations. This tabu-activation rule effectively restricts the available moves and hence shrinks the neighborhood of the current solution. In Variable Neighborhood Search (VNS), a metaheuristic methodology suggested by Mladenovic and Hansen (1997), escaping from local optima requires the systematic change from one neighborhood to another. This is achieved by embedding a set of move mechanisms of various degrees of complexity. Simple moves are tried first, followed by more complex moves as the search becomes trapped in local optima corresponding to the neighborhood defined by the moves currently under consideration. Both the set of moves (and therefore neighborhoods) to be explored and the order in which they will be considered are specified a-priori, a task that often requires a considerable effort in the development phase of the solution procedure.
Our goal is to define a general learning model, valid for any class of problems, whose output is the definition of the best set of guiding constraints and the conditions for their applications. The learning is performed offline by analyzing pairs of local optima drawn from a training set of problem instances belonging to a class under consideration. The learning process focuses on formulating constraints that guide the search in effective directions to escape local optimality. Effectiveness is measured by the ability to improve the current position after executing non-improving moves that are necessary to leave the local optimal point. We show how the rules learned during the offline process can be embedded in search procedures to be used online for tackling instances of the same class of problems. The process requires the following four elements:

1. A procedure to randomly generate feasible solutions for a given problem instance
2. A set of basic moves that define the neighborhood search
3. An attribute representation of a solution
4. An attribute representation of a pair of solutions

where the “attribute representation” of an object is a numerical vector whose components characterize the object.

Some work has already been done to couple machine learning and data mining (DM) techniques in meta-heuristic searches, as reported by Boyan, Buntine and Jagota (2000) and Jourdan, Dhaenens and Talbi (2006). In particular, the existing learning methods can be classified into two categories: online and offline methods. Both of these approaches are learning procedures that — after a training phase — are used within a search process in an attempt to
improve its effectiveness. The main difference between the two approaches lies on the training: online methods are trained using information collected during the current execution of the search procedure, while offline methods learn from a training set of instances of the same class of problems. Online procedures are used, for instance, to select a branch in a branch-and-bound algorithm given information gathered during the search itself, as in (Glover, Klingman, and Phillips 1989), or to find patterns useful to enhance the construction phase of GRASP, as proposed by Santos et al. (2008). On the other hand, offline procedures attempt to identify structures and patterns shared by the entire class of problems. In the literature, offline approaches have been further divided into two categories, depending on their goal: 1) algorithm-selection approaches and 2) parameter-tuning approaches. Algorithm-selection approaches use historical performance data with the goal of selecting the most promising procedure from a portfolio of available ones. Parameter-tuning approaches have the goal of finding the set of parameter values that will make a particular procedure perform at the highest level. Our work is an offline approach that cannot be directly casted as an algorithm-selection or parameter-tuning approach. We argue that our approach defines a new category of offline methods that may be referred to as \textit{data mining driven neighborhood search} (DMDNS). This category refers to search procedures that modify neighborhoods with constraints that have been learned offline by mining data. Padmanabhan and Tuzhilin (2003) point out that this area has been largely neglected in the literature. Broadly speaking, DMDNS consists of two components: 1) an offline learning procedure that learns the guiding rules for escaping local optimality, and 2) a procedure that applies the guiding rules online. The offline learning procedure may be viewed as a process that is in some ways similar to fitness landscape analysis (Höhn and Reeves 1996). Both approaches attempt to characterize the search space. However, while the results of fitness
landscape analysis are interpreted by the analysis in order to design global search strategies (e.g., effective neighborhood functions), the results of a learning procedure such as the one we propose are directly embedded in the neighborhood search in the form of constraints that are activated upon reaching local optima. In this work, we propose a \textit{data mining driven tabu search} (\textit{DMDS}) and we show that our approach leads to improved escape directions when compared to those resulting from enforcing customary tabu search rules. We intend to establish a new research avenue in the development of learning procedures whose purpose is to capture aspects of the structure of a given class of problems by considering a “training set” of instances. The approach focuses on learning “off line” and only once for a given class of problems.

We will be using the Constrained Task Allocation Problem (CTAP) as an example throughout the paper in order to enhance the presentation of our concepts. We then apply the method to the Matrix Bandwidth Minimization Problem (MBMP), in an attempt to show its general application. We realize, however, that to claim success in a variety of problem settings, additional applications are required. We hope that our work inspires others to follow this line of research and expand the applicability of the concepts that we introduce.

### 4.2. Learning Procedure

Our goal is to develop a process that learns how to move from a current local optimum to a better local optimum. The first step consists of constructing a large set of local optima relative to a set of basic moves. The set of local optima is constructed by generating random initial solutions and applying a pure local optimizer (based on the predefined basic moves). A training set of instances belonging to the class of problems under consideration is used for this purpose. For brand new practical problems, for which the availability of data may be limited, a training set could be constructed by generating instances with similar characteristics as the ones observed in
practice. In fact, Section 4.5.3 shows that a randomly generated training set can be used effectively to capture characteristics of the real world instances, suggesting that as long as the main characteristics are preserved, the information on the problem structure can be effectively learned from artificially created instances.

The second step consists of building the \textit{paired-data set} that will be the input to the data mining learning procedure. For each local optimum \(A\), we find all “close” local optima \(B\) to construct pairs of the form: \(\{(A, B) \mid B \in C(A), A \in \mathcal{L}, B \in \mathcal{L}, A \neq B\}\)

where \(\mathcal{L}\) is the local optima set and \(C(A)\) denotes the set of local optima close to \(A\). Proximity between the two solutions is calculated by an appropriate distance measure (e.g., Euclidean) and “closeness” is determined with a threshold. Furthermore, we only keep those local optima \(A\) that have at least a predefined number of close local optima \(B\). Without this policy, the learning procedure would reduce to individuating the local optima that are “easiest” to improve in few moves, rather than individuating escape directions. Section 4.4 shows how this is implemented in the context of the CTAP. A pair \((A, B)\) is called an \textit{improving pair} if the value of the objective function improves when moving from \(A\) to \(B\) and a \textit{non-improving pair} otherwise. Let \(\mathcal{I}\) be the set of pairs of local optima corresponding to improving pairs and \(\mathcal{N}\) the set of pairs of local optima corresponding to non-improving pairs, then:

\[
\mathcal{I} = \{(A, B) : B \in C(A); A \in \mathcal{L}; B \in \mathcal{L}; f(A) > f(B)\}
\]

\[
\mathcal{N} = \{(A, B) : B \in C(A); A \in \mathcal{L}; B \in \mathcal{L}; f(A) \leq f(B)\}
\]

where \(f(x)\) denotes the objective function value of solution \(x\). Note that we are assuming a minimization problem and therefore the improving direction decreases the value of the objective function.
Our methodology requires that the representation of each pair of local optima be formed by $\hat{A}$ (a vector representing the attributes for $A$) and $\Delta_{AB}$ (a vector, which we will refer to as the \textit{local optimum difference}, representing the changes to apply in order to move from $A$ to $B$). Note that the intention is for this representation to be independent of the size of the problem. That is, the number of attributes remains the same regardless of the size of the specific problem instance. As typically done in data mining, the pairs translate into a single vector of numerical values, which is the concatenation of $\hat{A}$ and $\Delta_{AB}$. As expected, the attributes chosen to represent a solution are problem-dependent. Therefore, a procedure that creates $\hat{A}$ from $A$ and $\Delta_{AB}$ from $A$ and $B$ is necessary to apply the method that we are proposing. This procedure should be carefully designed, keeping in mind that the combination of $\hat{A}$ and $\Delta_{AB}$ should enable the learning process. For instance, attributes of $\Delta_{AB}$ that contain information about the objective function may result in trivial rules, such as one that states that “in order to find an improving pair of local optima, the search directions should be restricted to those that eventually will improve the objective function.” Section 4.4 provides an example of selecting attributes in the context of the constrained task allocation problem. Once the paired-data set is produced, the learning process starts. The following definitions facilitate the description of the learning procedure:

- \textbf{Triggering Condition:} A Boolean function $F$ that, given an $\hat{A}$ vector representing a local optimum $A$, produces an output such that $F(\hat{A}) = 1$ indicates that $A$ satisfies condition $F$. We refer to a triggering condition just as “condition” when the context is clear.
• **Guiding Constraint**: A Boolean function $G$ that, given a $\Delta_{AB}$ vector representing a local optimum difference between $A$ and $B$, produces an output such that $G(\Delta_{AB}) = 1$ indicates that the difference between the two local optima satisfies the guiding constraint $G$.

• **Guiding Rule**: A pair $(F, G)$ composed of a triggering condition $F$ and a guiding constraint $G$ such that if $F(\Delta) = 1$ then there exists at least one improving pair $(A, B)$ such that $G(\Delta_{AB}) = 1$ and there does not exist a non-improving pair $(A, B)$ such that $G(\Delta_{AB}) = 1$.

The methodology is based on the assumption that guiding rules that are learned from the training set can be generalized to new instances. In other words, if a local optimum $A$ of a new problem instance satisfies condition $F$, then there probably exists a better local optimum $B$ — i.e. $(A, B)$ is an improving pair — such that $\Delta_{AB}$ satisfies the guiding constraint $G$. This indicates that the guiding constraint $G$ must be enforced every time the search encounters a local optimum satisfying condition $F$, as described in Section 4.3. In the remaining, we refer to coverage of rule $r$ as the proportion of local optima $A$ in the paired-data set that satisfy the rule.

The learning process can be formulated as a mathematical programming problem, whose output is a set of guiding rules of the form $\{(F_r, G_r), r = 1, \ldots, R\}$, where $R$ is a given value. However, before formally introducing the mathematical model, it is useful to describe briefly the basics of the binary classification applied in our method. Binary classification is a well-known data mining technique that has been broadly studied in the literature because of its numerous practical applications (Witten and Frank 2005). It consists of discriminating between two
(positive and negative) classes of objects of the same domain. For instance, in medical applications, patients (i.e. objects) may be classified as sick (positive) or healthy (negative), where each patient is represented by a vector of real numbers. A binary classifier is an “entity” that performs the binary classification through training and then prediction. During training, the binary classifier analyzes a set of objects (training set) where the class of each object is known and “learns” the difference between the two classes. The output of the training procedure is a classification rule. During the prediction procedure, the binary classifier uses this classification rule to predict the unknown class of a new object.

In our method, every Boolean function $F$ and $G$ is associated with a binary classifier — thus we have $2R$ binary classifiers in total. For each rule $r$, the classifier associated with $F_r$, denoted by $C_r^F$, discriminates between local optima satisfying $F_r$ (positive class) and not satisfying $F_r$ (negative class). Similarly, for each rule $r$, the classifier associated with $G_r$, denoted by $C_r^G$, distinguishes between local-optimum differences (i.e., $\Delta_{AB}$) satisfying $G_r$ (positive class) and not satisfying $G_r$ (negative class). Contrary to traditional binary classification, where in the training phase the class of each object is given, here the class of each object (local optimum or local optimum difference) has to be assigned. In other words, for each classifier $C_r^F$ (or $C_r^G$) we seek a partition of local optima (or local optimum differences) into two classes, such that the classification rule found by $C_r^F$ (or $C_r^G$) classifies them in the correct class. These classifiers are then used during a neighborhood search procedure that verifies whether a local optimum satisfies a condition in order to enforce the appropriate guiding constraint.

We use the hyperplane-based classifiers defined by Glover (1990), which are well known in the operations research community. He proposed the use of a hyperplane, identified by a vector $\vec{x}$ and a scalar $b$ (also called parameters of the classifier), to discriminate between two
classes of objects, in such a way that the objects belonging to the positive class lie on the positive side of the hyperplane and the ones belonging to the negative class lie on the negative side of the hyperplane. An object $\mathbf{p}$ lies on the positive side if $\mathbf{x} \cdot \mathbf{p} > b$ and on the negative side if $\mathbf{x} \cdot \mathbf{p} < b$. Usually the mathematical model forbids points to lie too close to the hyperplane by introducing a separation zone of width $2\epsilon$, as in Better et al. (2010). The $C^f$ classifiers classify local optima; therefore the length of the associated $\mathbf{x}$ vectors is equal to the number of attributes needed to represent a local optimum ($L^f$). The $C^g$ classifiers classify local-optimum differences; therefore the length of the associated $\mathbf{x}$ vectors is equal to the number of attributes needed to represent a local optimum difference ($L^g$).

The mathematical formulation employs the following sets of variables:

$$f(A, r) = \begin{cases} 1 & \text{if } A \text{ satisfies } F_r \text{ for } A \in L \text{ and } r = 1, \ldots, R \\ 0 & \text{otherwise} \end{cases}$$

$$g(A, B, r) = \begin{cases} 1 & \text{if } \Delta_{AB} \text{ satisfies } G_r \text{ for } A \in L; B \in C(A); r = 1, \ldots, R \\ 0 & \text{otherwise} \end{cases}$$

$$p(A, B) = \begin{cases} 1 & \text{if } \Delta_{AB} \text{ satisfies at least one } G_r \forall (A, B) \in \mathcal{Z} \\ 0 & \text{otherwise} \end{cases}$$

$$(\mathbf{x}_{F_r}, b_{F_r}) = \text{coefficients associated with the } C^f_r \text{ classifier, } r = 1, \ldots, R$$

$$(\mathbf{x}_{G_r}, b_{G_r}) = \text{coefficients associated with the } C^g_r \text{ classifier, } r = 1, \ldots, R$$
The mathematical model is:

\[
\text{maximize } \sum_{(A,B) \in \mathcal{S}} p(A, B)
\]

subject to

\[
\sum_{r=1}^{R} g(A, B, r) \geq f(A, r) \quad A \in \mathcal{L}; \ r = 1, \ldots, R
\]

\[
g(A, B, r) \leq 1 - f(A, r) \quad A \in \mathcal{L}; \ B: (A, B) \in \mathcal{S} \quad r = 1, \ldots, R
\]

\[
g(A, B, r) \leq f(A, r) \quad A \in \mathcal{L}; \ B: (A, B) \in \mathcal{S} \quad r = 1, \ldots, R
\]

\[
\sum_{r=1}^{R} g(A, B, r) \geq p(A, B) \quad \forall (A, B) \in \mathcal{S}
\]

\[
\bar{A} \cdot \bar{x}_{Fr} + M \cdot (1 - f(A, r)) \geq b_{Fr} + \epsilon \quad A \in \mathcal{L}; \ r = 1, \ldots, R
\]

\[
\bar{A} \cdot \bar{x}_{Fr} - M \cdot f(A, r) \leq b_{Fr} - \epsilon \quad A \in \mathcal{L}; \ r = 1, \ldots, R
\]

\[
\bar{A}_{AB} \cdot \bar{x}_{Gr} + M \cdot (1 - g(A, B, r)) \geq b_{Gr} + \epsilon \quad A \in \mathcal{L}; \ B \in \mathcal{C}(i); \ r = 1, \ldots, R
\]

\[
\bar{A}_{AB} \cdot \bar{x}_{Gr} - M \cdot g(A, B, r) \leq b_{Gr} - \epsilon \quad A \in \mathcal{L}; \ B \in \mathcal{C}(i); \ r = 1, \ldots, R
\]

\[
f(A, r) \in \{0, 1\} \quad A \in \mathcal{L}; \ r = 1, \ldots, R
\]

\[
g(A, B, r) \in \{0, 1\} \quad A \in \mathcal{L}; \ B \in \mathcal{C}(i); \ r = 1, \ldots, R
\]

\[
p(A, B) \leq 1 \quad \forall (A, B) \in \mathcal{S}
\]

\[-1 \leq x_{Fr}(j) \leq 1 \quad r = 1, \ldots, R \quad \forall j = 1, \ldots, l^f\]

\[-1 \leq x_{Gr}(j) \leq 1 \quad r = 1, \ldots, R \quad \forall j = 1, \ldots, l^g\]

\[-1 \leq b_{Fr} \leq 1 \quad r = 1, \ldots, R
\]

\[-1 \leq b_{Gr} \leq 1 \quad r = 1, \ldots, R
\]

The objective function is to maximize the number of improving pairs that satisfy a guiding constraint. When an improving pair \((A, B)\) satisfies a guiding constraint, a way to improve \(A\) has
been found, and therefore we would like to maximize the number of times that \( AB \)-pairs satisfy guiding constraints. In the formulation, (7) and (8) enforce the guiding rule definition, i.e., if \( A \) satisfies condition \( F_r \), then there must be at least one improving pair \( (A,B) \) and no non-improving pair \( (A,B) \) satisfying guiding constraint \( G_r \). Constraint set (9) strengthens the guiding rule by forcing \( A \) to satisfy condition \( F_r \) if there is an improving pair \( (A,B) \) that satisfies the guiding constraint \( G_r \). If an improving pair \( (A,B) \) satisfies at least one guiding constraint \( G_r \), then constraint set (10) allows this pair to be counted toward the objective function value.

Constraints (11) and (12) force \( f(A,r) = 1 \) if and only if \( \tilde{A} \) lies on the positive side of the hyperplane defined by \((\tilde{x}_F, b_F)\). Constraints (7) and (8) force \( g(A,B,r) = 1 \) if and only if \( \tilde{\Delta}_{AB} \) lies on the positive side of the hyperplane \((\tilde{x}_G, b_G)\). In these constraints, \( M \) is a conveniently large positive number, necessary to satisfy the inequalities corresponding to the side of the hyperplane where \( \tilde{A} \) (or \( \tilde{\Delta}_{AB} \)) does not lie. We set \( \epsilon \) equal to 0.01, to force some minimum separation from the classification point and the hyperplane. The model ends with the set of integrality restrictions and bounds for the hyperplane coefficients, which we bound between -1 and 1. Note that changing these bounds would just scale up or down all hyperplane coefficients by the same factor, leading to the same classification rules. The value of \( M \) would need to be changed accordingly, in order to keep constraints (11) to (14) valid.

The classification rule embedded in the hyperplane-based classifiers is an inequality involving a linear combination of the attributes of local optima or their differences. Depending on the problem context, the attributes may be expressed in more than one unit of measure or dimension. For instance, in section 4.4, we apply our methodology to a problem in which we represent local optima using attributes that are measures in three different units. A linear combination involving heterogeneous dimensions (units) is meaningless and the classification
accuracy suffers because the classifier is not “expressive” enough. For instance, a triggering condition in a particular context may involve the logical “and” of a condition that consists of costs and a condition that consists of capacities. In this case, there may not be a single linear combination capable of capturing this complex pattern.

We overcome this by expanding the definition of the $C_f$ classifiers in our model. To this end, let $D$ be the number of the dimensions used in the vector representation of the local optima and $\mathcal{A}^d$ be the sub-vector consisting of the attributes of $A$ belonging to dimension $d$ ($d = 1, \ldots, D$). Every classifier $C_f$ must be defined by a set of hyperplanes, each representing a linear combination of attributes that are measured in the same units. Therefore, we replace the hyperplane-definition variables $(\mathcal{A}^d, b^d)$ with the following:

$$(\mathcal{A}^d, b^d) = \text{coefficients associated with the } d^{\text{th}} \text{ dimension of the } C_f \text{ classifier}$$

The classification rules are changed accordingly, by adding a dimension-specific qualifier to the set of $f$-variables. In particular, we define $f^d(A, r)$ to indicate whether $\mathcal{A}^d$ lies or not on the positive side of the $(\mathcal{A}^d, b^d)$ hyperplane. The sets of constraints (11) and (12) must be changed as follows:

\begin{align*}
\mathcal{A}^d \cdot \mathcal{x}^d_{F_r} + M \cdot (1 - f^d(A, r)) & \geq b^d_{F_r} + \varepsilon \quad A \in \mathcal{L}; \ r = 1, \ldots, R; \ d = 1, \ldots, D \quad (11') \\
\mathcal{A}^d \cdot \mathcal{x}^d_{F_r} - M \cdot f^d(A, r) & \leq b^d_{F_r} - \varepsilon \quad A \in \mathcal{L}; \ r = 1, \ldots, R; \ d = 1, \ldots, D \quad (12')
\end{align*}

And two new constraint sets must be added along with the integrality constraints for the $f^d$ variables:
\[ f^d(A, r) \geq f(A, r) \quad A \in \mathcal{L}; \ r = 1, \ldots, R; \ d = 1, \ldots, D \]  
(15)

\[ \sum_{d=1}^{D} f^d(A, r) - D + 1 \leq f(A, r) \quad A \in \mathcal{L}; \ r = 1, \ldots, R \]  
(16)

\[ f^d(A, r) \in \{0, 1\} \quad A \in \mathcal{L}; \ r = 1, \ldots, R; \ d = 1, \ldots, D \]

Constraints (15) and (16) ensure that \( f(A, r) = 1 \) if and only if \( f^d(A, r) = 1 \) for all dimensions \( d = 1, \ldots, D \).

Instead of using the output of the model as “pure classifiers”, we use it for ranking. That is, for prediction purposes, the output of the classifiers is not simply the predicted class but rather a score which is “a numeric value that represents the degree to which an instance is a member of a class” (Fawcett 2006). Therefore instead of simply checking the value of \( F_r(\tilde{A}) \) for a particular local optimum, we calculate its score as follows. For each of the \( D \) hyperplanes associated with \( C'_r \), we compute the relative Euclidean distance between the hyperplane and \( \tilde{A} \), which is equal to a positive Euclidean distance if \( \tilde{A} \) lies in the positive side of the hyperplane, or to a negative Euclidean distance otherwise. The minimum among the \( D \) relative distances is the score obtained by applying the classification rule \( F_r \) to \( \tilde{A} \). We often refer to it as “the Euclidean distance between \( C'_r \) and \( A \)”. If \( \tilde{A} \) lies on the negative side of at least one hyperplane, the score is negative and \( F_r(\tilde{A}) = 0 \); otherwise it is positive and \( F_r(\tilde{A}) = 1 \). In either case, this distance measures how close \( \tilde{A} \) is to the decision boundary. We do the same for a given difference \( \tilde{\Delta}_{AB} \) and instead of simply checking the value of \( G_r(\tilde{\Delta}_{AB}) \), we calculate the Euclidean distance between \( C'_r \) and \( \tilde{\Delta}_{AB} \), where this distance will be strictly positive when \( G_r(\tilde{\Delta}_{AB}) = 1 \).

There are two reasons for ranking instead of applying “purely binary” classifiers. First, when a local optimum \( A \) is encountered during the search, we must retrieve the triggering
condition that is satisfied the most, which is the one that maximizes the distance between \( C_r^f \) and \( A \). Second, our method attempts to satisfy as much as possible the guiding constraint corresponding to that triggering condition, i.e. it attempts to maximize the distance between \( C_r^g \) and \( \Delta_{AB} \).

4.3. Escaping Local Optimality

The output of the learning procedure is a set of \( R \) guiding rules of the form \( \{(F_r, G_r), r = 1, \ldots, R\} \). Considering those local optima in the training set that are close to a given local optimum \( L \) and that satisfy \( F_r \), then there exists at least one local optimum \( M \) that is better than \( L \) and for which \( G_r(L, M) = 1 \) and at the same time there does not exist a local optimum \( M \) that is worse than \( L \) for which \( G_r(L, M) = 1 \). Assuming that this also holds for local optima not in the training set, we want to design a procedure to escape local optimality in such a way that the search identifies directions to move from the current local optimum \( L \) to a better solution \( M \). The escape procedure takes as input a local optimum \( L \) and attempts to find an escape direction that will move the search to a better solution \( M \) by enforcing the guiding constraint corresponding to the triggering condition that is satisfied by the largest margin.

The escape is performed through a sequence of tabu search steps that are limited to reaching solutions \( S \) such that the \((L, S)\) pair satisfies \( G_r \). The tabu search procedure is limited to a short-term memory structure with \( TabuTenure \) drawn from a uniform distribution between \( minTenure \) and \( maxTenure \) every time a move is performed. In the remainder of the article, we use \( TS_n \) to denote a tabu search process that stops after \( nonImprove \) consecutive non-improving moves. We also use \( TS_m \) to denote a tabu search process that stops either after \( maxSteps \) moves or immediately after reaching a solution better than \( L \). In both cases, the short
term memory is managed by the *minTenure* and *maxTenure* parameters. Figure 13 shows the complete outline of the escape procedure.

In step 1, the procedure identifies the triggering condition that is satisfied the most by calculating the Euclidean distance between the local optimum point *L* and all the conditions *C_r^f*. If no triggering condition is satisfied, then the one that is “closest” to being satisfied is chosen. Let this condition be *C_r^f*. Step 2 initializes a *TS_m*. The objective function in this part of the process is set to maximizing the Euclidean distance between the condition *C_r^f* and Δ_{LS}, where *L* is the local optimum from which the procedure is trying to escape and *S* is the current solution. This objective function — referred to as the *distance objective function* to distinguish it from the original *cost objective function* — has the goal of directing the search toward a point *S* for which \( G_r(Δ_{LS}) = 1 \). The process starts with no additional constraints imposed upon the neighborhood of the current solution.

**Figure 13:** Outline of the escape procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Calculate the Euclidean distance between the local optimum point <em>L</em> and all the conditions <em>C_r^f</em> in order to identify the most satisfied condition <em>C_r^f</em>.</td>
</tr>
<tr>
<td>2.</td>
<td>Initialize a tabu search <em>TS_m</em> with the distance objective function, termination criterion <em>maxSteps</em> and tabu tenure parameters <em>minTenure</em> and <em>maxTenure</em></td>
</tr>
<tr>
<td>3.</td>
<td>Perform tabu search steps until any of these criteria is satisfied:</td>
</tr>
<tr>
<td></td>
<td>a. If a solution <em>S</em> better (according to the cost objective function) than <em>L</em> is reached, the escape has been successful and <em>S</em> is returned</td>
</tr>
<tr>
<td></td>
<td>b. If a solution <em>S</em> is reached such that Euclidean distance between <em>C_r^f</em> and Δ_{LS} is strictly greater than zero then go to step 4</td>
</tr>
<tr>
<td></td>
<td>c. If no improving move according to the distance objective function is available or <em>maxSteps</em> iterations have been performed then the procedure terminates with an unsuccessful escape</td>
</tr>
<tr>
<td>4.</td>
<td>Change to the cost objective function and restrict moves to those for which the Euclidean distance between <em>C_r^f</em> and Δ_{LS} is strictly positive, where <em>S</em> is the current solution</td>
</tr>
<tr>
<td>5.</td>
<td>Continue the tabu search process until a solution <em>S</em> better than <em>L</em> is reached (successful escape) or <em>maxSteps</em> iterations have been performed (unsuccessful escape)</td>
</tr>
</tbody>
</table>
Tabu search iterations are performed in step 3, where the most improving move (according to the distance objective function) is chosen in each iteration. After the move is performed, conditions 3a to 3c are checked in the order shown. First, if a solution $S$ that is better than $L$ is reached, the escape is successful and $S$ is returned. Second, if a solution $S$ such that $G_r(\Delta_{LS}) = 1$ is found, then the search has reached the goal of satisfying the guiding constraint and therefore there is no need to keep searching using the distance objective function and the process moves to step 4. Third, if all solutions in the neighborhood lead to a deterioration of the distance objective function or the number of iterations has reached the maximum allowed ($maxSteps$) then the procedure terminates with an unsuccessful escape.

Steps 4 and 5 are executed if the guiding constraint is satisfied (criterion 3b). In this case, we continue the search in the limited space of solutions that satisfy the guiding constraint, in an attempt to find a solution that is better than $L$. Therefore, we set the constraint that the Euclidean distance between $C'_\nu$ and $\Delta_{LS}$ must be strictly positive for the search to move to solution $S$. The merit of a move is calculated using the cost objective function. The current number of iterations of the tabu search procedure is not reset to zero and the tabu search memory structure retains its current information. If a better solution is reached then the escape is successful and $S$ is returned. Otherwise, the search stops after a total of $maxSteps$ iterations have been performed. In summary, the escape procedure is a modified version of $TS_m$, where the search is driven not only by the tabu restrictions, but also by the most appropriate guiding constraint. Section 4.5.1 compares these two procedures.

We have designed three neighborhood-based procedures (i.e., local search, tabu search and escape) that use different strategies to select the next move to make and to determine when to stop. The basic neighborhood of the three procedures is the same, as defined by a common set.
of move mechanisms. However, some procedures modify the neighborhood and/or the move evaluation in order to navigate the solution space:

- **Local search** — The neighborhood includes all solutions that can be reached by the move mechanisms defined for the problem context. It uses the original (cost driven) objective function to evaluate moves and it stops when no improving move is available in the neighborhood of the current solution. It returns a local optimum point with respect to the entire neighborhood.

- **Tabu search** — The neighborhood includes only those solutions that can be reached by performing non-tabu moves from the current solution. The tabu status of a move may be overridden if the neighborhood contains a solution that is better than the incumbent. Two stopping criteria are used, one for $TS_n$ and one for $TS_m$, as specified above.

- **Escape procedure** — The neighborhood may be modified to include only those solutions that meet a guiding constraint. The procedure uses tabu search for which the objective function is switched from one that is based on Euclidean distance to the original based on cost. The procedure may fail to produce a path to a solution that is better than the local optimum from which it is trying to escape.

We combine these procedures into a single search organized as shown in Figure 14. We will refer to this procedure as a data mining driven tabu search ($DMDTS$) because of the data mining techniques employed to create the set of guiding rules used in the escape procedure.
As shown in Figure 14, \( DMDTS \) requires an initial local optimum point \( L \) (which could be chosen from the set generated for the offline learning process or generated by more sophisticated methods), the values for the tabu search parameters (\( maxSteps, minTenure \) and \( maxTenure \)) and a set of guiding rules. If at least one of the conditions is satisfied at the current local optimum, the escape procedure is invoked. Otherwise, the simple tabu search procedure is used to escape, allowing a maximum number of iterations. Step 2 attempts to improve upon the solution \( S \) found in step 1 if this solution is better than \( L \). The process stops when step 1 fails to improve upon the current local optimum \( L \), which is also the best solution found during the call to \( DMDTS \).

### 4.4. Constrained Task Allocation Problem

In order to test the merit of our methodology, we have chosen a difficult constrained combinatorial optimization problem. The Constrained Task Allocation Problem (CTAP), defined by Hadj-Alouane et al. (1999), consists of finding the feasible assignment of \( m \) tasks to \( n \) CPUs that minimizes the total assignment cost. An assignment is feasible if the capacity of each CPU is greater than or equal to the sum of the capacity required by the tasks assigned to it. There are two types of costs: fixed cost, paid if a CPU is used, and communication cost, paid for every pair of tasks assigned to different CPUs. The mathematical model is:
Minimize
\[ \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} c_{ij} \left( 1 - \sum_{k=1}^{n} x_{ik} x_{jk} \right) + \sum_{k=1}^{n} s_k y_k \]

Subject to
\[ \sum_{k=1}^{n} x_{ik} = 1 \quad i = 1, \ldots, m \]
\[ \sum_{i=1}^{m} a_i x_{ik} \leq b_k y_k \quad k = 1, \ldots, n \]
\[ x_{ik} \leq y_k \quad i = 1, \ldots, m; k = 1, \ldots, n \]
\[ x_{ik} \in \{0,1\}, y_k \in \{0,1\} \quad i = 1, \ldots, m; k = 1, \ldots, n \]

Where the notation corresponds to that used by Ernst, Jiang, and Krishnamoorthy (2006):

- \( x_{ik} \): binary decision variable that equals 1 if task \( i \) is assigned to CPU \( k \)
- \( y_k \): binary decision variable that equals 1 if CPU \( k \) is used
- \( c_{ij} \): communication cost between tasks \( i \) and \( j \)
- \( s_k \): fixed cost of using CPU \( k \)
- \( a_i \): capacity needed by task \( i \)
- \( b_k \): capacity of CPU \( k \)

The data set that we used is the same as the one in Experiment 2 of Lusa and Potts (2008). The set consists of 108 randomly generated instances of sizes ranging between 20 and 100 tasks and 5 and 30 processors, for a total of 12 different sizes. For each problem size, 3 “loose”, 3 “medium” and 3 “tight” instances were generated, resulting in \( 3 \times 12 = 36 \) different
combinations of size and “tightness” (where tightness refers to the relation between the capacity required by the tasks and the available capacity provided by the processors). Further details about these instances may be found in Lusa and Potts (2008). To have representation from each of the 36 combinations, we selected the first instance of each type to create a training set with a total of 36 instances. The remaining 72 instances were put aside as the test set.

We first define a solution representation and a set of basic moves. Solutions are represented by a vector $\pi$ of size $m$ consisting of the assignments of the tasks to processors. That is $\pi(i)$ contains the index of the processor to which task $i$ is assigned. Suppose that in the current solution $\pi(i) = k$ and $\pi(j) = l$. We define two simple moves:

1. The value of $\pi(i)$ is changed from $k$ to $l$, indicating that task $i$, currently assigned to processor $k$, will be now assigned to processor $l$

2. The value of $\pi(i)$ is changed from $k$ to $l$ and the value of $\pi(j)$ is changed from $l$ to $k$, indicating that tasks $i$ and $j$ will switch processors

With this solution representation and basic moves, we developed a local search and a simple tabu search. The local optimizer searches for the move that minimizes the assignment cost and executes the move if it improves the objective function value of the current solution. The process stops when no improving move is available in the neighborhood of the current solution. The simple tabu search adds to the local optimizer the possibility of executing non-improving moves. After the execution of a type 1 move, the attribute $(i, k)$ is recorded to prevent task $i$ from being assigned to processor $k$ in the next TabuTenure iterations. When a type-2 move is executed, the attribute pair $(j, l)$ is also recorded to prevent moves that will assign task $j$ to
processor $k$ in the next $TabuTenure$ iterations. The tabu-status of a move is overridden if the move results in a solution that is better than the incumbent. In order to construct the paired-data set, we generate local optima by executing the following procedure 1,000 times for each problem instance in the training set:

1. Generate a solution by randomly sorting tasks and processors, and by assigning each task to the first processor found with sufficient remaining capacity. If no such processor is found, the task is assigned to the one that yields the minimum capacity violation.

2. If the solution obtained is infeasible, then run the simple tabu search by switching the objective to minimizing the maximum capacity violation. As soon as a feasible solution is found, go to step 3. If no feasible solution is found in 1,000 iterations, then go to step 1.

3. Run the local search procedure by always selecting the feasible move that reduces the assignment cost the most.

For step 2, we choose the value of $TabuTenure$ from a discrete uniform distribution defined on the interval $[20, 200]$. In tabu search implementations, this is usually a function of the size of the problem, as shown in the experimental section. However, we observed that a feasible solution is always found in a few iterations. This suggests that the value of the parameters of the tabu tenure interval is not critical for the purpose of finding a feasible solution. We also observed that for small problem instances (e.g., those with only 5 CPUs) the local search performed at step 3
converged to the same solution more than once. For these instances, the number of local optima collected was less than 1,000.

We now must define a measure of distance, set a “proximity” threshold \( p \), and set a minimum number of “close” local optima \( t \). The distance \( d(A,B) \) between two local optimal solutions \( A \) and \( B \) is simply the number of assignment differences (i.e. Hamming distance between the solution representations). Therefore, the maximum distance between two solutions occurs when no task is assigned to the same processor in both solutions, resulting in a total distance \( d(A,B) = m \). We consider that two solutions are “close” to each other if:

\[
\frac{d(A,B)}{m} \leq 0.15
\]

Furthermore, we consider only the local optimal points with at least 5 local optima within their proximity. The choice of these two values is found through a simple sensitivity analysis, which is reported in Appendix B.

The next step consists of formulating the attributes used to represent local optima. This is where knowledge about the problem context is exercised. We based these attributes on the definition of two “appealing” moves (M1 and M2) from a local optimal solution \( A \) and the identification of seven processors (CPU1 to CPU7) in this solution. M1 and M2 are appealing because they have the potential of improving the value of the objective function if some constraints were relaxed or some costs ignored.

M1. The best feasible reassignment of task \( i \) from its current processor \( k \) to processor \( l \) if the largest communication cost between \( i \) and any other task currently assigned to \( k \) is ignored
M2. Same as M1 but allowing for a capacity violation in $l$ that is no more than half of the capacity requirements of task $i$

CPU1. Processor $k$ in M1
CPU2. Processor $l$ in M1
CPU3. Processor $k$ in M2
CPU4. Processor $l$ in M2
CPU5. The processor currently in use with the fewest tasks assigned to it
CPU6. The processor not currently in use with the minimum fixed cost
CPU7. The processor not currently in use with the largest total capacity

Note that the definitions above are independent of the size of the instance under consideration.

Using these definitions, we characterize $A$ with a vector $\tilde{A}$ consisting of 30 attributes:

- For each CPU$i$ ($i = 1, \ldots, 7$), create attributes for total capacity, available capacity, fixed cost, the percentage of tasks assigned to it ($4 \times 7 = 28$ attributes)
- The change in the objective function if move M1 were executed (1 attribute)
- The change in the objective function if move M2 were executed (1 attribute)

We then create 35 attributes to characterize $\tilde{A}_{AB}$:

- For each CPU$i$ ($i = 1, \ldots, 5$) in use and for each CPU$j$ ($j = 1, \ldots, 7$, $j \neq i$) different from CPU$i$, make the attribute value equal to 1 if there is at least one task that is assigned to CPU$i$ in $A$ and to CPU$j$ in $B$, and 0 otherwise ($5 \times 6 = 30$ attributes)
- For each CPU$i$ ($i = 1, \ldots, 5$) in use make the attribute value equal to 1 if there is at least one task that is assigned to CPU$i$ in $A$ and to processor $k$ in $B$, such that $k$ is unused in $A$, and 0 otherwise ($5$ attributes)
Three dimensions are used in $\tilde{A}$ to characterize $A$: capacity ($d = 1$), percentage of tasks ($d = 2$), and cost ($d = 3$), as given by the change in the objective function. Therefore, the $C^f_r$ classifiers use 3 hyperplanes, allowing a greater expressivity than using a single hyperplane classifier. Conversely, only one dimension is used in the attributes associated with $\bar{A}_{AB}$: the attributes are 1 if there is a subset of tasks moving from a given CPU to another and 0 otherwise.

The following is an example of a guiding rule that can be expressed by our attribute representation:

“If the total capacity of CPU6 in $A$ is larger than or equal to the difference between the total capacity of CPU5 and the available capacity of CPU5 (condition on dimension 1)

AND

the fixed cost of CPU6 is less than the fixed cost of CPU5 (condition on dimension 3)

THEN

move some tasks from CPU5 to CPU6 (guiding constraint)”

This classification $C^f_r$ rule is a logical AND of two relatively simple classification rules (linear combinations), one that considers only capacity attributes and one that considers only cost attributes (as shown below). Note that it is impossible to formulate such a rule using only one linear combination that involves all attributes. Interestingly, if this condition is verified by $A$, the objective function will certainly improve by moving all the tasks in CPU5 to CPU6, making this rule highly reliable.

One possible representation of this rule in terms of the mathematical formulation is as follows. Let $(\tilde{x}^d_{F_1}, b^d_{F_1})$ be the coefficients of the hyperplanes of the classifier $C^f_1$ (the only $C^f$ classifier in
this example) relative to dimension $d = 1, 2, 3$. The coefficients of $\tilde{x}_{F_1}^1$ are all equal to 0 except the ones corresponding to:

- the total capacity of CPU5, for which the value of the coefficient is equal to $-1$
- the total capacity of CPU6, for which the value of the coefficient is equal to $1$
- the available capacity of CPU5, for which the value of the coefficient is $1$

In addition, the hyperplane has $b_{F_1}^1$ equal to 0. Then, a solution $A$ satisfies the condition on dimension 1 if and only if it satisfies $\tilde{A}^1 \cdot \tilde{x}_{F_1}^1 \geq b_{F_1}^1$. The guiding rule above does not involve dimension 2, therefore all components of $\tilde{x}_{F_1}^2$ are equal to 0 and $b_{F_1}^2$ is equal to $-1$, so that any solution $A$ always satisfies $\tilde{A}^2 \cdot \tilde{x}_{F_1}^2 \geq b_{F_1}^2$. Finally, the coefficients of $\tilde{x}_{F_1}^3$ are all equal to 0 except the ones corresponding to:

- the fixed cost of CPU5, for which the value of the coefficient is equal to $1$
- the fixed cost of CPU6, for which the value of the coefficient is equal to $-1$

$b_{F_1}^3$ is equal to 0 in this hyperplane. Then, a solution $A$ satisfies the condition on the third dimension if and only if it satisfies $\tilde{A}^3 \cdot \tilde{x}_{F_1}^3 \geq b_{F_1}^3$. Therefore, $F_1(\tilde{A}) = 1$ when the conditions on both dimensions (1 and 3) are satisfied.

The coefficients of $\tilde{x}_{G_1}$ are all equal to 0 except for the one corresponding to the presence of at least one task assigned to CPU5 in $A$ and to CPU6 in $B$. Also, let $b_{G_1}$ be equal to 1. Then, $G_1(\tilde{A}_{AB}) = 1$ if and only if there is at least one task moving from CPU5 to CPU6.
4.5. Computational Experiments on the CTAP

We perform three main experiments to test our ideas on the CTAP. The first experiment assesses the effectiveness of the escape procedure and the other two tests the performance of the *DMDTS*. The search parameters for these experiments are set as follows, where \( m \) is the number of tasks and \( n \) is the number of CPUs.

\[
\begin{align*}
\text{maxSteps} & = m \\
\text{nonImprove} & = m \\
\text{minTenure} & = 0.01nm \\
\text{maxTenure} & = 0.1nm
\end{align*}
\]

The tabu search is set up as described in section 4.4. That is, there are two types of moves and a short term memory controlled by a tabu tenure that is drawn from a uniform distribution between \( \text{minTenure} \) and \( \text{maxTenure} \).

4.5.1. Experiment 1

For this experiment, we consider the subset of 36 problem instances used to build the paired-data set in section 4.4. The experiment consists of 30 executions of the steps outlined in Figure 15. We use two versions of the escape procedure, one for which the learning stage is limited to one guiding rule (i.e., \( R = 1 \)) and one for which \( R = 2 \). For the purpose of reporting results, we denote these versions as *EP* \(-1\) and *EP* \(-2\), respectively. We are interested in measuring the success rate of the two versions of the escape procedure. Success is defined as in Figure 13, that is, when the procedure is able to move from the initial local optimum \( L \) to another one of a better quality. We also track the relative improvement achieved by successful escapes. Table 17 reports the results in terms of the success rate and the improvement for *EP* \(-1\), *EP* \(-2\) and
The results associated with the escape procedures are divided into the “satisfying” and “not satisfying” sets.

**Figure 15:** Process to test the effectiveness of the escape procedure

1. Learn the guiding rules from a subset of the paired-data set involving a random sample of 30% of all local optima (training set)
2. The remaining local optima (test set) are partitioned into two subsets labeled *satisfying* and *not satisfying*, indicating whether there exists at least one guiding rule $r$ such that $F_r(L) = 1$
3. From each local optimum $L$ in the test set
   a. execute $TS_m$ followed by a local search after a successful escape
   b. execute the escape procedure followed by a local search after a successful escape.

We used CPLEX 11.0 to find exact solutions to the learning model. Solving the learning model required an average of 16 seconds for $EP - 1$ and 7.2 minutes for $EP - 2$ on an Intel Xeon 3.20 GHz machine with 2 GB of RAM and running Windows Server 2003 Enterprise Edition. Different random seeds were used in each of the 30 runs to obtain the sample of local optimal solutions and therefore build the training and test sets. The random seed also affected the sequence of tabu tenure values chosen for both the escape and the tabu search procedures.

The results confirm that the escape procedure is more effective when the triggering condition of the guiding rule is satisfied, as indicated by the high average success rate (81.58% for $EP - 1$ and 68.38% for $EP - 2$). While the expectation is that the escape procedure will not perform well when applied to local optima for which no guiding rule applies, we observed two cases (tests 11 and 27 for $EP - 2$) where the success rate associated with the “not satisfying” subset was higher than the rate for the “satisfying” group. We attribute this to the low ratio
between the cardinality of the training set and the one of the test set. The partitions corresponding to tests 11 and 27 resulted in rules that were optimal for the training set but lead to inferior escape directions in the test set. Unlike the success rate, the improvement does not dramatically decrease if the triggering condition is not satisfied. This is in agreement with the design of our experiment, since the same guiding constraint is used for both the “satisfying” and the “non satisfying” group.

The results in Table 17 also show that $EP - 1$ performs at a higher level than $EP - 2$. Additional experiments show that the performance of both $EP - 1$ and $EP - 2$ increases with the size of the training set relative to the test set. Nevertheless, the performance of $EP - 2$ is generally inferior. This result is counterintuitive given the flexibility added by embedding more than one guiding rule in the search. The proportion of local optima that satisfy at least one guiding rule increases from 23.92% for $EP - 1$ to 34.8% for $EP - 2$, however, the success rate drops. We attribute the inferior performance of $EP - 2$ to the higher complexity of the rules when compared to the complexity of the single rule used in $EP - 1$. We have observed that the neighborhood coverage of the single rule in $EP - 1$ is often greater than the individual coverage provided by the 2 rules used in $EP - 2$. This may be caused by the objective function of the learning model, which maximizes the total coverage as opposed to the coverage of individual rules. The result is that the 2 rules in $EP - 2$ tend to be more complex and more specific to the observed local optima instead of simple and general. Therefore, they fail to adhere to an important concept of classification, the Occam’s razor (Domingos 1999), which states that simple classification rules are preferred to complex ones.
Table 17: Summary of results of Experiment 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Satisfying (23.92%)</td>
<td>Not Satisfying (76.08%)</td>
</tr>
<tr>
<td>1</td>
<td>73.47%</td>
<td>29.78%</td>
</tr>
<tr>
<td>2</td>
<td>73.68%</td>
<td>32.45%</td>
</tr>
<tr>
<td>3</td>
<td>73.81%</td>
<td>22.15%</td>
</tr>
<tr>
<td>4</td>
<td>100.00%</td>
<td>25.99%</td>
</tr>
<tr>
<td>5</td>
<td>98.04%</td>
<td>27.05%</td>
</tr>
<tr>
<td>6</td>
<td>82.05%</td>
<td>25.77%</td>
</tr>
<tr>
<td>7</td>
<td>82.69%</td>
<td>24.34%</td>
</tr>
<tr>
<td>8</td>
<td>83.33%</td>
<td>20.92%</td>
</tr>
<tr>
<td>9</td>
<td>85.71%</td>
<td>22.82%</td>
</tr>
<tr>
<td>10</td>
<td>97.50%</td>
<td>30.73%</td>
</tr>
<tr>
<td>11</td>
<td>72.00%</td>
<td>30.38%</td>
</tr>
<tr>
<td>12</td>
<td>81.40%</td>
<td>26.61%</td>
</tr>
<tr>
<td>13</td>
<td>83.72%</td>
<td>23.95%</td>
</tr>
<tr>
<td>14</td>
<td>88.37%</td>
<td>27.92%</td>
</tr>
<tr>
<td>15</td>
<td>62.50%</td>
<td>20.39%</td>
</tr>
<tr>
<td>16</td>
<td>87.76%</td>
<td>24.11%</td>
</tr>
<tr>
<td>17</td>
<td>84.85%</td>
<td>31.15%</td>
</tr>
<tr>
<td>18</td>
<td>64.00%</td>
<td>27.19%</td>
</tr>
<tr>
<td>19</td>
<td>74.14%</td>
<td>21.22%</td>
</tr>
<tr>
<td>20</td>
<td>78.72%</td>
<td>24.11%</td>
</tr>
<tr>
<td>21</td>
<td>100.00%</td>
<td>28.48%</td>
</tr>
<tr>
<td>22</td>
<td>75.68%</td>
<td>17.75%</td>
</tr>
<tr>
<td>23</td>
<td>79.59%</td>
<td>23.53%</td>
</tr>
<tr>
<td>24</td>
<td>85.71%</td>
<td>27.49%</td>
</tr>
<tr>
<td>25</td>
<td>71.79%</td>
<td>19.78%</td>
</tr>
<tr>
<td>26</td>
<td>90.00%</td>
<td>21.87%</td>
</tr>
<tr>
<td>27</td>
<td>97.73%</td>
<td>26.01%</td>
</tr>
<tr>
<td>28</td>
<td>66.67%</td>
<td>12.45%</td>
</tr>
<tr>
<td>29</td>
<td>79.17%</td>
<td>19.09%</td>
</tr>
<tr>
<td>30</td>
<td>73.17%</td>
<td>24.16%</td>
</tr>
<tr>
<td>Avg</td>
<td>81.58%</td>
<td>24.65%</td>
</tr>
</tbody>
</table>

The escape procedures outperform $TS_m$, indicating that the learning process is able to provide effective directions to move away from local optima. The simple tabu search rules are
able to find a path out of the basin of attraction of the local optimal point (with an overall success rate of 62.35%) but the improvement is significantly smaller than the one achieved by the escape procedures. In a related experiment, we used randomly generated guiding rules (that is, we generated random coefficients for the hyperplanes instead of solving the learning model) to test whether learning was actually helping the procedure. When using random guiding rules, the average success rate of $EP - 1$ dropped to 48.75% with a corresponding average improvement of 14.09%. Interestingly, in this “random rules” scenario the average improvement is also greater than the one obtained by $TS_m$. This may be explained by the attributes used to define the local-optimum differences. These attributes represent groups of tasks that move from one CPU to another. Therefore, applying a guiding rule (even one randomly constructed) often leads to a change of the set of CPUs used. The objective function value of the solution reached after the move will change considerably due to the change in fixed costs, which determine the largest portion of the total cost. Then, the average improvement, which is computed only among those cases where the solution quality improves, ends up being relatively large.

The main and the secondary experiments show that the learning model is capable of identifying effective escape directions and the conditions to apply them. They also show that the escape procedure is able to use this knowledge effectively to reach better solutions. From a data mining standpoint, the relatively small percentage of the available data that was used for training (30% of the local optima) did not lead to under-fitting, suggesting that the learning process that we propose does not require an impractical amount of effort involving extremely large training sets.
4.5.2. Experiment 2

In this second set of experiments, we assess the performance obtained by embedding the escape strategy into a complete search. In particular, we focus on testing the performance of the data mining driven tabu search procedure described in section 4.3. Since this procedure requires a local optimum as one of its inputs, we developed a multi-start process (i.e., construction and improvement) based on the GRASP methodology. GRASP constructions results in local optima that on the average are of higher quality than those obtained by a totally random process, such as the one we used for the learning process. The goal during the learning stage was to construct a data set with a sample of local optima that were not necessarily concentrated around the high quality solutions. The goal now is to study the effectiveness of the resulting search procedure when applied to local optima that are constructed with a method that is not completely random and that balances both solution quality and diversity. The details of the GRASP process are included in Appendix A. Each GRASP iteration results in a local optimum that then is used as the input to either DMDTS or TS$_n$. Following the results obtained in the first set of experiments, we limit the escape procedure within DMDTS to employing a single guiding rule. The values of the search parameters are set as stated at the beginning of section 4.5.

The test set consists of 72 of the 108 instances generated by Lusa and Potts (2008). These are the instances that were not used for the learning process and the set contains 24 in each of three categories: loose, medium and tight. We execute 50 GRASP iterations, resulting in 50 different local optima. Then, DMDTS and TS$_n$ are applied to the resulting local optimum. We record, for each problem type at the end of a GRASP iteration, the number of DMDTS and TS$_n$ “wins”, where a win for a method is defined as finding a solution that is better than the competing method. We then subtract the number of TS$_n$ wins from the DMDTS wins to find the
DMDTS score (which ranges from -72 to +72). The DMDTS scores are graphically shown in Figure 16, where L, M and T are used to indicate the DMDTS scores for loose, medium and tight instances, respectively. The TOT area shows the total DMDTS score.

![Figure 16: DMDTS scores for 50 GRASP iterations](image)

We perform a one-tailed paired $t$-test for each instance in the test set with the goal of assessing if the value of the solution found by DMDTS across the 50 iterations is smaller (better) than the one found with $TS_n$. With $\alpha = 0.05$, the test concludes that in 28 cases DMDTS performs significantly better than $TS_n$, in 6 cases significantly worse, and in 38 cases the difference is not significant. The results in Figure 16 show that, when considering all instances, DMDTS outperforms $TS_n$ after 50 GRASP iterations. DMDTS has more consistently positive results for loose and medium instances than for tight instances. This is due to the limited number (7) of CPUs involved in the attribute representation used for the local-optimum differences. For the large tight instances, where 20 or more CPUs are used, this attribute representation is often insufficient, and the “task shuffling” produced by a simple tabu search proves to be more
effective in this case. It would be of great interest to analyze the effect of using a different learning procedure for each instance type, but this is outside the scope of this study. A change in the attribute representation would also lead to a change in performance. The horizon of 50 GRASP iterations was chosen to approximate the computational effort employed by procedures that have appeared in the literature for the CTAP and with which we compare DMDTS below. However, we have verified that for extremely long computational times the advantage of DMDTS versus TS\textsubscript{n} tends to disappear. In other words, as the number of GRASP iterations increase, the lines in Figure 16 converge to zero. In this case, the advantage of DMDTS on TS\textsubscript{n} starts to vanish after around 100 iterations. This simply shows that in long searches, the sampling effect of the GRASP constructions overshadows the efficiency gains achieved by the learning process.

There are two prominent heuristic procedures for the CTAP in the literature, the hybrid tabu search (HTS) method developed by Chen and Lin (2000) and the variable neighborhood search (VNS) proposed by Lusa and Potts (2008). We use the “win, tie or lose” criterion to compare the results of DMDTS versus these established methods. The results are summarized in Table 18. The values in this table indicate the percentage (in the set of 72 instances) of DMDTS wins, ties and loses against the alternative approaches. The DMDTS results are those obtained at the end of the 50 GRASP iterations in the previous experiment and they are compared to the solutions reported in the literature.

With respect to the average time required to obtain the results used to compute the values shown in Table 18, the fastest method is HTS, while VNS and DMDTS achieve similar performance: VNS is 2.64 times slower than HTS and DMDTS is 2.19 times slower than HTS. These relative comparisons of computational effort are accurate in the case of HTS and VNS,
because Lusa and Potts (2008) ran these procedures on the same computer equipment. \textit{DMDTS} was executed on a different computer (specified above), but accounting for the difference in equipment, we have determined that, given the same amount of computer time, \textit{DMDTS} and VNS examine approximately the same number of points in the solution space.

<table>
<thead>
<tr>
<th>\textit{DMDTS} vs.</th>
<th>Win</th>
<th>Tie</th>
<th>Lose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTS by Chen and Lin (2000)</td>
<td>66.67%</td>
<td>13.89%</td>
<td>19.44%</td>
</tr>
<tr>
<td>VNS by Lusa and Potts (2008)</td>
<td>18.06%</td>
<td>27.78%</td>
<td>54.17%</td>
</tr>
</tbody>
</table>

The results shown in Table 18 indicate that \textit{DMDTS} is able to outperform (on the test problems used for this experiment) an existing method based on tabu search (HTS). This is an encouraging result because it shows that a hybrid of tabu search obtained by performing the offline learning procedure seems to have an advantage over one that is based on design choices made by the developers of the procedure. The limitations of \textit{DMDTS} are shown when comparing its outcomes to the results obtained by the VNS procedure. As stated by Lusa and Potts (2008), this method is specialized to the CTAP and employs the following 5 neighborhoods to search the solution space:

1. Reallocate a task \(i\) from processor \(k\) to processor \(l\)
2. Exchange two tasks (task \(i\) from processor \(k\) to processor \(l\) and task \(j\) from processor \(l\) to processor \(k\))
3. Reallocate a cluster of tasks from processor \(k\) to processor \(l\)
4. Reallocate a cluster of tasks from different processors to processor \(l\)
5. Empty processor \(k\)
It is remarkable, however, that with only moves 1 and 2 from the list above, \textit{DMDTS} is capable of finding 13 new best-known solutions to the set of 72 problems. Table 19 reports the objective function values of the best solutions found by Lusa and Potts (2008) and the new benchmarks that we obtained with \textit{DMDTS}. The problem instances are identified using Lusa and Potts notation, i.e., the first 3 digits correspond to the number of tasks, followed by 2 digits identifying the number of CPUs, one digit to indicate a loose (1), medium (2), or tight (3) instance, and 3 digits for the instance number within the set (010, 050 or 100).

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Problem & Lusa and Potts (2008) & \textit{DMDTS} \\
\hline
060_10_1_050 & 120,100 & 120,057 \\
080_05_1_050 & 185,653 & 185,601 \\
080_05_1_100 & 352,343 & 352,291 \\
080_05_2_050 & 76,037 & 75,967 \\
080_05_2_100 & 139,072 & 139,002 \\
080_05_3_050 & 273,240 & 273,170 \\
080_05_3_100 & 524,203 & 524,019 \\
080_10_1_050 & 107,651 & 107,637 \\
080_10_1_100 & 181,704 & 181,677 \\
100_10_3_050 & 213,252 & 213,183 \\
100_20_2_050 & 112,153 & 112,094 \\
100_30_2_050 & 109,523 & 109,513 \\
100_30_2_100 & 164,733 & 164,726 \\
\hline
\end{tabular}
\caption{New best-solutions for 13 CTAP problem instances.}
\end{table}

As shown in Table 18, Lusa and Potts (2008) obtain solution of better quality than those found by \textit{DMDTS} on 39 instances out of 72 (54.17%). On average, these solutions are 0.89\% better than \textit{DMDTS} with a maximum of 5.33\%. 

116
4.5.3. Experiment 3

In our last experiment, we applied DMDTS to eight additional CTAP instances from the literature. Hadj-Aloune et al. (1999) introduced these problems and Ernst, Jiang, and Krishnamoorthy (2006) modified one and used the resulting set of nine instances for testing their exact procedure, concluding that:

“Our experiments indicated that the CTAP is a much harder problem than the UTAP [uncapacitated task allocation problem] mainly because of the capacity constraints. Neither integer programming formulations nor the column generation formulation for the CTAP performed well for the test problems.”

The same set of problems was used by Lusa and Potts (2008). They also coded and applied Chen and Lin (2000) hybrid tabu search to these problems. As Lusa and Potts explain, the problems were originated at “an automobile microcomputer system and a Hughes air-defense system.” Both Ernst, Jiang and Krishnamoorthy (2006) and Lusa and Potts (2008) provide a detailed description of the characteristics of these problem instances, which have been labeled A to H. (Ernst, Jiang and Krishnamoorthy (2006) added a ninth problem to the set, which they labeled H’, by ignoring the preallocation of eight tasks in problem H.) Table 20 of Ernst, Jiang and Krishnamoorthy (2006) reports the best solutions found by Hadj-Aloune et al. (1999) and the best solutions found by CTAP2t, which is the integer programming formulation that worked best for these problems. Table 17 in Lusa and Potts (2008) reports these results and adds performance information associated with two versions of their VNS procedure and the hybrid tabu search of Chen and Lin (2000). The information includes simple statistics (minimum, average and maximum) on the objective function values found in 50 runs performed on each
problem instance. We also perform 50 runs of our \textit{DMDTS} procedure and compare our outcomes with previous results in Table 20. We don’t include computational times, because we have already established the relative standing regarding computational effort of our method.

The Lusa and Potts (2008) results in Table 20 correspond to their GVNS2 procedure, which is reportedly their best procedure. This table and a number of additional results shown by Lusa and Potts (2008) support their conclusion that their VNS procedure outperforms the previous state-of-the-art for CTAP (namely, Chen and Lin’s hybrid tabu search). Our \textit{DMDTS} is competitive and is able to find one new best solution (for problem F) but we can’t claim that it outperforms Lusa and Potts’ VNS implementation because we fail to match the best-known solutions to problems A and D.

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\hline
A (20, 6) & 13,450 & 13,804 & 13,519 & 13,450 & 13,866 \\
B (20, 6) & 11,946 & 11,946 & 11,946 & 11,946 & 11,946 \\
C (20, 6) & 11,120 & 11,120 & 11,156 & 11,126 & 11,120 \\
D (40, 12) & 39,738 & 39,680 & 41,457 & 39,214 & 39,690 \\
E (40, 12) & 38,602 & 36,575 & 37,731 & 35,671 & 35,671 \\
F (40, 12) & 35,016 & 35,821 & 36,410 & 34,674 & 34,624 \\
G (15, 5) & 16 & 16 & N/A & 16 & 16 \\
H (41, 2) & 40 & N/A & 40 & 40 & 40 \\
\hline
\end{tabular}
\caption{Comparison of five solution procedures on eight real-world CTAP instances.}
\end{table}

Best solutions are shown in bold.

Nevertheless, this shows that the learning model could correctly characterize some improving directions that other ad-hoc methods do not consider; it also shows that the \textit{DMDTS} is able to effectively enforce these directions. From a learning point of view, it is particularly encouraging to observe that some valuable information about the structure of this class of
problems has been learned from random instances and then applied successfully to real world problems.

4.6. Application to the Matrix Bandwidth Minimization Problem

In order to show that the $DMDTS$ may be applied to other optimization problems, we consider the Matrix Bandwidth Minimization Problem (MBMP), which consists of finding the permutation of rows and columns that minimizes the bandwidth of a matrix. This problem is equivalent to labeling the vertices of an undirected graph so that the maximum difference between the labels of any pair of adjacent vertices is minimized. The most effective approaches to the MBMP use neighborhood search procedures (Martí et al. 2001, Martí et al. 2008, Piñana et al. 2004), where the elementary move is the “swap” between the labels of two vertices. The maximum difference between the label of a vertex $v$ and the one of the adjacent vertices is called “bandwidth of $v$” and denoted by $B(v)$. A vertex $v$ is critical if $B(v)$ is equal to the objective function. If a solution contains more than 2 critical vertices, it is possible that no swap leads to a decrease in the objective function. In fact, there may be several swaps that do not change the objective function value. Therefore, if such a solution is encountered during the search, a procedure based on these moves may perform many swaps that do not change the objective function value of the current solution and spend considerable computational time without improving the solution quality, until some stopping criterion is met. In other words, a procedure that relies solely on computing changes of the objective function value caused by swap will likely be ineffective to escape local optimality. This problem is overcome by establishing search direction with calculations that are different from changes on the objective function value. This characteristic makes our proposed methodology particularly suitable for the MBMP.
As previously established, our methodology requires an attribute representation for solutions and solution pairs. To apply it to the MBMP, first we group the vertices according to their bandwidth, and then we describe these groups through attributes. Let $w$ be the objective function value of a solution. Then, we build $G$ “bandwidth groups”, each formed by the vertices whose bandwidth is included in a given range. The range of the first group is $[w - \alpha w, w)$, where $\alpha$ is a design parameter; the range of the second group is $[w - 2\alpha w, w - \alpha w)$, and so on.

A solution is characterized by two attributes for each bandwidth group: 1) the percentage of vertices belonging to the group and 2) their average degree. We add another attribute that is equal to the bandwidth of the vertex with the highest degree (in case of ties, the vertex whose neighbors have the highest average degree).

The attributes used to describe a pair of solutions $(A, B)$ are defined by the bandwidth groups in $A$. For each group, we add an attribute equal to +1 if the number of vertices in the group increased from $A$ to $B$, 0 if it did not change, or -1 if it decreased; we add an attribute equal to +1 if their average degree increased from $A$ to $B$, 0 if it did not change, -1 if it decreased. Finally, the last attribute is equal to +1 if the bandwidth of the vertex with the highest degree increased from $A$ to $B$, 0 if it did not change, -1 if it decreased.

As in the case of the CTAP, we compared the performance obtained by the simple tabu search described in section 4.5 to the one obtained by $DMDTS$. Relatively to the 126 instances considered by Martí et al. (2001), $DMDTS$ finds a better solution than the one found by the simple tabu search in 107 cases, a worse solution in 17 cases, and a solution with the same value in 2 cases. In average, the solution found by $DMDTS$ is 28.50\% better than the one found by the simple tabu search. Although these results show the advantage of using our methodology to enhance a simple tabu search, the value of the solution found is generally far from the best
known solution values. In order to obtain more competitive results, we applied our methodology to the method developed by Martí et al. (2001), who propose a tabu search algorithm where the quality of a swap is not measured by the objective function, but is inversely proportional to the number of vertices that become critical or “near critical”. We considered the procedure that they refer to as $TS(200)$, which executes a tabu search with a limit of 200 non improving steps starting from a random solution, without applying a re-starting mechanism or any other long term memory structure. We overlaid our method on $TS(200)$ by using a guiding constraint to evaluate the swaps in order to escape local optimality (escape phase). Once the guiding constraint is satisfied, we use their evaluation function with the restriction that the guiding constraint be satisfied (exploration phase). Once a better solution is reached (successful escape), the swaps are again evaluated using the function proposed by Martí et al. (2001), until a new local optimum is reached. Maximum number of non-improving steps, tabu tenure, and all the other search parameters are set as in the original algorithm.

Among the 126 instances considered by Martí et al. (2001), we selected those for which the best solution obtained by running $TS(200)$ from 50 different randomly chosen starting solutions has a value that is over 30% from the best known solution value. This was done to identify the instances (11 in this case) in which the performance of $TS(200)$ could be improved. In all other instances the procedure is capable of matching or nearly matching the best known solutions, making it almost impossible to assess the merit of overlaying our methodology. Nonetheless, we applied $DMCTS$ to the remaining 115 instances to verify that there is no performance loss compared to the original $TS(200)$.

First, we selected 30 instances (training set) from the 115 instances and reserved the 11 “difficult” instances as our test set. Second, we built the paired-data set by forming a pair $(A, B)$
whenever the expected relative distance between $A$ and $B$ was less than or equal to 0.02. The expected relative distance between two solutions is the ratio between their distance and the expected distance. The distance between two solutions is computed as the average absolute difference between the labels assigned to the same vertex. To define the expected distance between two solutions of an instance of size $n$, we note that a vertex that is labeled $i$ in solution $A$ may be labeled $1, 2, \ldots, n$ in solution $B$. Therefore, the absolute difference between the labels of $i$ is uniformly distributed in $(i - 1, i - 2, \ldots, 0, 1, 2, \ldots, n - i)$. So, the expected absolute difference between the labels of $i$ is:

$$\frac{1}{n} \left( \sum_{j=1}^{i-1} j + \sum_{j=1}^{n-i} j \right)$$

The expected distance is the average across all vertices:

$$\frac{1}{n^2} \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{i-1} j + \sum_{j=1}^{n-i} j \right) \right]$$

The attributes are built using 8 bandwidth groups and $\alpha = 0.1$. These parameters were tuned by analyzing the performance obtained on the training set. The time limit for the learning procedure is 20 minutes.

For each test instance, a random solution was generated and both $TS(200)$ and $DMDTS$ were executed from this solution. After the search was performed for all 11 instances, we recorded the number of wins of $DMDTS$ over $TS(200)$ and the score of $DMDTS$, as defined in Section 4.5. This procedure was repeated 50 times, each time using a different seed to generate the initial solutions. Figure 17 shows the score for each repetition.
After 50 repetitions, \textit{DMDTS} wins 7 times (average deviation 11.6\%), loses once (deviation -12.5\%), and ties 3 times over \textit{TS}(200), leading to a score equal to 6. Interestingly, the pattern depicted in Figure 17 is very similar to the one in Figure 16, corresponding to the CTAP. In both cases, the superiority of \textit{DMDTS} is evident only after a few repetitions. As we did for the CTAP, we performed a one-tailed paired \textit{t}-test for each instance in the test set with the goal of assessing if the value of the solution found by \textit{DMDTS} across the 50 iterations is smaller (better) than the one found by \textit{TS}(200). With $\alpha = 0.05$, the test determined that \textit{DMDTS} performs significantly better than \textit{TS}(200) on 7 instances, while the difference is not significant in 4 instances. Further analysis of the data suggests that, sometimes, the guiding constraint leads the \textit{DMDTS} to a low quality solution. For some instances this may happen for all the initial repetitions, thus reducing the \textit{DMDTS} score. At some point though, the guiding constraint leads to a high quality solution, better than any solution found by tabu search up to then. At this point the \textit{DMDTS} score starts increasing. In terms of computational time, \textit{DMDTS} takes about 3 times longer than the simple tabu search, mainly because of the extra time spent computing the attributes at local optima. Nevertheless, one should compare the time spent to overlay \textit{DMDTS}
on $TS(200)$, which consists of designing the attributes and collecting local optima data against the time spent to modify $TS(200)$ in order to improve its performance. We argue that this comparison favors $DMDTS$.

4.7. Conclusions

Existing approaches based on neighborhood search may be viewed as consisting of a local search and a set of predefined rules (e.g., tabu activation) that impose constraints (e.g., tabu restrictions) that are enforced during the search in order to allow the process to escape local optimality. We propose a method that learns the rules (learning procedure) from a training set of instances of a problem class and applies them (escape procedure) to new instances. Our method is general and we show its application to the constrained task allocation problem. The escape procedure is also one instance of a set of possible ways in which the rules that result from the learning procedure may be used to constrain a neighborhood search. The results that we obtained for the CTAP are encouraging. First, we showed that the “enhanced” tabu search outperforms the version that does not take advantage of the learning process. Then we showed that it also outperforms a tabu search that is hybrid in nature (Chen and Lin, 2000), where the hybridization is the result of a predefined noising strategy that modifies the search directions by perturbing key problem data. This is equivalent to adding “soft” constraints with the purpose of finding new search directions out of local optimal points. The difference is that the directions are not the result of rules that were learned but of design choices (e.g., the parameters to perturb and the amount of the perturbation).

Although our method requires the design of problem-specific attributes, the tradeoff is that the effort required to implement the search procedure is minimized. In other words, our
framework focuses on the characterization of the problem (e.g., by the definition of attributes) rather than on the design of specialized solution strategies. While what we propose may not always be the faster approach, our experiments show that it can achieve better results. Furthermore, our approach may be overlaid on existing procedures based on neighborhood search in order to improve performance, as we did for the MBMP.

Our proposal involves some offline effort because the learning procedure is based on a mathematical formulation that is solved exactly with commercial software. This opens the door to interesting opportunities for future research, for instance, one consisting of the development of efficient exact or heuristic procedures to accelerate the learning process. The need for such a method depends on the context because typically the learning process would be performed once for a given class of problems. A more challenging problem perhaps will be to design on-line learning procedures that use a portion of the computational budget to learn rules and apply them to the problem instance being solved. Furthermore, computational studies on the impact obtained by varying the training set or the effort spent on the learning procedure would certainly be of interest.

Other research directions point to the application of our concepts to more sophisticated neighborhood searches, such as those based on VNS, or to search techniques that are based on evolving a population of solutions via combination mechanisms. In the former, the learning must consider a paired-data set originated from points that are local optima relative to all VNS moves. In the latter, the rules could identify situations for applying certain combination methods over others. We envision an alternative paradigm for the design of metaheuristic procedures, where data mining techniques are used not as a second level enhancement (e.g. for parameter tuning), but as a main tool to design the rules used to search the solution space.
5. FUTURE OPPORTUNITIES IN ANALYTICS

This dissertation has presented different methods to integrate analytics into the decision making process. The process exemplified by the Appointment Scheduling paper presented in Section 3 can be applied anywhere data is available, but its application has especially great opportunities in health care. The expanding use of health information technology, especially as the American Reinvestment and Recovery Act provides incentives for providers to make meaningful use of electronic health records, leads to growing numbers of clinics developing more complete and useful databases of patient and service information. This provides fertile ground to identify meaningful performance-predicting attributes to improve schedule performance and to further develop our data mining approaches.

A particularly attractive area is the use of data mining to analyze the output of optimization procedures, with the goal of finding a simple operating rule. For example, Samorani and Ganguly (2012) propose an efficient method based on stochastic optimization to help doctors decide which patient to see next – this problem is particularly relevant whenever a patient arrives early. However, clinics may be unwilling to adopt such a complex procedure to make this decision; they would prefer to have a simple operating rule that can be easily understood and implemented. To this end, a decision tree can be built from the set of optimal decisions made during a simulation procedure. The branches and nodes of the tree represent an operating rule expressed in terms of “if-then-else” embedded statements, which can be readily adopted by clinics. Very little work has been done in this promising area of research. Besides obtaining a heuristic procedure, this process has the potential of leading to new managerial insights in the problem at hand.
In conclusion, the use of analytics to improve decision making is in its early stages. While most of the decisions made in organizations are currently based on intuition (Davenport and Harris 2007), the premises for analytics-based decision making are present, both from a technological and from a “data availability” point of view. Health care represents the most suitable field where analytics can be successfully applied in practice.
REFERENCES


LaGanga, L.R., S.R. Lawrence. 2007b. Clinic overbooking to improve patient access and increase provider productivity. Decision Sciences, 38(2), 251-276.


APPENDIX

Appendix A: Extended Example of Comparison Refinements

The attribute considered in this example is a strong indicator of mutagenicity. The procedure that created it is a simple way to represent:

Label each atom different from carbon with the number of atoms of nitrogen to which it is connected. Compute the mean of these labels.

Consider the path in Figure 18. The elements are denoted by a number to distinguish multiple appearances of the same element (e.g., Atom and AtomBond appear twice). The derived attributes are included in boxes and linked to the elements to which they belong. Table 21 reports, for each step, the values of currentEle, \( A_g \), \( A_r \), the aggregating function and the possible refinement chosen.

<table>
<thead>
<tr>
<th>Step</th>
<th>currentEle</th>
<th>Aggr. Function</th>
<th>( A_g )</th>
<th>( A_r )</th>
<th>Ref. Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>-</td>
<td>5.ele</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>MostFrequent</td>
<td>5.ele</td>
<td>5.idAtom</td>
<td>!= 1.idAtom</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-</td>
<td>3.Derived</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>CountDistinct</td>
<td>3.idBond</td>
<td>2.Derived</td>
<td>= N</td>
</tr>
<tr>
<td>5</td>
<td>Target</td>
<td>Avg</td>
<td>1.Derived</td>
<td>1.ele</td>
<td>!= C</td>
</tr>
</tbody>
</table>

This is a detailed description of the steps in Table 21 and the description of the attribute virtually attached to each element by the Roll-Up algorithm:

1. currentEle is 4. Choose \( A_g \) equal to “5.ele”. This results in attaching 5.ele to element 4.
2. currentEle is 3. Choose $A_g$ equal to “5.ele” with MostFrequent as aggregating function. Choose $A_r$ equal to “5.idAtom” and the Comparison refinement “not equal to 1.idAtom”. “1.idAtom” is the atom “on the other side” of the bond. Note that since every bond is connected to exactly 2 atoms, the “most frequent element on the other side” is just the “element on the other side”.

3. currentEle is 2. Choose $A_g$ equal to the attribute derived at step 2 and attach it to element 2.

4. currentEle is 1. Choose $A_g$ equal to “3.idBond” with CountDistinct as the aggregating function. Choose $A_r$ equal to the attribute derived at step 2 and the value refinement “equal to N” (i.e., equal to nitrogen). For each row in element 2 (i.e. for each atom), we derived the number of bonds to which it is connected that have a nitrogen on the other side.

5. currentEle is “Target”. Choose $A_g$ equal to the attribute derived at the previous step with Avg as the aggregating function. Choose $A_r$ equal to “1.ele” and the value refinement “not equal to C” (i.e., different from carbon). The final attribute is then obtained.
Figure 18. Graphical representation of extended example

Avg
  (  
    Count distinct (3.idBond)  
    Where  
      (  
        Most Frequent (5.ele)  
        Where 5.idAtom != 1.idAtom  
      ) = N  
    )  
  )  
Where 1.ele != C

For each atom different from carbon, the number of atoms of nitrogen to which it is connected

Count distinct (3.idBond)  
Where  
(  
  Most Frequent (5.ele)  
  Where 5.idAtom != 1.idAtom  
) = N

For each atom, the number of atoms of nitrogen to which it is connected
Appendix B: Online Procedure

The Online procedure consists of solving the problem described in the last paragraph of Section 3.4 for each \( s = 1, \ldots, S, \) \( d = 0, \ldots, h - 1, \) and \( j = 0, \ldots, K. \) The input is composed by:

- The existing schedule of day \( d. \) For each constrained visit request \( v, \) we know its predicted revenue \( r(v), \) waiting time cost category \( c(v), \) and predicted showing outcome \( s(v). \) In this section, we treat \( q \) like any other constrained visit request.

- For each available visit request \( v \in \mathcal{F}^s(d), \) we know its predicted revenue \( r(v), \) waiting time cost category \( c(v), \) and predicted showing outcome \( s(v). \)

- \( \alpha(s), \gamma(i, d, s) \)

The problem consists in finding the optimal assignment of the visit requests in \( \mathcal{F}^s(d) \) to the open slots of day \( d. \) The waiting time costs, overtime cost, and revenues, are modified by the dual variables \( \alpha(s) \) and \( \gamma(i, d, s), \) as follows:

- \( w_c := \alpha(s)w_c \)
- \( \tau := \alpha(s)\tau \)
- \( r(i) := \alpha(s)r(i) + \gamma(i, d, s) \)

This problem is solved by using the data structure built during the offline procedure, composed of the non-dominated complete sn-sequences for each day \( d = 0, \ldots, h - 1, \) for each slot assigned to \( q \) \( j = 0, \ldots, K, \) for each feasible total number of shows \( i. \) From the optimality conditions mentioned in section 3.4, it follows that the optimal assignment results in one of these non-dominated sequences. Therefore, for each non-dominated sequence \( e, \) we find the optimal assignment of the visit requests in \( \mathcal{F}^s(d) \) to the open slot of day \( d, \) such that sequence \( e \) is obtained. The scheduling with the highest profit is finally returned. The problem can be divided into two sub-problems: finding the optimal assignment of the available no-shows to the 'n' slots
of $e$ and finding the optimal assignment of the available shows to the ‘$s$’ slots of $e$. The assignment of the no-shows is trivially solved by selecting the highest profit no-shows from $\mathcal{F}^S(d)$ and assigning them to the ‘$n$’ slots in an arbitrary fashion. The assignment of the shows is a harder problem, because there is a trade-off to consider. On one hand, it is desirable to use the shows belonging to the least expensive waiting time cost categories; on the other hand, it is desirable to use those with the highest revenues.

The algorithm that we propose to solve the assignment of the shows is better explained from a graphical point of view. We can view any assignment as a table $A$, which has one row for each ‘$s$’ in $e$ and one column for each category. We sort the columns by waiting time cost (decreasing from left to right) and the rows by the waiting time experienced by the ‘$s$’ associated with the row (increasing from top to bottom). The waiting time costs used here are not the original ones: they need to be divided by the number of showing patients, in order for the average waiting time cost to be correctly computed. In the remainder of this section, we simply refer to waiting time costs to indicate these adjusted waiting time costs. We use notation $A(i) = w_k$ to indicate that the ‘$s$’ corresponding to the $i$-th row from the top is assigned to an appointment request of category $w_k$. From now on, let us consider the following example: $D = 30$, $T = 20$, $e = sssnssnssn$ (no appointment request is constrained), 4 categories with waiting time costs $w_1 > w_2 > w_3 > w_4$. The sequence includes 7 shows, which experience the following waiting times: 0, 10, 20, 0, 10, 0, 10. Figure 19a shows the assignment table corresponding to a possible optimal solution.
In this solution, two appointment requests belonging to the first category are assigned to 2 of the 3 slots that have no waiting time, one appointment request of the second category is assigned to the remaining slot with no waiting time and another one is assigned to a slot with 10 minutes of waiting time. Two appointment requests of the third category are assigned to two slots with 10 minutes of waiting time. Finally, an appointment request of the fourth category is assigned to the slot with 20 minutes of waiting time. Note that, for each category, the appointment requests used are the ones with the highest revenues within that category. Since the revenues of the appointment requests are known once we know how many appointment requests of each category are selected, the real goal is to minimize the waiting time cost. Recall that the overtime cost is fixed, given the sequence $e$. Therefore, we treat the problem as a minimization problem.

We intend to prove that the optimal solution has the diagonal-like structure of Figure 19a. If it does not, then there is another optimal solution that does. Therefore, we can focus our search only on the tables with this structure. In other words, we can exclude from our search all the solutions that look like the one in Figure 19b. The reason is that we obtain a non-worse solution by changing the assignment as indicated by the arrows. Changing $A(7)$ from $w_2$ to $w_4$ reduces
the cost by $20 \cdot (w_2 - w_4)$, because the patient that waits 20 minutes switches to a less expensive category; similarly, changing $A(6)$ from $w_4$ to $w_2$ increases the cost by $10 \cdot (w_2 - w_4)$. Therefore, the objective function value decreases. The formal proof is the following:

**Necessary condition for optimality:** Let $A^*$ be the optimal solution. If $\exists i, j, p, q$ such that $i < j$, $p < q$, $A^*(i) = w_q$ and $A^*(j) = w_p$, then there is an equivalent solution $B^*$ such that and $B^*(i) = w_p$ and $B^*(j) = w_q$

**Proof:** Call $t_x$ the waiting time associated with row $x$. Changing $A^*(i)$ from $w_q$ to $w_p$ would increase the objective function by $t_i(w_p - w_q)$; changing $A^*(j)$ from $w_p$ to $w_q$ would decrease the objective function by $t_j(w_p - w_q)$. The overall change is therefore $(w_p - w_q)(t_i - t_j)$, which is less than or equal to 0 because $t_i < t_j$. □

Therefore, a solution is completely defined by a vector $V$ whose components are the number of appointment requests that are scheduled for each category. For example, if $V$ is $(2, 2, 2, 1)$, then the assignment table is the one depicted in Figure 19a.

We now want to define an algorithm that finds the optimal solution. Our procedure starts by sorting the showing appointment requests by decreasing waiting time cost, and, within the same category, by decreasing revenue. Following this order, the sorted appointment requests are sequentially used to potentially update the solution. We prove that the solution obtained at the end of the procedure is optimal. The first step of the algorithm finds the optimal solution obtained by considering only the first $K$ appointment requests, which is built by inserting the first $K$ appointment requests into the vector $V$. Now we need to answer the following question: given the optimal solution obtained by considering the first $g$ appointment requests (i.e. after the $g$-th
step), what is the optimal solution if we consider the first \( g + 1 \) appointment requests? The following theorem proves that the optimal solution is obtained either by not modifying the current solution or by inserting the \( (g + 1) \)-th appointment request and removing an existing one.

**Theorem:** Suppose that after the \( g \)-th step we have considered all appointment requests of categories \( w_1, \ldots, w_j \), and not all those of category \( w_{j+1} \). Let \( v \) be the next appointment request and let \( V^g(j) \) be the \( j \)-th component of the optimal vector \( V \) at the end of the \( g \)-th step. Then:

\[
\forall g = 1, \ldots, N - 1: \exists V^{g+1} \text{ optimal such that } \forall j = 1, \ldots, \bar{j}: V^{g+1}(j) \leq V^g(j)
\]

**Proof:** Suppose there is no optimal solution \( V^{g+1} \) such that \( \forall j = 1, \ldots, \bar{j}: V^{g+1}(j) \leq V^g(j) \). Therefore, given any optimal solution \( V^{g+1} \), \( \exists j: 1 \leq j \leq \bar{j}, \ V^{g+1}(j) > V^g(j) \). Let \( I^g(j) \) be the row assigned at step \( g \) to the bottom element of category \( j \) (i.e. among the selected elements of category \( j \), the one with the lowest profit). The number of selected elements of category \( j \) has increased from step \( g \) to step \( g + 1 \), but, depending on the variation of the elements of the categories \( j < \bar{j} \), the position of its bottom element might have changed. So, let us consider two cases:

**Case 1:** \( I^{g+1}(\bar{j}) \leq I^g(\bar{j}) \)

In the example in Figure 20, \( w_j = w_{j+1} = w_4 \). At step \( g+1 \), we insert an appointment request of category \( w_5 \) and \( V^{g+1}(4) > V^g(4) \). Furthermore, \( I^{g+1}(4) \leq I^g(4) \).
Since the position of the last element of $\bar{j}$ has not increased, and since we have added at least one element to category $\bar{j}$, there must be some category among $w_1, w_2, ..., w_{\bar{j}-1}$ whose number of elements has decreased. We scan the categories from $j = \bar{j} - 1$ to $j = 1$, until we find a category $\bar{j} < \bar{j}$ such that $V^{g+1}(\bar{j}) < V^g(\bar{j})$. In our example, $\bar{j} = 2$. By construction, $\forall j = \bar{j} + 1, ..., \bar{j} - 1: V^{g+1}(j) \geq V^g(j)$. By combining this last condition with $V^{g+1}(\bar{j}) > V^g(\bar{j})$, we obtain that $\forall j = \bar{j}, ..., \bar{j} - 1: I^{g+1}(j) < I^g(j)$. In our example, the position of the last element of category $w_2$ changed from 5 to 3, and the position of the last element of category $w_3$ changed from 7 to 5.

Now, we want to prove that the solution at step $g + 1$ is not optimal because there exists a better one, obtained by removing an element of $w_j$ and adding an element to $w_j$. Note that this operation is possible because there is at least one available appointment request of $w_j$ that can be used: the one that was inserted in step $g$. Beside inserting an element and removing another, this operation causes all the elements of categories $w_{j+1}, ..., w_{j-1}$ to be shifted one position.
downwards. In our example, this happens to the two elements of category \(w_3\), as depicted in Figure 21.

**Figure 21:** Assignment corresponding to \(V = (3, 1, 0, 3)\)

The change in objective function \(\Delta^{g+1}\) takes into account the insertion of the second highest-revenue element of \(w_2\), the removal of the third highest-revenue element of \(w_4\), and the two shifting moves indicated in Figure 21. Let \(p^j(q)\) be the revenue of the \(q\)-th highest revenue element of \(w_j\). Also, let \(t(i)\) denote the waiting time of row \(i\); therefore, \(t(I^{g+1}(j))\) indicates the waiting time experienced by the bottom element of category \(w_j\) and \(t(I^{g+1}(j) + 1)\) the waiting time experienced by the top element of category \(w_{j+1}\).

\[
\Delta^{g+1} = p^2(2) - p^4(3) - 0 \cdot (w_2 - w_3) - 10 \cdot (w_3 - w_4)
\]

In general:

\[
\Delta^{g+1} = p^j(V^{g+1}(j) + 1) - p^j(V^{g+1}(j)) \quad \sum_{j=j_0}^{j_1} t(I^{g+1}(j) + 1) \cdot (w_j - w_{j+1})
\]

Note that since \(V^{g+1}\) is optimal, \(\Delta^{g+1} \leq 0\).
Now consider $V^g$. By inserting an element of $w_j$ and removing an element of $w_j$ we obtain a solution that is worse than $V^g$, since $V^g$ is optimal. This move is depicted in Figure 22.

**Figure 22:** Assignment corresponding to $V = (3, 1, 0, 3)$

In the example, the change in objective function $\Delta^g$ takes into account the insertion of the third highest-revenue appointment request of waiting time cost $w_4$, the removal of the second highest-revenue element of waiting time cost $w_2$, and the two shifting moves indicated in Figure 22:

$$\Delta^g = p^4(3) - p^2(2) + 10 \cdot (w_2 - w_3) + 10 \cdot (w_3 - w_4)$$

In general:

$$\Delta^g = p^j(V^g(j) + 1) - p^j(V^g(j)) + \sum_{j=j-1}^{j} t(V^g(j)) \cdot (w_j - w_{j+1})$$

Since $V^g$ is optimal, $\Delta^g \leq 0$.

Let us compute $\Delta^g + \Delta^{g+1}$:

$$\Delta^g + \Delta^{g+1} =$$

$$p^j(V^g(j) + 1) - p^j(V^{g+1}(j)) + p^j(V^{g+1}(j) + 1) - p^j(V^g(j))$$

144
\[ + \sum_{j=\bar{j}, \ldots, \bar{j}-1} [t(I^g(j)) - t(I^{g+1}(j) + 1)] \cdot (w_j - w_{j+1}) \]

The first line of the expression above is greater than or equal to 0 because \( V^{g+1}(\bar{j}) > V^g(\bar{j}) \); similarly, the second line is greater than or equal to 0 because \( V^{g+1}(\bar{j}) < V^g(\bar{j}) \); the third line is greater than or equal to 0 because \( \forall j = \bar{j}, \ldots, \bar{j} - 1: I^{g+1}(j) < I^g(j) \). Therefore, \( \Delta^g + \Delta^{g+1} \geq 0 \).

The only way not to contradict the statements \( \Delta^g \leq 0, \Delta^{g+1} \leq 0 \) is to have \( \Delta^g = \Delta^{g+1} = 0 \). Suppose that this is the case and execute the move corresponding to \( \Delta^{g+1} \) and obtain an equivalent optimal solution \( \hat{V}^{g+1} \), whose components are:

\[ \hat{V}^{g+1}(\bar{j}) = V^{g+1}(\bar{j}) - 1 \]
\[ \hat{V}^{g+1}(\bar{j}) = V^{g+1}(\bar{j}) + 1 \]
\[ \hat{V}^{g+1}(j), \forall j \neq \bar{j}, \bar{j} \]

Note that even by adding the element to \( \bar{j} \), \( \hat{V}^{g+1}(\bar{j}) \leq V^g(\bar{j}) \), because \( V^{g+1}(\bar{j}) < V^g(\bar{j}) \); by removing an element of \( \bar{j} \), we may or may not obtain \( \hat{V}^{g+1}(\bar{j}) \leq V^g(\bar{j}) \). If we do, and there is no other category \( j \) such that \( \hat{V}^{g+1}(j) > V^g(j) \), then we found an optimal solution \( \hat{V}^{g+1} \) that contradicts the initial hypothesis because \( \exists \bar{j}: 1 \leq \bar{j} \leq \bar{j}, V^{g+1}(\bar{j}) > V^g(\bar{j}) \), and in this case the proof is concluded. If, on the other hand, there is still some category \( j \) such that \( \hat{V}^{g+1}(j) > V^g(j) \), then we can re-apply all the computations done in this proof to \( \hat{V}^{g+1} \). First, find a category \( \bar{j} \) such that \( \hat{V}^{g+1}(\bar{j}) > V^g(\bar{j}) \); then, depending on which case we are in (case 1 or case 2), find the category \( \bar{j} \); then, we can prove that an equivalent optimal solution is obtained by removing an element from \( \bar{j} \) and adding an element to \( \bar{j} \); finally, we execute this operation and check if the theorem is finally proven, or we continue. The proof will eventually end because, at some point, we will obtain a solution that contradicts the hypothesis. In fact, at each step, the removal of an element of \( \bar{j} \) brings us closer to contradict the hypothesis, because \( V^{g+1}(\bar{j}) \)
decreases. On the other hand, the addition of adding an element of \( j \) will not prevent to contradict the hypothesis. All the other components of \( V^{g+1} \) do not vary, and therefore will not prevent to contradict the hypothesis.

Case 2: \( I^{g+1}(\bar{j}) > I^g(\bar{j}) \)

This case is analogous to Case 1. The main difference is that the category \( \bar{j} \) whose number of elements decreased has to satisfy \( j > \bar{j}. \) Therefore, the computation of the moves \( \Delta^g \) and \( \Delta^{g+1} \) is different, but the procedure is very similar to the one seen for Case 1. ■

The previous theorem is extremely important because it allows us to define a polynomial algorithm to solve the assignment problem of the shows. As already said, the algorithm starts by sorting the appointment requests and by building the optimal solution using the first \( K \) appointment requests. Then, at each step \( g, \) the next appointment request is considered and we try to insert it into the current solution \( V^g. \) The theorem above guarantees that the optimal solution is one obtained by either not including the next appointment request, or by inserting it and removing another one. In the first case, \( V^{g+1}(j) = V^g(j) \forall j = 1, \ldots, \bar{j}; \) in the second case, \( V^{g+1}(j) = V^{g+1}((\bar{j}) + 1, V^{g+1}(j^*) = V^g(j^*) - 1, V^{g+1}(j) = V^g(j) \forall j \neq j^*, k, \) where \( j^* \) is the \( j \) that maximizes the change in the objective function corresponding to inserting an element of \( \bar{j} \) and removing an element of \( j. \)

\[
j^* = \arg\max_{j=1, \ldots, j-1} \left\{ p^j(V^g(j)) + p^j(V^g(j)) + \sum_{j=j, \ldots, j-1} t(I^g(j)) \cdot (w_j - w_{j+1}) \right\}
\]

Initially, a data structure is built that can be used to retrieve the value of \( I^g(j). \) At each step of the algorithm, we use an accumulator to compute \( \sum_{j=j, \ldots, j-1} t(I^g(j)) \cdot (w_j - w_{j+1}). \) We start from \( \bar{j} = j - 1 \) and set the accumulator to \( t(I^g(j - 1)) \cdot (w_{j-1} - w_j). \) This expression is
computed in $O(1)$. Then, from $j = j - 1$ to $j = 1$, the value of the accumulator is increased by $t(I^j(j)) \cdot (w_j - w_{j+1})$ in $O(1)$. So, one step of the algorithm has a complexity equal to $O(c)$, where $c$ is the number of categories. Since the algorithm must execute $n$ steps, its overall complexity is $O(cn)$, where $n$ is the number of appointment request and $c$ the number of waiting time cost categories.
Appendix C: A Greedy Randomized Adaptive Search Procedure for the Constrained Task Allocation Problem

GRASP, or greedy randomized adaptive search procedure, is a multi-start technique, introduced by Feo and Resende (1995), that constructs and improves solutions at each iteration. The technique is often used as a standalone metaheuristic, but we use it to generate high-quality local optima from which we want to test our escape procedures. A solution for the CTAP is constructed by assigning at each step a task to a CPU, where, initially, all the CPUs are empty (i.e., no task has been assigned to any of the processors). The process stops when all tasks have been assigned. In the context of GRASP, the candidate list (CL) is formed by all the possible assignments of the unassigned tasks. The assignments in the CL have a score that measures their attractiveness and the score is updated after every assignment. A restricted candidate list (RCL) is constructed containing the best $\alpha\%$ of the assignments in CL, where the candidate assignment with the smallest score is the best. The next assignment is chosen from RCL at random. The construction step is completed when RCL is empty, that is when all tasks have been assigned. We then use the local search procedure described in section 4.4 to find a local optimum point and this completes a GRASP iteration.

Given a partial solution, the score of assigning task $i$ to processor $k$ is an estimate of the increase in total cost, which includes both the fixed and the communication costs. The incremental total cost is estimated as the sum of these two components:

1. The incremental fixed cost is estimated as the sum of the fixed costs of all the currently unused processors that will have to be used to meet the resource requirements of the unassigned tasks after task $i$ is assigned to processor $k$. The calculation sorts the processors in a “bang-for-buck” order (i.e., by the ratio of fixed cost to total capacity). The fixed costs are added until the capacity requirements of the unassigned tasks are
satisfied (considering that some of these requirements will be satisfied by the processors currently in use). The calculation is a lower bound on the incremental fixed cost because it assumes that the tasks may be split between two or more processors.

2. The incremental communication cost is estimated as the cost of the communication cost of task $i$ and all the unassigned tasks that are not assigned to processor $k$ in future steps. Since future assignments are unknown, we calculate a per-unit-of-capacity communication cost between task $i$ and all the unassigned tasks. The total communication cost is then calculated as the product of the per-unit cost and the difference between the total capacity requirements of the unassigned tasks and the remaining capacity in processor $k$. This difference estimates the tasks (represented by capacity units) that will have to be assigned to other processors.

In our computational experiments, we set $\alpha = 15\%$, a value that is typically recommended and used in GRASP implementations and that also proved to be effective in our setting.
Appendix D: Sensitivity Analysis for the Parameters of the Constrained Task Allocation Problem

Associated with applying the proposed methodology to the CTAP, the value of the “proximity” threshold \( p \) and of the minimum number of “close” local optima \( t \) must be set up. These are the two parameters necessary for the construction of the paired-data set. We let \( p \) assume the values 0.05, 0.10, 0.15, and 0.2. Values above 0.2 would force the model to look for escape directions that are too complex (i.e. escape directions involving the change of more than 20% of the task assignments); values below 0.05 would lead to an empty paired-data set —because it would be difficult to find two local optima in close proximity of each other. We let \( t \) assume the values 2, 5, and 8. We discard the value \( t = 1 \) because the paired-data set would be formed by a set with only one local optimum. Therefore, the learning procedure would find a valid triggering condition but a meaningless guiding constraint, given that once a local optimum \( A \) satisfies the condition there is no need to discriminate between pairs (as there is only one). In other words, the guiding rule would be “If \( A \) is easy to improve, then move to any direction.” Since we are interested in guiding rules that are specific to the characteristics of the local optima, we need to consider only local optima with more than one local optimum close to them. We limit the value of \( t \) to 8, otherwise the paired data set would be empty —not many local optima are close to 8 or more local optima.

For each combination of the \((p, t)\) values, half of the paired-data set is used to solve the learning problem, while the other half is used to evaluate coverage and accuracy. The coverage is the proportion of initial local optima that satisfy the triggering condition. For example, a 30% coverage means that the guiding constraint is activated in 30% of the local optima. The accuracy is a measure of the reliability of the guiding constraint. The first step to compute it is to consider
only the pairs \((A, B)\) such that \(A\) satisfies the triggering condition; among the pairs that satisfy the guiding constraint, the accuracy is the proportion of improving pairs. For example, an 80% accuracy means that when the guiding rule is activated, 80% of the close local optima located along the direction pursued are better than the initial one. We did not consider the combinations \((p, t)\) for which the resulting paired-data set was too large, i.e. when the learning procedure cannot find a positive value solution within 2 minutes. We also discarded the combinations for which the paired-data set was too small (less than 50 pairs), in order to ensure that the guiding rule obtained is valid on a large sample of local optima. Table 22 reports the results of our sensitivity analysis on the combinations that have not been discarded.

<table>
<thead>
<tr>
<th>(p)</th>
<th>(t)</th>
<th>Number of pairs</th>
<th>Coverage</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>2</td>
<td>177</td>
<td>0.23</td>
<td>0.84</td>
</tr>
<tr>
<td>0.15</td>
<td>5</td>
<td>142</td>
<td>0.41</td>
<td>0.85</td>
</tr>
<tr>
<td>0.20</td>
<td>8</td>
<td>216</td>
<td>0.25</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The two combinations \((0.10, 2)\) and \((0.15, 5)\) have a similar accuracy, but we choose the rule obtained with \((0.15, 5)\) because it has a higher coverage, and therefore it can be applied more often.