Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2015-001 January 21, 2015

Efficiently Solving Repeated Integer Linear Programming Problems by Learning Solutions of Similar Linear Programming Problems using Boosting Trees

Ashis Gopal Banerjee and Nicholas Roy

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

CSAIL

Efficiently Solving Repeated Integer Linear Programming Problems by Learning Solutions of Similar Linear Programming Problems using Boosting Trees

Ashis Gopal Banerjee ASHIS@CSAIL.MIT.EDU
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Nicholas Roy NICKROY@CSAIL.MIT.EDU
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Abstract

It is challenging to obtain online solutions of large-scale integer linear programming (ILP) problems that occur frequently in slightly different forms during planning for autonomous systems. We refer to such ILP problems as repeated ILP problems. The branch-and-bound (BAB) algorithm is commonly used to solve ILP problems, and a significant amount of computation time is expended in solving numerous relaxed linear programming (LP) problems at the nodes of the BAB trees. We observe that the relaxed LP problems, both within a particular BAB tree and across multiple trees for repeated ILP problems, are similar to each other in the sense that they contain almost the same number of constraints, similar objective function and constraint coefficients, and an identical number of decision variables. We present a boosting tree-based regression technique for learning a set of functions that map the objective function and the constraints to the decision variables of such a system of similar LP problems; this enables us to efficiently infer approximately optimal solutions of the repeated ILP problems. We provide theoretical performance guarantees on the predicted values and demonstrate the effectiveness of the algorithm in four representative domains involving a library of benchmark ILP problems, aircraft carrier deck scheduling, vehicle routing, and vehicle control.

Keywords: Combinatorial optimization, linear programming, regression, boosting trees, planning

1. Introduction

Combinatorial optimization plays a vital role in many planning problems for autonomous systems ranging from flexible manufacturing cells to unmanned vehicles. The branch-and-bound (BAB) algorithm is popularly used to solve integer linear programming (ILP) problems that occur frequently in such optimization problems. Although commercial solvers such as CPLEX and open-source software libraries like COIN-OR (Lougee-Heimer, 2003) provide efficient implementations, it takes minutes to hours to compute the solutions of ILP problems involving tens of thousands of decision variables and comparable number of constraints. Moreover, the solution needs to be recomputed from scratch even when only a small fraction of the problem data (constraint and objective function coefficients) change as more information becomes available in uncertain environments. We refer to

such large-scale ILP problems that arise multiple times in slightly different forms as *repeated* ILP problems.

A representative example domain is that of airport operations, where hundreds of flights of many different types need to take off and land safely every day with as few delays as possible. Each type of aircraft has a different fuel consumption rate, refueling time, aerial and ground speeds. Since the airport has a limited number of resources (landing strips, launching zones, parking spots, refueling stations, crew members to assist in loading and unloading of baggage etc.), a large number of constraints are imposed on the system. Planning such an operation can be conveniently posed as an ILP problem, where the objective is to minimize the disruption in the nominal schedule of each aircraft while satisfying all the system constraints. An optimum plan can be generated by solving this huge ILP problem offline using hours of computation. However, the environmental conditions are subject to frequent but rarely drastic changes. For example, worse weather conditions may render certain launching zones or landing strips unusable, or increase the fuel burn rates of the aircraft. Mechanical or hydraulic failures in an airborne aircraft may force it to make an emergency landing. In all such cases, the pre-computed plans cannot be used, and new plans have to be generated on-the-fly.

Since it is not possible to compute exactly optimal plans for quick response to dynamic environmental conditions corresponding to repeated ILP problems, we need approximate ILP solvers. Even though many approximation algorithms and parametric techniques are available in the literature, they do not specifically leverage the similarity among the repeated ILP problems. Hence, they cannot quickly estimate near-optimal solutions of problems that are often similar to many other repeated ILP problems for which optimal solutions have been already computed. Moreover, they are often designed for a specific class of ILP problems such as routing, packing, or covering. As a result, they are unable to provide desirable performance for a wide variety of problem domains.

Solving any ILP problem using the BAB algorithm involves solving a large number of relaxed linear programming (LP) problems at the BAB tree nodes, where both the objective function and the constraints are linear functions of continuous-valued decision variables that do not have to be integers. The relaxed LP problems often share multiple common constraints and always contain the same decision variables. Especially in the case of repeated ILP problems, often the decision variables do not change, only some of the objective function and constraint coefficient values change, and a few constraints are added or deleted. Henceforth, we refer to LP problems that contain the same decision variables, and similar objective function and constraint coefficients with some variation in the number of equality constraints as a set of *similar* LP problems.

We observe that the solutions of similar LP problems are usually quite similar themselves in the sense that most of the optimal decision variable values do not change much from one problem to another. Since the optimal values lie at the intersection of the active constraints for any LP problem, our observation indicates that the set of active constraints also do not vary much for similar LP problems. Hence, there is an opportunity to use supervised learning for inferring the correct set of active constraints, and consequently the optimal solution, for any new (test) LP problem by leveraging its similarity with other LP problems present in the training phase. We show that regression can, indeed, be effectively applied to rapidly estimate the optimal LP solutions instead of computing them exactly using a standard optimization solver. Thus, this technique provides a practically useful approach for solving repeated ILP problems online.

More specifically, in this paper, we define the concept of similarity for LP problems and present a boosting tree-based approach, called the Boost-LP algorithm, to learn a set of regression functions that map the objective function and constraints to the individual decision variables of similar LP

problems. We provide absolute performance guarantees in terms of deviations from the optimal for both the LP problem objective function and the individual decision variable values. We then use the objective function performance bounds to prune BAB tree paths while solving the overall ILP problem. We also empirically demonstrate the effectiveness of our algorithm in a collection of benchmark ILP problems, aircraft carrier deck scheduling, vehicle routing, and some well-studied vehicle control problems. The algorithm achieves marked computational speed-up (up to 100 times) as compared to the popular optimization solvers, and obtains promising results with respect to the quality of the overall ILP solutions as the mean deviations of the predicted objective function values are not more than 5% from the true optimal.

2. Related Work

Computing approximately optimal solutions of combinatorial optimization problems in general, and ILP problems in particular, has received a great deal of attention in the literature. The available techniques can be broadly classified into four categories: machine learning, evolutionary optimization, metaheuristics, and multiparametric programming. We now discuss some of the representative work in each of these categories along with a brief summary of their limitations, and conclude by outlining the goals of the current research in addressing some of the shortcomings.

Machine learning: Zhang and Dietterich (1996) applied temporal difference learning to job shop scheduling using a time-delay neural network architecture to learn search heuristics for obtaining close-to-optimal schedules in shorter periods of time than using the previous non-learned heuristics. However, the factor of improvement in computation speed was relatively modest. Lee et al. (1997) used decision trees for scheduling release of jobs into the shop floor and a genetic algorithm (GA) for allocating jobs to the machines and obtained promising results as compared to the conventional single release and allocation approach for 3-7 workstations. Telelis and Stamatopoulos (2001) applied a kernel regression-supported heuristic methodology in knapsack and set partitioning problems and obtained solution qualities of about 60% (measured with respect to the optimal) for constraint matrices involving up to 6000 rows and 25 columns. Vladušič et al. (2009) employed locally weighted learning, naïve Bayes, and decision trees for job shop scheduling over heterogenous grid nodes in conjunction with baseline round-robin, random, and FIFO algorithms to obtain improvements over using the baseline algorithms on their own for tasks involving 1000 jobs. We can summarize that although machine learning approaches provide a promising research direction, further work is needed to enhance the computational speed-up factor, and the ability to solve larger-sized problems involving tens of thousands of decision variables (instead of thousands) for a wide variety of problem domains.

Evolutionary optimization: Evolutionary techniques have been most popularly used for solving the class of vehicle routing problems (VRP), where a number of cities must be traversed just once by one or more vehicles, all of which must return to the originating cities. The optimum solution is the set of routes that minimizes the total distance traveled. Representative examples include a genetic algorithm to solve the multi-VRP (involving multiple vehicles) developed in Louis et al. (1999), an ant colony optimization method to search for multiple routes of the VRP presented in Bell and McMullen (2004), and a combination of k-means clustering and genetic algorithm to solve the multi-VRP proposed in Nallusamy et al. (2009). These approaches have had mixed success as

none of them work uniformly well for problems of varying sizes and types, and the benefit in terms of computational savings is not that large.

Metaheuristics: Metaheuristic techniques have been widely used to solve a variety of combinatorial optimization problems, and the most common technique for ILP problems is known as LP rounding. Rounding refers to the process of obtaining integral solutions from fractional values of the decision variables. Rounding techniques can be broadly classified into two types: those that round variables non-deterministically (known as randomized rounding) and those that perform rounding deterministically. Within randomized rounding (RR), notable works include the basic approach proposed in Raghavan and Thompson (1987), Raghavan (1988), and Srinivasan (1999) for packing, routing, and multicommodity flow problems, the variant developed in Goemans and Williamson (1994) for the maximum satisfiability problem, and the combination of RR with semidefinite programming in Goemans and Williamson (1995) for the maximum cut problem. Other key works include simultaneously rounding a set of variables in Bertsimas et al. (1999) for the maximum cut problem, better handling of the equality constraints in Arora et al. (2002) for dense graph problems, and randomized metarounding in Carr and Vempala (2002) for the multicast congestion problem. Within deterministic rounding, representative works include iterated rounding in Jain (2001) for the generalized Steiner network problem, conditional expectation in Chudak and Shmoys (2003) for the uncapacitated facility location problem, and pipage rounding in Ageev and Sviridenko (2004) for the maximum coverage problem. We refer the reader to Chapter 7 in Gonzalez (2007) and Chapters 4 and 5 in Williamson and Shmoys (2011) for a much more comprehensive survey of all the available rounding techniques. Although rounding approaches provide useful performance bounds on several types of ILP problems, no single approach is found to work well on all types of problems. Moreover, all of these approaches obtain exactly optimal solutions of LP problems, and, hence, even though they have polynomial time complexities, they do not provide orders of magnitude reduction in actual computation times.

Multiparametric programming: Another class of ILP solution techniques that is related to our work is multiparametric programming. In multiparametric programming, the optimization problem data are functions of multiple parameters. This technique provides a valuable tool to analyze the effect of variations in optimization problems that arise due to model uncertainties, and has been popularly used in the context of process scheduling and model predictive control. The effects of parameterizing the objective function and the right-hand side of the constraint equations are most commonly studied in the literature. One of the earliest solution methods was proposed in Gal and Nedoma (1972), who partitioned the parameter space into critical regions such that there is one optimal solution for all the LP problems that are parameterized within each region. Later on, several approaches have been developed that extend the previous method using, for example, a geometric representation Adler and Monteiro (1992), a recursive construction of the critical regions Borrelli et al. (2003), or a linear interpolation based approximate generation of the critical regions Filippi (2004). With respect to multiparametric ILP problems, Dua and Pistikopoulos (2000) decomposed the overall problem into a multiparametric LP problem with fixed values of the integer variables and a mixed ILP with the parameters treated as variables. Li and Ierapetritou (2007) developed another decomposition technique, wherein each sub-problem encodes the information around a given parameter value and is then solved using a series of multiparametric LP and mixed integer linear or non-linear programming problems. We refer the interested reader to Pistikopoulos et al. (2007) and Alessio and Bemporad (2009) for a more thorough survey of other approaches. Even though these

approaches have been successful where there are uncertainties in the objective function or right-hand side constraint vector, little work has been done to handle the general case where uncertainties exist in the left hand side constraint matrix as well. Moreover, the majority of approaches build an exact representation of the parameter space, which makes solution estimation relatively hard both in terms of computation time and memory requirements.

Warm starting interior-point solution: A final category of techniques that is quite similar to our work is warm starting interior-point solutions of LP problems. In this category, the original instance of an LP problem is solved using any interior-point method, and the perturbed instance of an identical dimensional problem with slightly different data is then solved using a warm start strategy. Interior-point iterates of the original problem are used to obtain warm start points for the perturbed problem, so that the interior-point method is able to compute the solution in fewer iterations as compared to the case when no prior knowledge is available. Representative works include the least squares correction and Newton step correction methods presented in Yildirim and Wright (2002) and Elizabeth and Yildrim (2008), using the l_1 penalized forms of the given LP problems in Benson and Shanno (2007), and creating additional slack variables to define equivalent LP problems in Engau et al. (2009) and Engau et al. (2010). While all the warm start approaches provide an interesting research direction to tackle the issue of efficiently solving similar LP problems that we are interested in, state-of-the-art methods have not yet been found to provide computational gains over competing methods in large-scale problems involving tens of thousands of decision variables and constraints.

In this paper, we present an alternative regression-based approach that leverages the underlying structure of the LP problems (the correlation between active constraints and optimal solutions) to develop performance guarantees that often do not exist for the current techniques. Our approach is applicable to a wide variety of optimization domains, including routing, scheduling, and control, unlike most of the existing techniques. It is also found to scale well to large problem domains, again unlike almost all of the current methods. Furthermore, we obtain more than an order of magnitude speed-up in computation time while solving new problems instances, which has not been reported previously to the best of our knowledge. An additional application of our approach to a chance-constrained optimal control problem with non-convex constraints can be found in Banerjee et al. (2011).

3. Boost-LP Algorithm

We are interested in developing a regression model of a system of similar linear programming (LP) problems to predict the solution of any new but similar LP problem. The predicted solutions are then used directly within the search process (tree nodes) of the branch and bound (BAB) algorithm to obtain approximate solutions of repeated integer linear programming (ILP) problems efficiently. The methods for learning the boosting tree based regression model and performing inference over the learned model are referred to as the Boost-LP Learning and Boost-LP Inference algorithms respectively. While the actual boosting tree model is quite standard with a few modifications, there are two key novel aspects of the Boost-LP algorithms: a) using a formal definition of similarity metric to cluster the system of similar LP problems and identify a suitable set of training LP problems to estimate the optimum solution of test LP problems, and b) exploiting the structure of LP problems to generate a predictor variable vector that transforms the LP problems to a lower-dimensional space for the regression model to operate on so as to yield accurate predictions. We begin with the

similarity clustering operation before formulating the regression problem and then presenting the boosting tree model and the overall ILP solution procedure.

Throughout the paper, unless otherwise stated, vectors are represented as column matrices, rectangular matrices are denoted by capital symbols, indices over rows are written as superscripts, indices over columns are written as subscripts and denoted by i or o, and indices over elements of a set (e.g. training set problem instances) are also written as subscripts but denoted by j, k, or w. If both column index and set element index are associated with a symbol, then the column index is written before the element index and the two indices are separated by a comma. Thus, $x_i \in \Re$ is the i-th component of the vector x, whereas $x_{i,k}$ denotes a specific value of x_i corresponding to the k-th element of a set.

3.1 Clustering using Similar LP Problems

Any LP problem is represented in the standard form as:

$$\min z = c^T x \tag{1}$$

$$\text{s. t. } Ax = b \tag{2}$$

$$x \geq 0 \tag{3}$$

where $c, x \in \Re^n$, $A \in \Re^{m,n}$, $b \in \Re^m$, and each individual equation of the system (2) given by $(a^r)^T x = b^r, 1 \leq r \leq m$, defines a hyperplane in $\Re^n$. The constraint matrix, A, constraint vector, b, and objective function vector, c, are together termed as the data of the LP problem, and x is referred to as the solution or decision variable vector. The optimum solution is denoted by x^* and the feasible region is an affine subspace formed by the intersection of the hyperplanes in $\Re^n$. An arbitrary LP problem with less than (or greater than) type inequality constraints and unbounded decision variables is converted to the standard form by adding (or subtracting) slack (or surplus) variables and by replacing every unbounded decision variable x_i by $x_i^+ - x_i^-$, such that $x_i^+, x_i^- \geq 0$, respectively. This conversion implies that the number of constraints is always less than the number of decision variables for an LP problem in the standard form. Any maximization problem is represented in minimization form by simply multiplying the objective function vector by -1. An ILP problem has additional integer restrictions on some or all of the decision variables.

To better understand the various concepts and steps of our approach, we first give an example of two repeated ILP problems taken from Lee and Mitchell (2001):

$$\begin{aligned} ILP_{E_1} : \min \quad & -13x_1 - 8x_2 \\ \text{s. t.} \quad & x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 2x_2 + x_4 = 20 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2, x_3, x_4 \quad \text{are integers} \end{aligned} \qquad \text{and} \qquad \begin{aligned} ILP_{E_2} : \min \quad & -14x_1 - 8x_2 \\ \text{s. t.} \quad & x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 2x_2 + x_4 = 21 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2, x_3, x_4 \quad \text{are integers.} \end{aligned}$$

Solving the LP-relaxations of ILP_{E_1} and ILP_{E_2} without integer restrictions on any of the decision variables at the root nodes of the respective BAB trees, we obtain $[2.5 \quad 3.75 \quad 0 \quad 0]^T$ and $[2.75 \quad 3.625 \quad 0 \quad 0]^T$ as the solution vectors. We then branch both the BAB trees using $x_1 = 3$ and $x_1 = 2$ as additional constraints. We obtain $[3 \quad 2.5 \quad 2 \quad 0]^T$ and $[2 \quad 4 \quad 0 \quad 2]^T$ as the solution vectors in the case of ILP_{E_1}. Let us consider the relaxed LP problem in the right child node as

our first example problem LP_{E_1}. For ILP_{E_2}, branching at the root node yields $[3 \quad 3 \quad 1 \quad 0]^T$ and $[2 \quad 4 \quad 0 \quad 3]^T$ as the solution vectors. We again consider the relaxed LP problem in the right child node as our other example problem LP_{E_2}. The two example problems are mathematically represented as:

$$\begin{aligned} LP_{E_1} : \min \quad & -13x_1 - 8x_2 \\ \text{s. t.} \quad & x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 2x_2 + x_4 = 20 \\ & x_1 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned} \quad \text{and} \quad \begin{aligned} LP_{E_2} : \min \quad & -14x_1 - 8x_2 \\ \text{s. t.} \quad & x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 2x_2 + x_4 = 21 \\ & x_1 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0 \quad . \end{aligned}$$

We are now ready to define what we mean by two similar LP problems and show that our example problems satisfy the criteria.

Definition 1 *For a given $\xi_c \in (0, 1]$ and $\xi_A \in (0, 1]^{n+1}$, two feasible and bounded LP problems, LP_1, with data (A_1, b_1, c_1), m_1 constraints, and decision variable vector $x \in \Re^n$, and LP_2, with data (A_2, b_2, c_2), m_2 constraints, and the same decision variable vector x, are said to be ξ-similar, where $\xi = [\xi_c \quad \xi_A^T]^T$, if*

$$\frac{\|c_1 - c_2\|_2}{\|c_1\|_2 + \|c_2\|_2} \leq \xi_c \quad \textit{and} \tag{4}$$

$$\frac{\|a_{i,1} - a_{i,2}\|_2}{\|a_{i,1}\|_2 + \|a_{i,2}\|_2} \leq \xi_{i,A}, 1 \leq i \leq n+1 \quad . \tag{5}$$

Here, $a_{i,q} = [a_{i,q}^1, \ldots, a_{i,q}^{m_q}]^T, 1 \leq q \leq 2$, where $a_{i,q}^r$ is the element in the r-th row and i-th column of the augmented constraint matrix $[A_q \quad b_q]$. If $m_1 \neq m_2$, $|m_1 - m_2|$ zeros are appended to either $a_{i,1}$ or $a_{i,2}$ to make their dimensions equal.

As the above definition requires that any two ξ-similar LP problems must have the same decision variables, both the slack/surplus variables and the originally unbounded decision variables must be identical for the two problems under consideration. Hence, the two LP problems must have the same number of inequality constraints when they are represented arbitrarily. The number of equality constraints can, however, be different in the arbitrary representation, and, thus, the total number of constraints may be different when the problems are written in the standard form.

We use the column vectors of the augmented constraint matrices comprising of the ith coefficients of *all* the constraints in the two LP problems that are denoted by $a_{i,1}$ and $a_{i,2}$, respectively, to make the dimensionality of the similarity vector ξ_A constant and equal to $n + 1$. The alternative choice of the rows of the augmented matrices, i.e. the individual constraints of the two problems denoted by a_1^r and a_2^r, respectively, causes issues when $m_1 \neq m_2$. Although we can still compute ξ_A by discarding the extra $|m_1 - m_2|$ constraints, the extra constraints may significantly modify the feasible region and consequently the optimal solution of the corresponding LP problem, resulting in dissimilar problems being incorrectly labeled as ξ-similar.

Note that the constraints are always specified in the same order for the LP problems in the BAB tree nodes of our repeated ILP problems. This allows us to compute the shortest L^2 distance between every pair of $a_{i,1}$ and $a_{i,2}$. For a more general training set of ξ-similar LP problems, we would need to explicitly find the best matching of the constraints, possibly using some variant of the well-known Hungarian algorithm with the L^2 distance as the cost.

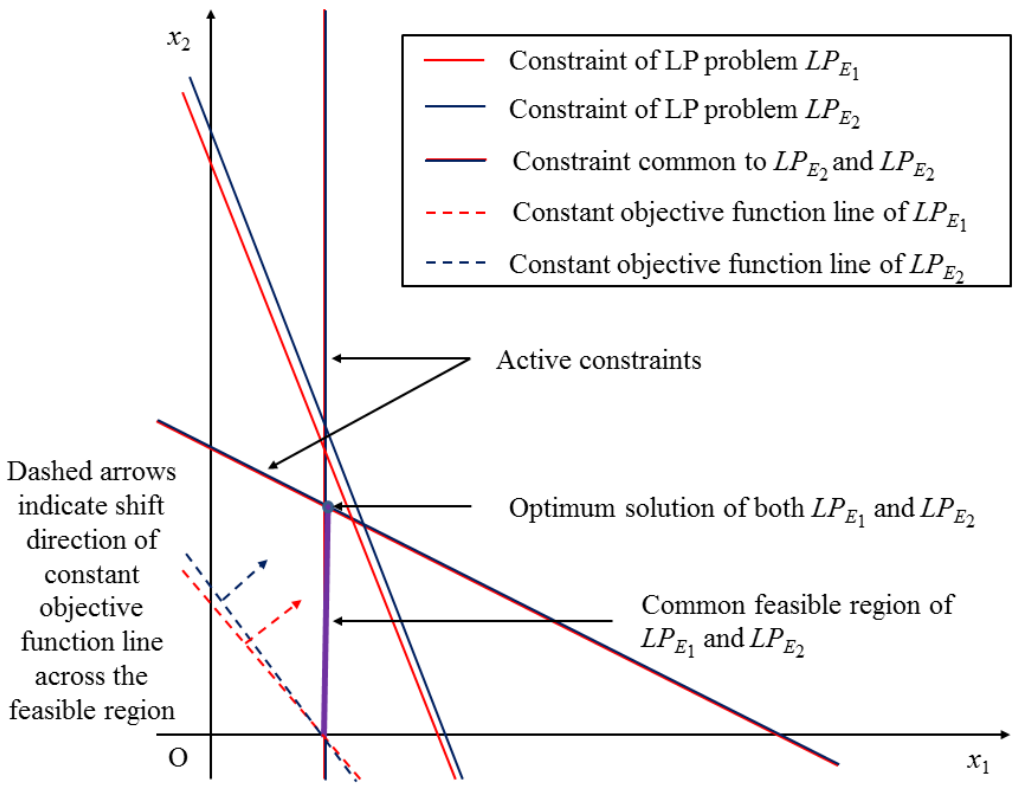

Figure 1: Schematic illustration of ξ-similar LP problems that have a common feasible region and identical optimal solutions

Let us select $\xi_c = 0.1$ and $\xi_A = [0.1 \quad 0.1 \quad 0.1 \quad 0.1 \quad 0.1]^T$. Following the notation in Definition 1, we can write $c_1 = [13 \quad 8]^T$, $c_2 = [14 \quad 8]^T$. The augmented constraint matrices for LP_{E_1} and LP_{E_2} are equal to $\begin{pmatrix} 1 & 2 & 1 & 0 & 10 \\ 5 & 2 & 0 & 1 & 20 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix}$ and $\begin{pmatrix} 1 & 2 & 1 & 0 & 10 \\ 5 & 2 & 0 & 1 & 21 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix}$ respectively. We then apply (4) and (5) to get $\frac{\|c_1 - c_2\|_2}{\|c_1\|_2 + \|c_2\|_2} = 0.0319$, $\frac{\|a_{i,1} - a_{i,2}\|_2}{\|a_{i,1}\|_2 + \|a_{i,2}\|_2} = 0, i = 1, \ldots, 4$, and $\frac{\|a_{5,1} - a_{5,2}\|_2}{\|a_{5,1}\|_2 + \|a_{5,2}\|_2} = 0.0218$. Thus, we conclude that LP_{E_1} and LP_{E_2} are ξ-similar. Figure 1 shows the geometric representation of LP_{E_1} and LP_{E_2}, and illustrates our notion of similarity. The arrows originating from the constant objective function value lines show the directions along which the lines need to be translated so as to optimize the problems within the feasible regions. The feasible regions and optimal solutions are identical (or nearly identical in other cases) for the ξ-similar LP problems, as the constraint and objective function coefficients are very close in terms of Euclidean distances. We now extend the concept of ξ-similar LP problems to first define a ξ-similar LP problem set and then a ξ-similar LP problem space, both of which are used in the Boost-LP Learning algorithm to generate a suitable training set.

Definition 2 *For a given ξ and a finite collection of LP problems LP^f, the ξ-similar LP problem set LP^s is defined as a subset of LP^f such that every problem $\in LP^s$ has the same decision variable vector, and $LP_{j_1} \in LP^s$ is ξ-similar to at least one other $LP_{j_2} \in LP^s, j_1 \neq j_2$.*

This set is represented as a collection of clusters, which are themselves sets, such that all the LP problems belonging to a cluster are pairwise ξ-similar to each other. We refer to each such cluster as a *ξ-similar LP problem cluster*. With respect to the BAB trees of the repeated ILP problems that we want to solve efficiently, every cluster consists of the relaxed LP problems along the branches formed by imposing identical or nearly-identical constraints on parent LP problems that are themselves ξ-similar. For example, LP_{E_1} and LP_{E_2} are both constructed by branching the root nodes of the respective trees by enforcing the additional $x_1 = 2$ constraint, and are, thus, ξ-similar. The ξ-similar LP problems can also comprise of neighboring sibling nodes, or even parent and child nodes in the nearby levels in large trees, where the variations in the constraints involving a few of the decision variables are small enough to satisfy the overall similarity criteria.

Definition 3 *For a given ξ-similar LP problem set LP^s of cardinality N, the ξ-similar LP problem space (F^s, C^s) is defined as a subset of $\Re^n \times \Re^n$, where F^s is the union of the feasible regions of all the LP problems present in LP^s, and C^s is given by $[c_{1,\min}, c_{1,\max}] \times \ldots \times [c_{n,\min}, c_{n,\max}]$. Here, c_i denotes the i-th component of the objective function vector c, and $c_{i,\min(\max)}$ is equal to $\min(\max)\{c_{i,1}, \ldots, c_{i,N}\}$. LP^s is then referred to as the member set of (F^s, C^s).*

The union operation defines a combined feasible region F^s that contains the optimal solutions of all the member set LP problems even though portions of it may be infeasible for particular LP problems. Similarly, a combined objective function space is performed by taking the Cartesian product of n 1-dimensional spaces defined by the lower and upper bounds of the elements of the c vector.

The geometric representation of a ξ-similar LP problem space is shown in Figure 2. The member space contains two ξ-similar LP problem clusters. The first cluster consists of LP_{E_1} and LP_{E_2}. The second cluster contains two other problems, LP_{E_3} and LP_{E_4}, with a common objective function and sets of constraints that are significantly different from the objective functions and the constraints present in LP_{E_1} and LP_{E_2}. The optimal solutions and feasible regions vary significantly among the problems in the different clusters. This shows the usefulness of our definition, which enables us to simultaneously group LP problems that are ξ-similar to each other but not to the rest in separate clusters, and yet define an overall mapping space for regression functions to operate on. We now use this concept to characterize any LP problem that need not be an element of the member set but belongs to this problem space. Such LP problems form our test cases, for which we infer solutions using the Boost-LP Inference algorithm. We use the ξ-similar LP problem clusters to aid us in obtaining near-optimal solutions for test problems by identifying only the ξ-similar member set problems that should be used in prediction.

Definition 4 *Any LP problem is said to belong to a ξ-similar LP problem space (F^s, C^s), if it is ξ-similar to all the problems in one of the clusters of the member set of (F^s, C^s), its feasible region has a non-empty intersection with F^s, and the point defined by its objective function vector lies inside C^s.*

The condition of non-empty intersection of the feasible region of the LP problem under consideration with F^s is required to exclude pathological LP problems where the feasible regions lie

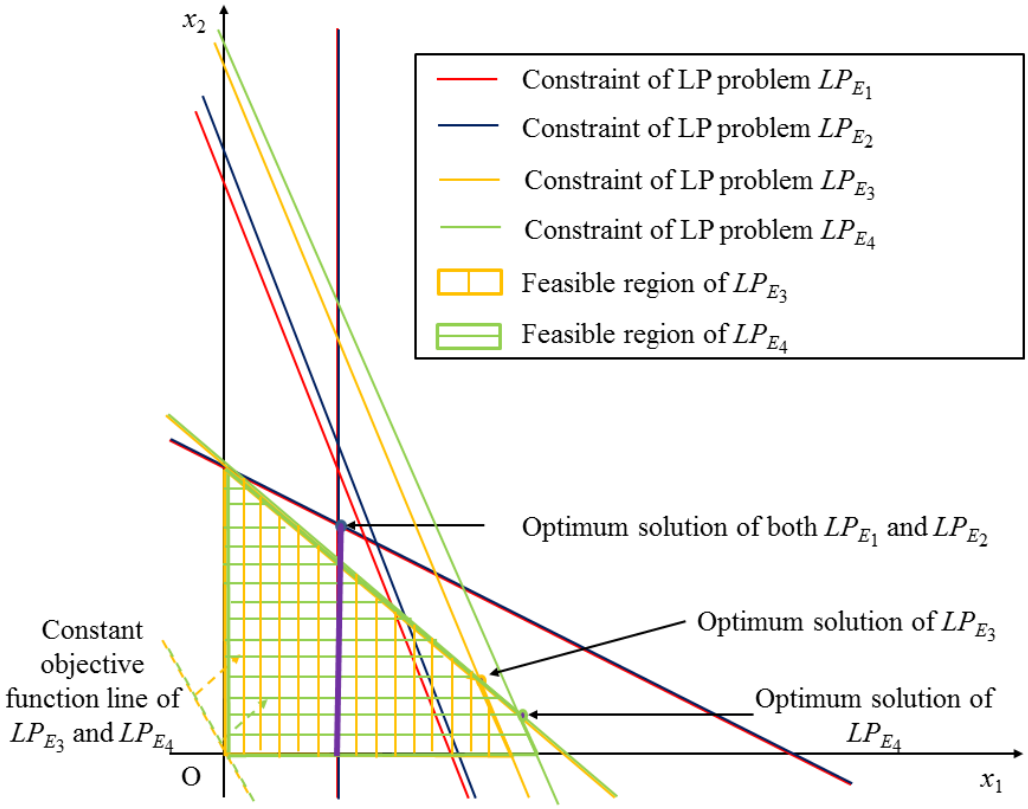

Figure 2: Schematic illustration of the member set of ξ-similar LP problem space consisting of four LP problems. LP_{E_1} and LP_{E_2} form one ξ-similar LP problem cluster, and LP_{E_3} and LP_{E_4} form the other cluster

completely outside of, but close enough to F^s, so that we cannot estimate any feasible solution based upon the existing training data even though ξ-similarity still holds for any small non-zero value of the ξ vector components. The other requirement of the location of the objective function vector inside C^s arises due to the fact that all the components of c are used in constructing the predictor variable vector of the regression model (as explained in Section 3.2.1), and, hence, we cannot use outlier values to query the model to estimate a solution.

We can now begin explaining how to use a given training set of N feasible and bounded LP problems and their optimal solutions to predict the unknown optimum solution of any arbitrary test LP problem. The training set problems are first arranged into a ξ-similar LP problem set LP^s using a minimum linkage clustering algorithm. Every problem is first considered as an individual cluster and the cluster pairs are then hierarchically merged using the minimum ξ-similarity based distance (Definition 1) between any two elements in the two clusters. We discard all the problems that are not member elements of one of the ξ-similar problem clusters. This clustered training set forms the member set of a ξ-similar LP problem space (F^s, C^s). (A_k, b_k, c_k) represents the data of the k-th LP problem, $LP_k \in LP^s$, such that $c_k, x \in \Re^n$, $A_k \in \Re^{m_k,n}$, and $b_k \in \Re^{m_k}$ $\forall k$. LP^s is then used

to formulate the regression problem as described in Section 3.2, and learn the boosting tree-based regression models as detailed in Section 3.3. For inference, we first verify that the test LP problem belongs to (F^s, C^s) but is not identical to any of the problems present in LP^s before applying a suitable form of the regression predictor vector and the learned boosting tree models to estimate the optimum solution.

3.2 Regression Formulation

The Boost-LP Learning algorithm outputs a set of n regressor functions, each of which is a mapping of the form $f_i : (F^s, C^s) \rightarrow \Re$, which is then used by the Boost-LP Inference algorithm to estimate the optimum x for any test LP problem. Each regressor function requires a predictor variable vector denoted by v, and a scalar response variable represented by y. Thus, we want to learn $f_i : v \mapsto y_i, 1 \leq i \leq n$, where v is identical for all f_i, and $y_i = x_i$ for every f_i. A separate regressor function is used for inferring each component of x to avoid the sample complexity associated with learning multiple response variables simultaneously. The inter-dependence of the decision variables is captured by incorporating the problem constraints in the predictor variable vector and by introducing the penalization loss function (discussed in section 3.3). The optimum decision variable value, x_i^*, acts as the training set response variable for the corresponding function f_i.

Constructing the predictor variable vector is significantly more challenging for both training and test problems. The challenges arise from the facts that a) we only want to retain those useful problem data (constraint and objective function coefficients) that result in certain feasible solutions becoming optimal, and b) the vector is reasonably compact and its size is independent of the number of training and test LP problem constraints. We devote the rest of this sub-section to explain our construction procedure that address these challenges.

3.2.1 PREDICTOR VARIABLE VECTOR CONSTRUCTION METHOD

A naïve way to generate the predictor variable vector is to include all the elements in A_k, b_k, and c_k following some sequence. For example, we can construct a predictor vector by concatenation as $[a_1^1, \ldots, a_n^1, b^1, \ldots, a_1^{m_k}, \ldots, a_n^{m_k}, b^{m_k}, c_1, \ldots, c_n]^T$. This construction not only results in a very high-dimensional vector, it can also lead to variable dimensionality for different training set problems since the number of constraints, m_k, may not be the same for all LP_k. Adding elements to make the dimensionality constant leads to two further issues. First, we need a systematic way of populating these elements. And, second, we cannot handle the scenario where the test problem results in a higher-dimensional vector than the one generated from the training set. Nevertheless, some pilot trials were conducted to evaluate the effectiveness of this method without considering the second issue, either by inserting zeros or random values that lie uniformly within the span of the coefficients for the additional elements. In both the cases, the regression model performed significantly worse than using the v construction procedure discussed below in a scheduling problem domain (described in section 5.2) with respect to both computation time and solution prediction accuracy.

Construction of v during learning: To construct a useful representation of v during the training phase (we describe how to do the same during testing afterwards), we begin from the system of constraints for the kth training set problem LP_k as given by

$$A_k x = b_k \tag{6}$$

where $A_k \in \Re^{m_k,n}$ and $b_k \in \Re^{m_k}$. From the underlying theory of LP problems, the optimum solution is given by a *basic feasible* solution, which always exists for any feasible and bounded LP problem that is expressed in the standard form. Geometrically, every basic feasible solution represents a corner or extreme point (vertex) of the feasible region as shown in Figures 1 and 2. This property of obtaining the basic feasible solution from the system of linear equations defined by the set of active constraints[1]is henceforth referred as the *simplex* property. Accordingly, (6) can be partitioned as

$$A_{B,k}x_B + A_{NB,k}x_{NB} = b_k$$

where $A_k = [A_{B,k} \quad A_{NB,k}]$, $x = [x_B^T \quad x_{NB}^T]^T$, and x_B and x_{NB} refer to the basic and non-basic solutions, respectively. Thus, the basic solution consists of m_k decision variables, whereas the non-basic solution contains the remaining $(n - m_k)$ decision variables. Since the basic solution is obtained by setting $x_{NB} = 0$ and then inverting $A_{B,k}$, it implies that $A_{B,k}$ is of full rank and consists of a set of linearly independent active constraints. We use this property to construct a suitable representation of v.

To do so, we first introduce a matrix E_k to transform (6) to a form analogous to its left-inverse solution[2]as represented by

$$d_k = (E_k^T A_k)^{-1} E_k^T b_k \tag{7}$$

where $E_k \in \Re^{m_k,n}$ should be non-negative to drive every component of d_k toward non-negative values and generate feasible solutions in accordance with (3).

We choose a particular form of E_k that is given by m_k rows of n-dimensional unitary vectors[3]. Mathematically, we represent $E_k = \{e_{q_1}, \ldots, e_{q_{m_k}}\}^T$ and A_k as $\{a_1, \ldots, a_{m_k}\}^T$, where each $e_{q_i} \in \Re^{1,n}$ is a unitary row vector and $q_i \in \{1, \ldots, n\}$, $1 \leq i \leq m_k$, denotes the position of the unity element. Thus, E_k will be of the general form $\begin{pmatrix} 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 0 & 0 & \cdots & 0 \end{pmatrix}$. We now enforce a sequence of additional rules to ensure the invertibility of $E_k^T A_k$ and identify a unique representation of E_k that uses the simplex property.

- **Rule 1**: Select q_is such that $e_{q_i} a_i^T \neq 0 \quad \forall i$. In other words, for any given row of E_k, a unity element is placed in a column only when the corresponding coefficient in the same row of A_k is non-zero. This selection ensures that $E_k^T A_k$ is strongly non-singular when A_k is of full rank and $m_k = n$ as all the determinants of its principal submatrices are non-zero. However, this rule does not lead to a unique selection of E_k or guarantee non-singularity for the general case of $m_k < n$.

- **Rule 2**: Select q_is such that no column of E_k has more than one unity element and the corresponding coefficient in the same row of A_k is a basic variable at the optimum solution, x_k^* that is known during training. In conjunction with Rule 1, this rule implies that every basic variable at x_k^* is associated with a particular constraint of A_k, referred to as the *relevant* constraint.

1. Active or tight constraints are those which hold tightly, i. e. become or remain equalities for a given x.
2. We can opt for a form analogous to the right-inverse solution; this does not affect any of our subsequent design choices or the results presented in this paper.
3. A unitary vector refers to a type of unit vector such that only one of the elements of the vector is unity and the rest are all zeros. For example, $[0 \quad 0 \quad 1 \quad 0]$ is a 4-dimensional unitary row vector.

After enforcing Rules 1 and 2, if a column of E_k only contains zero elements, it corresponds to a non-basic variable at x_k^*.

- **Rule 3**: Eliminate all the columns from E_k that only contain zero elements. Also, eliminate the same constraints from A_k. These elimination operations turn both E_k and A_k into square matrices of dimensionality $m_k \times m_k$, where exactly one unity element is present in every row and column of truncated E_k.

This form of the truncated E_k matrix occurs in the Gauss elimination method of solving a system of linear equations and is referred as the *permutation* matrix P_k. It has some useful properties, notably orthogonality and the effect of permuting the rows and columns of another matrix as a result of premultiplication and postmultiplication operations, respectively. The truncated form of A_k is the same as $A_{B,k}$, which means that the relevant constraints represent the linearly independent active constraints at x_k^*. Transformation (7) then becomes equivalent to

$$d_k = [d_{B,k}^T \quad d_{NB,k}^T]^T = [[(P_k^T A_{B,k})^{-1} P_k^T]^T \quad 0^T]^T b_k \tag{8}$$

where $d_{B,k}$ and $d_{NB,k}$ represent the basic and non-basic partitions of d_k corresponding to x_B and x_{NB} at x_k^*. Also, $d_{B,k} = A_{B,k}^{-1} b_k = x_{B,k}^*$ as formally proved below in Lemma 5. Observe that, based on Lemma 5, we no longer need any of the rules to construct d_k, and can simply enforce it to be equal to x_k^*. However, this procedure sets the background for constructing the predictor variable vector for the test LP problems as discussed later in this sub-section.

Lemma 5 *d_k is equal to the optimum solution x_k^* of any training set LP problem LP_k if it is chosen according to Rules 1-3.*

Proof

Mathematically, $d_k = [d_{B,k}^T \quad d_{NB,k}^T]^T$, where all the components of $d_{NB,k}$ are equal to 0 by construction, and the size of $d_{NB,k}$ is $(n - m_k)$. Thus, $d_{NB,k}$ is identical to the non-basic optimum solution of LP_k, $x_{NB,k}^*$. Now, $d_{B,k}$ is given by $(P_k^T A_{B,k})^{-1} P_k^T b_k$ from (8) and, hence, the Lemma is proved if we can show that $d_{B,k}$ is the same as the basic optimum solution $x_{B,k}^*$. This follows as

$$\begin{aligned} d_{B,k} &= A_{B,k}^{-1} (P_k^T)^{-1} P_k^T b_k \\ &= A_{B,k}^{-1} (P_k^T)^T P_k^T b_k \quad \text{from the orthogonality property of the permutation matrix } P_k \\ &= A_{B,k}^{-1} (P_k P_k^T) b_k \\ &= A_{B,k}^{-1} (P_k P_k^{-1}) b_k \quad \text{again from the orthogonality property of } P_k \\ &= A_{B,k}^{-1} b_k = x_{B,k}^* \quad . \end{aligned} \tag{9}$$

■

Finally, the common predictor variable vector v_k for all the regressor functions $f_i, 1 \leq i \leq n$, in a problem LP_k is formed by augmenting d_k with c_k as given by

$$v_k = [d_{1,k}, \ldots, d_{n,k}, c_{1,k}, \ldots, c_{n,k}]^T \quad . \tag{10}$$

We insert c_k in v_k so that the regression model would not try to infer optimal solutions (response variables) using the modeled response variables of training set LP problems with very different objective functions as compared to the test problem. Note that the size of v_k is always constant and equal to $p = 2n$ for any value of k as both d_k and $c_k \in \Re^n$.

Generating additional training data: If the range of $d_i, 1 \leq i \leq n$, is large in the training data set, then more problems need to be generated so that additional values of those sparsely distributed components are used in learning. We first sort all the $d_{i,k}$ values in non-decreasing order and check whether $(d_{i,k+1} - d_{i,k}), 1 \leq k \leq N-1$, is greater than a pre-defined threshold ζ_i; if yes, we add a minimum number of problems with uniformly spaced values of d_i such that $(d_{i,k+1} - d_{i,k}) \leq \zeta_i \forall k$. Henceforth, N refers to the total number of training data samples that includes both the original and the newly added samples. To generate complete LP problems corresponding to the newly created d_i values, we select the average values of the two original points that violated the threshold condition for all the components of v except d_i. Although this procedure directly generates the c vector, it does not yield a solution for A and b. We handle this by selecting the A and P matrices for the lower d_i valued point, and then solving for b using (7). Although the user-specified vector ζ helps us in bounding the approximation errors (shown in section 4) and results in improved performance (shown in section 5.2), it can potentially lead to a large number of samples. Even otherwise, acquiring sufficient amount of training data to guarantee good quality of the estimated solutions is time-consuming. We briefly discuss this as a part of future work in section 6 since addressing it is beyond the scope of the current work.

Construction of v during inference: As discussed earlier during the training phase construction, the main motivation is to come up with a compact form of constant sized predictor vector v that leverages the simplex property of LP problems. This motivation led to a series of rules to identify a suitable form of the square permutation matrix P with dimensionality equal to the number of problem constraints, and consequently the number of basic solution variables. We now adopt a similar procedure to compute v for the test LP problem LP_{te} with constraint matrix A_{te}, constraint vector b_{te}, and objective function c_{te}.

Since optimum x, x^*_{te}, and the set of basic variables at x^*_{te} are unknown for LP_{te}, we can apply Rules 1 and 3 but not Rule 2. Hence, we enforce the following sequence of rules in addition to Rules 1 and 3 to compute P_{te} and the resultant d_{te} for the test LP problem.

- **Rule 2a**: (Applied in place of Rule 2): Select the columns of P_{te} that correspond to non-zero coefficients of the non-slack and non-surplus decision variables in the same rows of A_{te} as the candidate locations of the unity elements q_is. Furthermore, select q_is such that no column of P_{te} has more than one unity element.

Rules 1 and 2 imply that every row of P_{te}, and, as a result, every row of A_{te} is associated with a unique non-slack or non-surplus decision variable. As before, the constraint associated with any decision variable is termed as a relevant constraint. While training, the relevant constraints are identical to the set of linearly independent active constraints at optimum x leading to Lemma 5, the objective here is to select the relevant constraints to be as close as possible to the active constraints at x^*_{te} to obtain a useful form of d_{te} , and correspondingly an accurate estimate of x^*_{te}. The selection is based on the observation that some of the non-slack and non-surplus variables must belong to the basic solution set at x^*_{te}.

Depending on the number of test problem constraints m_{te} and the total number of non-slack and surplus variables that satisfy Rule 1 (termed as candidate variables) n_{NS}, two cases arise that require different q_i selection procedures:

- **Case 1**: $m_{te} = n_{NS}$: Selection is straightforward here as every candidate variable can be associated with a particular row of P_{te}. We retain all possible P_{te}s for varying choices of q_is.

- **Case 2**: $m_{te} \neq n_{NS}$: Selection is more challenging here as we need to either exclude some candidate variables when $m_{te} < n_{NS}$ or include some slack or surplus variables that satisfy Rule 1 when $m_{te} > n_{NS}$. If variables have to excluded, we select them randomly from the list of variables that are not present in the union of the basic variables at the optimal solutions of the training LP problems that are ξ-similar to LP_{te}, LP^s_{te}. If additional variables have to be included, we choose them randomly from the intersection of the basic variables at the optimal solutions of the problems present in LP^s_{te}. Once the required number of candidate variables have been excluded or included, as in the previous case, we retain all possible choices of P_{te}.

- **Rule 4**: (Applied after Rules 1-3): Partition d_{te} into *potential* basic and non-basic sets as $d_{te} = [d^T_{B,te} \quad d^T_{NB,te}]^T$, where $d_{B,te} \in \Re^{m_{te}}$ and $d_{NB,te} \in \Re^{n-m_{te}}$. All the components of $d_{NB,te}$ are set to 0, and the constraint $d_{B,te} \geq 0$ is imposed to ensure feasibility from (3). Compute the set of all potential $d_{B,te}$ using the set of previously computed P_{te} analogous to (8) as

$$d_{B,te} = (P^T_{te} A_{B,te})^{-1} P^T_{te} b_{te} \quad .$$

- **Rule 5**: Choose a particular d_{te} from the potential set obtained after applying Rules 1-4 to minimize the following distance function

$$D_{te} = \sum_{LP_{tr} \in LP^s_{te}} \|d_{te} - x^*_{tr}\|_2 \tag{11}$$

where x^*_{tr} is the optimum solution of the training problem LP_{tr}.

Rule 5 is based on the observation that the optimal solutions of ξ-similar problems lie close to each other in F^s (shown in Figures 1 and 2). Thus, d_{te} that is closest to the optimal solutions of problems belonging to LP^s_{tr} is going to be equal or almost-equal to x^*_{te}. We minimize the L^2 norm instead of a ξ-similarity metric-based distance as we want to find an estimated solution that lies close to x^*_{te} in the Euclidean distance sense. ξ-similarity is, however, used within the distance function to assist in identifying a suitable choice of d_{te} among multiple feasible options. The overall predictor vector for the test LP problem, v_{te}, is then constructed analogous to (10) as $v_{te} = [d^T_{te} \quad c^T_{te}]^T$. All the steps for constructing v_{te} are stated formally in Algorithm 1.

Figure 3 depicts the geometric representation of a test LP problem that is ξ-similar to the second cluster but not to the first cluster of the ξ-similar LP problem space shown in Figure 2. The linearly independent active constraints that generate the optimum solution for the test problem are the perturbed forms of the active constraints of the ξ-similar training set problems. The above-mentioned rules enable us to correctly identify the active constraints as the relevant constraints, which makes d identical to the unknown optimum solution. Even though this is not necessarily true (otherwise there would not be any need for learning at all), the constructed d mostly yields feasible solutions that lie close to the optimal. The benefit of our construction procedure in obtaining low prediction error bounds is formalized in section 4 by utilizing the properties of LP problems, definitions of ξ-similarity, choice of the regression distance function, and well-known principles of linear algebra.

3.3 Boosting Tree Models

Given a procedure for transforming an LP problem into a form given by the predictor variable vector v that permits mapping of the problem data directly to the predicted optimal values of the decision

Input: A member set LP^s of ξ-similar LP problem space (F^s, C^s) with known optimal solutions x^* and basic solutions set at x^* of each of the LP problems, a test LP problem LP_{te} with data (A_{te}, b_{te}, c_{te}) that is not an element of any of the clusters of LP^s but belongs to (F^s, C^s), and similarity metric ξ.

Output: Predictor vector v_{te} that can be fed to a regression model to infer an approximately optimum solution of LP_{te}.

Initialize $P_{te} \leftarrow 0_{m_{te},n}$, where m_{te} and n are the number of rows and columns in the constraint matrix A_{te} respectively.
Initialize candidate set $P^c \leftarrow \phi$ and the cardinality of P^c, $N^c \leftarrow 0$.
Identify the ξ-similar LP problem cluster, $LP^s_{te} \subseteq LP^s$, for LP_{te} by checking whether LP_{te} and any element of each cluster of LP^s are ξ-similar using Definition 1. Stop as soon as ξ-similarity is found for the first time as all the ξ-similar problems are included in the same cluster.
Populate P^c and using the procedures described in Case 1 and Case 2 in section 3.2.1.
$h_{\min} \leftarrow \infty$.
for $k = 1$ to N^c **do**
  $d_k \leftarrow [d^T_{B,k} \quad d^T_{N,k}]^T$, where $d_{B,k} \leftarrow [(P^T_k A_{te})^{-1} P^T_k b_{te}]$ and $d_{N,k} \leftarrow 0_{n-m_{te},1}$.
  if $d_{B,k} \geq 0$ **then**
    $h_k \leftarrow \sum_{LP_{tr} \in LP^s_{te}} \|d_k - x^*_{tr}\|_2$, where x^*_{tr} is the optimum solution of training LP Problem LP_{tr}.
    if $h_k < h_{\min}$ **then**
      $h_{\min} \leftarrow h_k$ and $d_{te} \leftarrow d_k$.
    end if
  end if
end for
Construct $v_{te} \leftarrow [d^T_{te} \quad c^T_{te}]^T$.

Algorithm 1: Test LP Problem Predictor Variable Vector Construction

variables, we now learn regression models for a set of ξ-similar transformed LP problems. We modify the gradient boosting tree algorithm (Hastie et al., 2008) to learn a regressor function separately for every decision variable. Boosting trees are chosen as they provide adequate representational power due to the generalized additive form, have proved successful in several practical large-scale applications, and give a compact model that permits efficient querying operations. The boosting tree algorithm iteratively constructs a pre-defined number of regression trees, where each regression tree performs binary partitioning of the v-space to create mutually exclusive regions at the leaf nodes. Each region models the response variable using a constant value. At every partitioning step, a particular predictor variable is chosen as the splitting variable, such that creating two child regions using a splitting variable value greedily minimizes the error in estimating the response variable y based on the currently constructed model.

The regressor function is then represented as a sum of W binary trees $T(v; \Theta_w)$, each of which is given by

$$T(v; \Theta_w) = \sum_{j=1}^{J} \gamma_{j,w} I(v \in R_{j,w})$$

where $\Theta_w = \{R_{j,w}, \gamma_{j,w}\}$ encodes the data of the w-th regression tree having terminal regions $R_{j,w}$ and modeling variables $\gamma_{j,w}$, $j = 1, \ldots, J$. The indicator function I is defined as

$$I(v \in R_{j,w}) = \begin{cases} 1 & \text{if } v \in R_{j,w}, \\ 0 & \text{otherwise.} \end{cases}$$

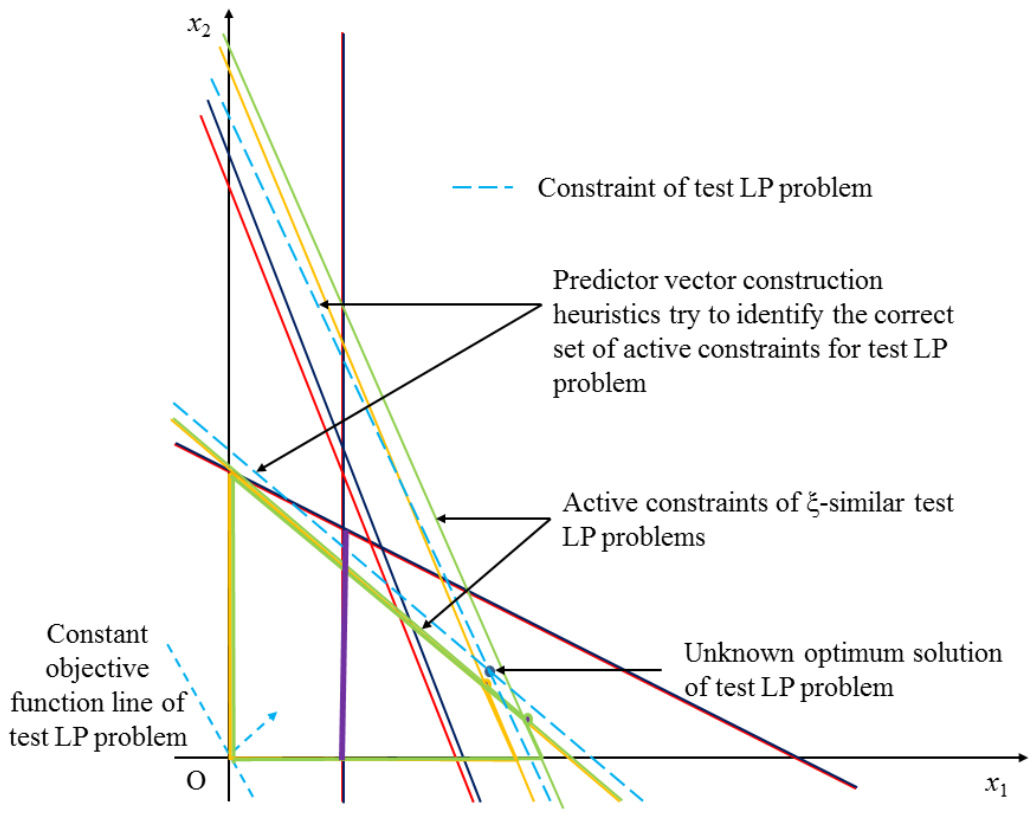

Figure 3: Schematic illustration of identifying relevant constraints to generate a suitable d vector for predicting the optimum solution of a test LP problem that belongs to a ξ-similar LP problem space whose member set comprises of the training set LP problems

The regressor function f_i for estimating the optimum value of any decision variable x_i is written in an iterative form as

$$f_{i,w}(v) = f_{i,w-1}(v) + \nu \sum_{j=1}^{J} \gamma_{j,w} I(v \in R_{j,w}), w = 1, \ldots, W \tag{12}$$

where $\nu \in (0, 1)$ is the shrinkage parameter that controls the learning rate; the modeling variable is given by

$$\gamma_{j,w} = \arg\min_{\gamma} \sum_{v_k \in R_{j,w}} L(y_k, f_{i,w-1}(v_k) + \gamma); \quad y_k = x^*_{i,k} \quad .$$

Here, $L(y, f(v))$ denotes any of the standard loss functions, such as L_2 or the squared-error loss, L_1 or the absolute-error loss, or the more robust Huber loss L_H (Hastie et al., 2008)

$$L_H = \begin{cases} [y - f(v)]^2, & \text{if } |y - f(v)| \leq \delta, \\ 2\delta(|y - f(v)| - \delta^2), & \text{otherwise.} \end{cases} \tag{13}$$

Further details on how to obtain the child nodes from the internal parent nodes are provided in Appendix A. The Boost-LP Inference algorithm simply selects the γ value stored in the terminal region corresponding to the location of the point defined by the predictor vector of the test LP problem as the estimated optimum value of x_i.

In the Boost-LP Learning algorithm, we restrict ourselves to modeling only the response variables of ξ-similar LP problems within any terminal region $R_{j,w}$. We also consider a modified form of the standard loss functions that heavily penalizes selection of any $\gamma_{j,w}$ value that lies outside the common feasible region of all the ξ-similar LP problems for which the predictor vectors v lie in $R_{j,w}$. We refer to this modified loss function as the *penalization loss* L_p and represent it as

$$L_p = L(y, f(v)) + \tau I(y \notin P_c). \tag{14}$$

Here, τ is a very large positive number and P_c is the common (intersecting) feasible region of all the ξ-similar LP problems whose $v \in R_{j,w}$. Although this modification cannot guarantee generation of feasible solution for any test LP problem that has a bounded feasible region, it significantly increases the possibility of doing so (shown in section 5.2).

Table 1: Cross-validation error (in %) using different regression approaches.

Approach	Loss function			
	L_2	L_1	L_H	L_p
Ridge[4]	10.4	11.1	10.9	11.5
Polynomial	7.9	8.2	8.0	8.3
SVM + ridge[5]	10.5	11.1	10.8	11.4
SVM + polynomial	8.0	8.2	8.1	8.3
Nearest neighbor	4.9	5.2	5.1	5.4
Boosting tree	**1.9**	**2.2**	**2.1**	**2.3**

Table 1 presents a comparison of the average (taken over all the response variables) cross-validation error in predicting the solutions of all the relaxed LP problems in the BAB nodes of the ILP problem instances corresponding to a repeated scheduling problem described later in section 5.2. 16 ILP problem instances are used for training, and 4 new instances are used for testing purposes. The objective function and constraint matrix remain the same in all the problem instances, and only the constraint vector differs from one instance to the other.

The ridge, polynomial (quadratic), and support vector machine (SVM) regression methods did not utilize the similarity based clustering of the training set LP problems but followed an identical predictor vector generation procedure as the boosting tree model. On the other hand, the nearest neighbor approach used a variant of the standard k-nn estimation model with equal weights given to all the neighbors, wherein k was selected at inference time as the cardinality of the set of training LP problems that were ξ-similar to the test problem under consideration. Thus, it did not require any predictor vector construction and estimated the optimal solutions directly by averaging the optimal solutions of all the ξ-similar training set problems. Since our boosting tree regression outperforms

4. Even though, strictly speaking, ridge regression uses an L_2 loss function, we tested all the loss functions for the sake of consistent comparisons. L_2 regularization is used regardless of the loss function form.
5. The SVM + ridge approach uses a linear kernel.

all the other regression approaches regardless of the form of the loss function, we conclude that both similarity based clustering and generation of a compact predictor vector by leveraging the simplex property of LP problems are important. While using just clustering without problem structure exploitation yields better results than using problem structure without clustering, it often results in infeasible solutions as demonstrated later in section 5.1.

3.4 ILP Solution Approach

There are three salient characteristics of the Boost-LP Inference approach that uses the inferred solutions of the relaxed LP problems at the nodes of the BAB tree to solve the repeated ILP problem(s) which we are interested in. These characteristics are listed as follows:

- The LP solution absolute error bound (derived later in section 4.2) is added to the incumbent (current best) solution for pruning the BAB tree paths. Mathematically, it implies that we fathom any node where the estimated objective function z^e satisfies the condition $z^e \geq z_{inc} + \epsilon_{LP}$, where z_{inc} is the current best or the incumbent solution and ϵ_{LP} is the absolute error bound. This a conservative pruning process that ensures that no potentially promising path, which might yield a better solution, is discarded at the cost of exploring some paths which may result in sub-optimal solutions.

- Fractional solutions are rounded off to the nearest greater or lower-valued integers if they lie within certain user-defined threshold values. Thus, if $\lceil x_i^e \rceil - x_i^e \leq \delta_u$, we set $x_i^e = \lceil x_i^e \rceil$, and if $x_i^e - \lfloor x_i^e \rfloor \leq \delta_l$, we set $x_i^e = \lfloor x_i^e \rfloor$, where x_i^e is the estimated optimum value of the i-th component of the decision variable vector. This rounding is done to account for some of the estimation errors and prevent unnecessary branching from the BAB nodes.

- Many BAB tree nodes generate infeasible or unbounded solutions for the corresponding LP problems. Since the Boost-LP algorithm cannot directly detect such cases as it has no explicit notion of any feasible region, it treats a solution as infeasible or unbounded if the corresponding LP problem does not belong to (F^s, C^s) or if the estimated solution satisfies the condition $z^e < z_{min}^e - \epsilon_{LP}$. Here, z_{min}^e represents the estimated solution of the LP problem at the root node, which is theoretically the minimum possible value for all the LP problems in the BAB tree. As the algorithm cannot ensure that the estimated values lie within the feasible region for problems even where feasible and bounded solutions do exist, it checks whether the estimated solution satisfies all the constraints for the problems that do not satisfy the above-mentioned conditions. If constraint violations are detected, the predicted values are discarded and the LP problem is solved using any standard optimizer.

3.5 Overall Algorithms

Algorithm tables 2 and 3 present the Boost-LP Learning and Boost-LP Inference algorithms by summarizing the various steps described earlier in this section.

4. Theoretical Results

In this section, we present an absolute performance guarantee on the test LP problem objective function in the form of Theorem 12. This result is obtained by using the Cauchy-Schwarz inequality and

Input: A set of N feasible and bounded LP problems LP^s, where all the problems are expressed in the standard form and contain identical decision variables $\in \Re^n$, the optimal solutions x^* of all the problems, similarity metric ξ, threshold vector ζ, Huber loss function parameter δ, number of regression trees W, number of leaf nodes in each tree J, shrinkage parameter ν, and penalization loss parameter τ.

Output: A member set of ξ-similar LP problem space (F^s, C^s), a set of n regression functions that map from (F^s, C^s) to $\Re$, and the LP objective function absolute error bound ϵ_{LP}, or an error message that a ξ-similar LP problem set cannot be created.

Update LP^s and N by discarding the LP problems that do not form a ξ-similar LP problem set according to Definitions 1 and 2.
if $LP^s = \phi$ **then**
  Generate the error message and terminate.
else
  Create the ξ-similar LP problem space (F^s, C^s) by considering LP^s as the member set using Definition 3.
end if
for $k = 1$ to N **do**
  Generate the predictor vector $v_k = [(x_k^*)^T \quad c_k^T]^T$, where any problem $LP_k \in LP^s$ has data (A_k, b_k, c_k) and optimum solution x_k^*.
end for
for $i = 1$ to n **do**
  Sort $\{x_{i,1}^*, \ldots, x_{i,N}^*\}$ in non-decreasing order.
  for $k = 1$ to $N - 1$ **do**
    if $(x_{i,k+1}^* - x_{i,k}^*) > \zeta_i$ **then**
      Construct additional training set LP problems following the steps described in section 3.2.1, update LP^s, (F^s, C^s) and N accordingly, and construct v_k for the newly-added problems.
    end if
  end for
end for
for $i = 1$ to n **do**
  for $k = 1$ to N **do**
    Obtain the response variable $y_k = x_{i,k}^*$.
  end for
  Use $v_k, y_k \forall k$ and the parameters W, J, ν, δ, and τ to generate the regression function f_i that maps from (F^s, C^s) to $\Re$ following the procedure described in section 3.3. Also compute the absolute error bound for x_i as given by (28).
end for
Compute the LP objective function error bound ϵ_{LP} using (30).

Algorithm 2: Boost-LP Learning

the formula for the outer radius of the hyperbox created by boosting trees in the predictor variable vector (v) space. The hyperbox arrangement is formally stated in Lemma 8 and relies on a useful form of v that is specified in Lemma 5. The dimensions of the box are bounded by the prediction errors for the individual decision variables. This error is derived in Lemma 11 and depends on whether the variable is labeled as basic or non-basic. The error terms have four components. The first component is obtained in Lemma 10 based upon our choice of v that uses Rule 5. The second component represents the error introduced in predicting the test LP problem solution by using ξ-similar (not identical) training set LP problems, again using our particular form of v based on Rules 1-4; it is specified in Lemma 9. The third component depends on the complexity of the boosting tree

Input: A member set of ξ-similar LP problem space (F^s, C^s), a set of n regression functions that map from (F^s, C^s) to $\Re$, the LP objective function absolute error bound ϵ_{LP}, similarity metric ξ, threshold parameters for rounding off fractional values δ_u and δ_l, and a repeated ILP problem with unknown optimum solution that has not been used to generate the member set.

Output: An approximate solution of the given ILP problem.

Follow the BAB algorithm to construct a tree, where each node represents a relaxed LP version of the given ILP problem, branch from existing nodes to create child nodes, and fathom nodes when the paths are guaranteed to not yield any solution that is better than the incumbent.
Instead of solving any of the relaxed LP problems optimally using a standard solver, set the estimated solution vector $x^e_{te} \leftarrow \phi$ and estimated objective function $z^e_{te} \leftarrow -\infty$.
while $z^e_{te} = -\infty$ **do**
  if LP_{te} does not belong to (F^s, C^s) from Definition 4 **then**
    $z^e \leftarrow \infty$.
  else
    Construct the predictor variable vector v_{te} for the corresponding LP problem LP_{te} with data (A_{te}, b_{te}, c_{te}) using Algorithm 1 and the similarity metric ξ.
    for $i = 1$ to n **do**
      Use the regression function f_i and v_{te} to infer an approximate solution $x^e_{i,te}$.
    end for
    $z^e_{te} \leftarrow c^T_{te} x^e_{te}$.
  end if
end while
Use the characteristics described in section 3.4 and the input parameters ϵ_{LP}, δ_u, and δ_l to modify the bounding step in the BAB algorithm, make certain fractional values integers, and handle infeasiblity of estimated solutions for feasible and bounded LP problems.

Algorithm 3: Boost-LP Inference

model and the last component is the user-specified vector ζ that is discussed in section 3.2. Both of these are used in Lemma 11. We begin by defining and presenting the formula for outer radius of a hyperbox.

Definition 6 *The* outer j-radius R_j *of a p-dimensional convex polytope P is defined as the radius of the smallest j-dimensional ball containing the projection of P onto all the j-dimensional subspaces, whereby the value of R_j is obtained by minimizing over all the subspaces. Note that R_p is the usual outer radius or circumradius.*

Remark 7 *Let B be a p-dimensional box (or p-box) in Euclidean space $\mathbb{E}^p$ with dimensions $2r_1 \times 2r_2 \times \ldots \times 2r_p$, such that $\{x \in \mathbb{E}^p \mid -r_i \leq x_i \leq r_i, i = 1, \ldots, p\}$. It can be assumed without any loss of generality that $0 < r_1 \leq r_2 \leq \ldots \leq r_p$. Then R_j is simply given by Brandenberg (2005) as*

$$R_j = \sqrt{r_1^2 + \ldots + r_j^2} \quad . \tag{15}$$

4.1 Relations between Predictor Variable Vectors and Optimal Solutions

We first present a lemma that formalizes the hyperbox arrangement generated by boosting trees. This is used in deriving the performance guarantees for the test LP problem decision variables and objective function in Lemma 11 and Theorem 12 respectively.

Lemma 8 *Boosting trees transform the LP problems present in the training data set to an arrangement of p-dimensional, axis-aligned boxes.*

Proof

We have

$$0 \leq d_k < \infty, 1 \leq k \leq N \tag{16}$$

where N is the training data set size. The LHS inequality in (16) follows from Lemma 5 and (3), while the RHS inequality is obtained from Lemma 5 and the fact that the decision variables cannot assume infinite values that result in unbounded LP solutions (from (1)). Thus, the irregularly-shaped convex polytopes that represent the feasible regions of the LP problems are transformed into axis-aligned boxes in the completely positive 2^n-ant of the n-dimensional d-space.

The predictor variable vector v of the boosting tree also includes the components of the objective function vector c. As both the minimum and maximum values of c_i must be finite $\forall i$ (infinite values again lead to unbounded LP solution as given by (1)), c forms another axis-aligned box of dimensionality n. Thus, the overall v space is also a box B of higher dimensionality $p = 2n$. Since each node in a regressor tree is formed by performing binary partitioning of the v space, every terminal tree region corresponds to a p-dimensional box $B' \subseteq B$, thereby creating an arrangement of boxes. ■

We now present a lemma that provides an upper bound on the absolute difference between any component of the d vector of the test LP problem and the optimum value of the corresponding decision variable of a ξ-similar LP problem that belongs to the training data set. This result is used Lemma 11 to derive the performance bounds for the test problem decision variables.

Lemma 9 *A ξ-similar training LP problem (LP_{tr}) exists for any test LP problem (LP_{te}) such that the maximum value of $\left|d_{i,te} - x^*_{i,tr}\right|, 1 \leq i \leq n$, is bounded by a constant that depends on the similarity vector ξ_A, the data of the training set LP problems, and whether $x_{i,tr}$ and $x_{i,te}$ are basic or non-basic.*

Proof

LP_{te} must be ξ-similar to at least one of the training set LP problems, as otherwise it does not belong to the ξ-similar LP problem space whose member set is given by the training set (from Definition 4), and, hence, is not regarded as a test problem at all. Let us denote that ξ-similar training problem by LP_{tr}[6]. From (5), using the property $\|a_{i,te} - a_{i,tr}\|_2 \geq |\|a_{i,te}\|_2 - \|a_{i,tr}\|_2|$, we have

$$\frac{|\|a_{i,te}\|_2 - \|a_{i,tr}\|_2|}{\|a_{i,te}\|_2 + \|a_{i,tr}\|_2} \leq \xi_{i,A} \quad .$$

Therefore, if $\|a_{i,te}\|_2 \geq \|a_{i,tr}\|_2$, we get

$$\begin{aligned} &\|a_{i,te}\|_2 - \|a_{i,tr}\|_2 \leq \xi_{i,A}(\|a_{i,te}\|_2 + \|a_{i,tr}\|_2) \\ \Rightarrow &\|a_{i,te}\|_2(1 - \xi_{i,A}) \leq \|a_{i,tr}\|_2(1 + \xi_{i,A}) \\ \Rightarrow &\frac{\|a_{i,te}\|_2}{\|a_{i,tr}\|_2} \leq \frac{1 + \xi_{i,A}}{1 - \xi_{i,A}} = \xi_{i,r} \quad . \end{aligned} \tag{17}$$

Alternatively, if $\|a_{i,te}\|_2 < \|a_{i,tr}\|_2$, we have $\frac{\|a_{i,te}\|_2}{\|a_{i,tr}\|_2} \geq \frac{1}{\xi_{i,r}}$. As $\xi_{i,r}$ is always greater than 1 (since $\xi_{i,A} \in (0,1]$ in Definition 1), (17) holds true regardless of whether $\|a_{i,te}\|_2$ is greater than or less than $\|a_{i,tr}\|_2$.

We can then re-write (17) as

$$\begin{aligned}\sum_j (a^j_{i,te})^2 &\leq \xi^2_{i,r} \sum_j (a^j_{i,tr})^2 \\ &= \sum_j (a^j_{i,tr} + \epsilon_{i,tr})^2 \end{aligned} \tag{18}$$

where $\epsilon_{i,tr}$ is a small number that is given by one of the solutions of the quadratic equation $\sum_j (a^j_{i,tr} + \epsilon_{i,tr})^2 - \xi^2_{i,r} \sum_j (a^j_{i,tr})^2 = 0$ (see Appendix B for greater details). Now, (18) is satisfied if $a^j_{i,te} \leq a^j_{i,tr} + \epsilon_{i,tr} \forall i, j$. So, by selecting $\epsilon_{tr} = \max\{\epsilon_{1,tr}, \ldots, \epsilon_{n,tr}\}$ and a non-negative matrix $F \in \Re^{m_{te},n}$ suitably with no element greater than 1, we get

$$A_{te} = A_{tr} + \epsilon_{tr} F \tag{19}$$

where only the matched set of constraints are considered in A_{tr} when $m_{tr} > m_{te}$. By replacing the i-th column of A_{te} and A_{tr} with b_{te} and b_{tr}, and denoting the resulting matrices by A^i_{te} and A^i_{tr} respectively, we have an analogous expression given by

$$A^i_{te} = A^i_{tr} + \epsilon^i_{tr} F^i \tag{20}$$

where $\epsilon^i_{tr} = \max\{\epsilon_{1,tr}, \ldots, \epsilon_{i-1,tr}, \epsilon_{n+1,tr}, \epsilon_{i+1,tr} \ldots, \epsilon_{n,tr}\}$. Note that (19) and (20) are also valid for truncated forms of the constraint matrices that only contain the columns corresponding to the basic solutions.

The deviation between $d_{i,te}$ and $x^*_{i,tr}$ can be decomposed into four different cases: first, when the corresponding decision variable in both $d_{i,te}$ and $x^*_{i,tr}$ is either selected by the Rules in section 3.2 or known to be a basic variable, second, when the decision variable is considered to be basic in LP_{te} but is actually non-basic in LP_{tr}, third, when the decision variable is considered to be non-basic in LP_{te} but is actually basic in LP_{tr}, and fourth, when the decision variable is non-basic in both LP_{te} and LP_{tr}. The absolute deviation is exactly given by $x^*_{i,tr}$ in the third case as $d_{i,te}$ is equal to 0, and by 0 in the last case as both $d_{i,te}$ and $x^*_{i,tr}$ are equal to 0. Let us now derive the expressions for the first two cases.

For the first case, we have

$$\left|d_{i,te} - x^*_{i,tr}\right| = \left|\frac{\det(A^i_{H,te})}{\det(A_{H,te})} - x^*_{i,tr}\right| \quad \text{from Cramer's rule} \tag{21}$$

where $A_{H,te}$ denotes the rule-determined truncated A matrix of LP_{te} that gives the non-zero (expected to be basic at optimum x^*_{te}) components of d_{te}. Combining (21) with (19) and (20), we get

$$\left|d_{i,te} - x^*_{i,tr}\right| = \left|\frac{\det(A^i_{H,tr} + \epsilon^i_{tr} F^i_H)}{\det(A_{H,tr} + \epsilon_{tr} F_H)} - x^*_{i,tr}\right|$$

6. If multiple ξ-similar training problems exist, then this lemma holds true for each one of them.

where F_H^i and F_H represent the truncated forms of F^i and F respectively that only contain the columns corresponding to the rule-determined basic variables. Expanding the determinants using the property that ϵ_{tr}^i and ϵ_{tr} are small numbers, we get

$$\begin{aligned}\left|d_{i,te} - x_{i,tr}^*\right| &= \left|\frac{\det(A_{H,tr}^i) + \det(A_{H,tr}^i)\text{tr}((A_{H,tr}^i)^{-1}F_H^i)\epsilon_{tr}^i + O((\epsilon_{tr}^i)^2)}{\det(A_{H,tr}) + \det(A_{H,tr})\text{tr}((A_{H,tr})^{-1}F_H)\epsilon_{tr} + O((\epsilon_{tr})^2)} - x_{i,tr}^*\right| \\ &\leq \left|\frac{\det(A_{H,tr}^i) + \det(A_{H,tr}^i)\text{tr}((A_{H,tr}^i)^{-1}F_H^i)\epsilon_{tr}^i}{\det(A_{H,tr}) + \det(A_{H,tr})\text{tr}((A_{H,tr})^{-1}F_H)\epsilon_{tr}} - x_{i,tr}^*\right| \text{[7]} \\ &= \left|\left(\frac{\det(A_{H,tr}^i)}{\det(A_{H,tr})}\right)\left(\frac{1 + \text{tr}((A_{H,tr}^i)^{-1}F_H^i)\epsilon_{tr}^i}{1 + \text{tr}((A_{H,tr})^{-1}F_H)\epsilon_{tr}}\right) - x_{i,tr}^*\right| \quad .\end{aligned}$$

Expanding the trace of the product of two matrices as the sum of all the elements constituting the Hadamard product, we have

$$\left|d_{i,te} - x_{i,tr}^*\right| \leq \left|\left(\frac{\det(A_{H,tr}^i)}{\det(A_{H,tr})}\right)\left(\frac{1 + \sum_{k,l} a_{l,H,tr}^{i,k} f_{k,H}^{i,l}\epsilon_{tr}^i}{1 + \sum_{k,l} a_{l,H,tr}^{k} f_{k,H}^{l}\epsilon_{tr}}\right) - x_{i,tr}^*\right|$$

where $a_{l,H,tr}^{i,k}$ and $a_{l,H,tr}^{k}$ represent the elements in the kth row and lth column of matrices $A_{H,tr}^i$ and $A_{H,tr}$ respectively, and likewise, $f_{k,H}^{i,l}$ and $f_{k,H}^{l}$ denote the elements in the lth row and kth column of matrices F_H^i and F_H respectively. Therefore,

$$\begin{aligned}\left|d_{i,te} - x_{i,tr}^*\right| &\leq \max\left(\left[x_{i,tr}^{\max}\left(1 + \left|\sum_{k,l} a_{l,H,tr}^{i,k}\epsilon_{tr}^i\right|\right) - x_{i,tr}^*\right], \left[x_{i,tr}^* - x_{i,tr}^{\min}\left(\frac{1}{1 + \left|\sum_{k,l} a_{l,H,tr}^{k}\epsilon_{tr}\right|}\right)\right]\right) \\ &= G_{i,tr}(\xi_{i,A}, A_{tr}, b_{tr}, c_{tr}) \qquad (22)\end{aligned}$$

as $f_{k,H}^{i,l}, f_{k,H}^{l} \in [0,1] \forall l,k$; $x_{i,tr}^{\max}$ and $x_{i,tr}^{\min}$ are the maximum and minimum basic feasible solutions respectively.

Finally, for the second case, we have

$$\begin{aligned}\left|d_{i,te} - x_{i,tr}^*\right| &\leq \left[x_{i,tr}^{\max}\left(1 + \left|\sum_{k,l} a_{l,H,tr}^{i,k}\epsilon_{tr}^i\right|\right)\right] \quad \text{as } x_{i,tr}^* = 0 \text{ and } x \geq 0 \text{ is the feasible region} \\ &= H_{i,tr}(\xi_{i,A}, A_{tr}, b_{tr}) \quad . \qquad (23)\end{aligned}$$

This concludes the proof of the lemma. ■

We now use the expressions derived in the previous lemma along with Rule 5 to estimate the number of differently labeled basic and non-basic decision variables since that will govern the test LP problem objective function estimation error in Theorem 12.

7. This inequality holds as both $\det(A_{H,tr}^i) + \det(A_{H,tr}^i)\text{tr}((A_{H,tr}^i)^{-1}F_H^i)\epsilon_{tr}^i \leq \det(A_{H,tr}) + \det(A_{H,tr})\text{tr}((A_{H,tr})^{-1}F_H)\epsilon_{tr}$ and $O((\epsilon_{tr}^i)^2) \leq O((\epsilon_{tr})^2)$.

Lemma 10 *Application of Rule 5 yields a minimum number of decision variables $n_{d,B}^{\min}$ that are labeled as basic in the test problem LP_{te} and non-basic in a ξ-similar training problem LP_{tr} with a worst-case value of* $\arg\min_{n_{d,B}}[\sum_{i_1=1}^{m_{te}-n_{d,B}}(G_{i_1,\max})^2+\sum_{i_2=1}^{n_{d,B}}(H_{i_2,\max})^2+\sum_{i_3=1}^{n_{d,B}}(x^*_{i_3,\max})^2]$. *Here m_{te} is the number of constraints in the test problem LP_{te}, LP_{te}^s is the set of training set problems that are ξ-similar to LP_{te}, $G_{i_1,\max}=\max\{G_{i_1,tr}|x_{i_1}\in x_{B,te},x_{B,tr};LP_{tr}\in LP_{te}^s\}$, $H_{i_2,\max}=\max\{H_{i_2,tr}|x_{i,2}\in x_{B,te},x_{NB,tr};LP_{tr}\in LP_{te}^s\}$, $G_{i_1,tr}$ and $H_{i_2,tr}$ are obtained from Lemma 9, and $x^*_{i_3,\max}=\max\{x^*_{i_3,tr}|x_{i,3}\in x_{NB,te},x_{B,tr};LP_{tr}\in LP_{te}^s\}$. $x_{B,te}$ ($x_{B,tr}$) and $x_{NB,te}$ ($x_{NB,tr}$) denote the set of labeled (known) basic and non-basic variables in LP_{te} (LP_{tr}) respectively.*

Proof

According to Rule 5, we select d_{te} that minimizes the sum of L_2 distances to the optimal solutions of all the ξ-similar training set LP problems present in the set LP_{te}^s. This distance D_{te} is given by (11) to be

$$\begin{aligned} D_{te} &= \sum_{LP_{tr}\in LP_{te}^s}\|d_{te}-x^*_{tr}\|_2 = \sum_{LP_{tr}\in LP_{te}^s}\left[\sum_{i=1}^{n}\left|d_{i,te}-x^*_{i,tr}\right|^2\right]^{\frac{1}{2}} \\ &\leq \sum_{LP_{tr}\in LP_{te}^s}\left[\sum_{i_1=1}^{n_{s,B}}(G_{i_1,tr})^2+\sum_{i_2=1}^{n_{d,B}}(H_{i_2,tr})^2+\sum_{i_3=1}^{n_{d,NB}}\left(x^*_{i_3,tr}\right)^2\right]^{\frac{1}{2}} \quad \text{from Lemma 9} \end{aligned} \tag{24}$$

such that $n_{s,B}$, and $n_{d,NB}$ denote the number of cases where the decision variables are basic in both LP_{te} and LP_{tr} and non-basic in LP_{te} but basic in LP_{tr} respectively. Note here that $n_{d,NB}=n_{d,B}$ as the sum of basic and non-basic variables is fixed and equal to n for all the LP problems. Using this observation and the fact that the total number of basic variables in LP_{te} is equal to the number of constraints m_{te}, we re-write (24) as

$$D \leq N_{te}^s\left[\sum_{i_1=1}^{m_{te}-n_{d,B}}(G_{i_1,\max})^2+\sum_{i_2=1}^{n_{d,B}}(H_{i_2,\max})^2+\sum_{i_3=1}^{n_{d,B}}\left(x^*_{i_3,\max}\right)^2\right]^{\frac{1}{2}} \tag{25}$$

where N_{te}^s is the cardinality of LP_{te}^s. Thus, $n_{d,B}$ is the only variable parameter, and in the worst case, Rule 5 minimizes the distance given by the RHS of (25) with respect to $n_{d,B}$. This choice of $n_{d,B}$ is referred to as $n_{d,B}^{\min}$, which is then given by

$$n_{d,B}^{\min}=\arg\min_{n_{d,B}}\left[\sum_{i_1=1}^{m_{te}-n_{d,B}}(G_{i_1,\max})^2+\sum_{i_2=1}^{n_{d,B}}(H_{i_2,\max})^2+\sum_{i_3=1}^{n_{d,B}}\left(x^*_{i_3,\max}\right)^2\right] \quad . \tag{26}$$

■

4.2 Performance Bounds

We now combine the results derived in the previous lemmas with some properties of boosting trees to derive performance bounds for the test LP problem decision variables.

Lemma 11 *If x_i^e denotes the predicted value of x_i^* for any test LP problem LP_{te}, then $|x_i^* - x_i^e|$ is bounded by $G_{i,\max} + (N - WJ + 1)\zeta_i$ if x_i is labeled as basic when it is also basic in a ξ-similar training set problem LP_{tr}, $H_{i,\max} + (N - WJ + 1)\zeta_i$ if labeled as basic when it is non-basic in LP_{tr}, 0 if labeled as non-basic when it is also non-basic in LP_{tr}, and $(N-1)\zeta_i + x_{i,\min}^*$ if labeled as non-basic when it is basic in LP_{tr}. Here, W and J denote the number of regression trees and the number of leaf nodes in each regression tree respectively, they are chosen such that $N \geq WJ$, $G_{i,\max}$ and $H_{i,\max}$ are obtained from Lemma 9 and Lemma 10, ζ_i is defined in section 3.2.1, and $x_{i,\min}^* = \min\{x_{i,1}^*, \ldots, x_{i,N}^*\}$.*

Proof

From (12), it is evident that the maximum error in predicting any x_i^* arises when the corresponding v vector lies in a region $R_{j,w}^{crit}$ such that the worst-case error in predicting the response variable by a constant value $\gamma_{j,w}^{crit}$ is the highest among all the tree regions. Mathematically,

$$|x_i^* - x_i^e| = |y^* - f_{i,W}(v)| \leq |y^* - \gamma_{j,w}^{crit}| \quad .$$

Clearly, at least one training data point needs to be present in every tree region. Since the total number of regions is equal to WJ, in the extreme case it may so happen that $(WJ - 1)$ regions all have one data point and one region has the remaining $(N - WJ + 1)$ points. Now, the target value for any training set problem LP_k can be either equal to $(y_k - f_{i,w}(v_k))$ when the negative gradient of the L_2 loss functional is taken, or it can be equal to ± 1 and $\pm\delta$ when the L_1 and Huber loss functional are used respectively.

Thus, child tree regions are iteratively created to cluster the response variable based on similar deviations from predictions ($\gamma_{j,w}$), or similarities in terms of positive and negative deviations from the predicted values. Hence, it can be concluded that boosting tree always partitions the p-dimensional v-space into boxes (see Lemma 8) such that the corresponding response variable values are naturally sorted, although the exact arrangement as well as the constant predicted value inside a box varies depending upon the nature of the target values. Without any loss of generality, let us assume that this sorting is in non-decreasing order.

So, if we have a region containing $N_{j,w} = (N - WJ + 1)$ data points, the algorithm models $y_l, \ldots, y_{l+N-WJ}$ response variable values by a single $\gamma_{j,w}$, where $1 \leq l \leq WJ$, such that $y_l \leq \ldots \leq y_{l+N-WJ}$. From Lemma 5, as $d_{i,l} = y_l$ for any value of l, we have a lower bound on the value of $\gamma_{j,w}^{crit}$ given by $d_{i,l}$, and an upper bound given by $d_{i,l+N-WJ}$.

On the other hand, the true (optimum) value of the response variable for the test problem LP_{te} is upper bounded by the maximum value of $d_{i,te}$ as d_{te} is obtained by solving the system of linear equations given by $A_{H,te}x = b_{te}$, where $A_{H,te}$ contains the m_{te} columns of A_{te} that correspond to the basic variables as determined by Rules 1-5. From Lemma 9, there exists a ξ-similar training LP problem LP_{tr} with optimum solution x_{tr}^*, which sets the maximum value of $d_{i,te}$, given by $d_{i,te}^{\max}$ such that

$$y^* \leq d_{i,te}^{\max} = \begin{cases} G_{i,tr} + x_{i,tr}^* & \text{if } x_i \text{ is basic in both } LP_{te} \text{ and } LP_{tr} \\ H_{i,tr} + x_{i,tr}^* & \text{if } x_i \text{ is basic in } LP_{te} \text{ but non-basic in } LP_{tr} \\ 0 & \text{if } x_i \text{ is non-basic in both } LP_{te} \text{ and } LP_{tr} \\ 0 & \text{if } x_i \text{ is non-basic in } LP_{te} \text{ but basic in } LP_{tr} \end{cases} \quad .$$

Therefore, for the first case,

$$\begin{aligned}\left|y^* - \gamma_{j,w}^{crit}\right| &\leq G_{i,tr} + x_{i,tr}^* - d_{i,l} \\ &= G_{i,tr} + d_{i,tr} - d_{i,l} \quad \text{from Lemma 5} \\ &\leq G_{i,\max} + d_{i,l+N-WJ} - d_{i,l} \quad .\end{aligned}$$

Since, $|d_{i,j+1} - d_{i,j}| \leq \zeta_i, 1 \leq j \leq N-1$, we have,

$$\left|y^* - \gamma_{j,w}^{crit}\right| \leq G_{i,\max} + (N - WJ + 1)\zeta_i \quad . \tag{27}$$

The result for the second case follows exactly as before. For the third case, $d_{i,j} = 0, l \leq j \leq l+N-WJ$, and the deviation between true and estimated optimum is always equal to 0. However, for the final case, we have,

$$\begin{aligned}\left|y^* - \gamma_{j,w}^{crit}\right| &= \gamma_{j,w}^{crit} \quad \text{as } y^* = 0 \text{ for non-basic } x_i \text{ in } LP_{te} \\ &\leq d_{i,l+N-WJ} \\ &\leq d_{i,N} \quad \text{substituting maximum value of } l \text{ and using motononic arrangement of } d_i \\ &= (d_{i,N} - d_{i,1}) + d_{i,1} \\ &\leq (N-1)\zeta_i + x_{i,\min}^* \quad . \end{aligned} \tag{28}$$

Thus, the lemma follows. ■

Since $G_{i,tr} \leq H_{i,tr}$ for any training problem LP_{tr} that is ξ-similar to the test problem LP_{te}, as expected, the error in estimating decision variables is less when it is labeled as a basic variable in both LP_{te} and LP_{tr} as compared to when it is labeled as basic in LP_{te} and non-basic in LP_{tr}. Similarly, the estimation error is less when the variable is labeled as non-basic in both the problems as compared to when it is labeled as non-basic in LP_{te} and basic in LP_{tr}. We finally use all the previous results to derive the performance guarantee for the test problem objective function.

Theorem 12 *If x^e denotes the predicted x^* for any test LP problem LP_{te} such that the objective function values are given by z^e and z^* respectively, then $|z^* - z^e| \leq \|c'\|_2[\sum_{i_1=1}^{m_{te}-n_{d,B}^{\min}}(G_{i_1,\max} + (N-WJ+1)\zeta_{i_1})^2 + \sum_{i_2=1}^{n_{d,B}^{\min}}(H_{i_2,\max} + (N-WJ+1)\zeta_{i_2})^2 + \sum_{i_3=1}^{n_{d,B}^{\min}}((N-1)\zeta_{i_3} + x_{i_3,\min}^*)^2]^{\frac{1}{2}}$. Here, W and J denote the number of regression trees and the number of leaf nodes in each regression tree respectively, and they are chosen such that $N \geq WJ$. m_{te} is the number of constraints in LP_{te}, c' is the objective function vector of LP_{te} that only contains the coefficients corresponding to the non-slack and non-surplus variables, and LP_{te}^s is the set of training set problems that are ξ-similar to LP_{te}, $x_{i_1} \in x'_{B,te}, x'_{B,tr}$, $x_{i_2} \in x'_{B,te}, x'_{NB,tr}$, $x_{i_3} \in x'_{NB,te}, x'_{B,tr}$, $LP_{tr} \in LP_{te}^s$, $x_{i_3,\min}^* = \min\{x_{i_3,1}^*, \ldots, x_{i_3,N}^*\}$, $G_{i_1,\max}$ and $H_{i_2,\max}$ are obtained from Lemma 9 and Lemma 10, $n_{d,B}^{\min}$ is derived in Lemma 10, and ζ is defined in section 3.2.1. $x'_{B,te}$ ($x'_{B,tr}$) and $x'_{NB,te}$ ($x'_{NB,tr}$) denote the set of non-slack and non-surplus variables that are labeled (known) as basic and non-basic in LP_{te} (LP_{tr}) respectively.*

Proof

The overall error in predicting the optimum z^* by z^e is given by

$$|z^* - z^e| = |c^T x^* - c^T x^e| = |c.(x^* - x^e)| = |c'.(x'^* - x'^e)|$$

as the coefficients of c corresponding to the slack and surplus variables are equal to 0. x'^* and x'^e denote the optimal and estimated set of non-slack and non-surplus variables of LP_{te} respectively. Using Cauchy-Schwarz inequality, the error is bounded by

$$|z^* - z^e| \leq \|c'\|_2 \|x'^* - x'^e\|_2 \quad . \tag{29}$$

Based upon Lemma 11, if we consider the maximum prediction error for all the components of x, then we find n number of p-boxes, each constrained only in terms of a particular d_i corresponding to x_i whose value is being predicted. Thus, the maximum error in predicting the entire vector x occurs when the generated v lies in $R^{crit}_{j,w}$ of each individual boosting tree model. So, if we take the intersection of all the critical regions in v space, we obtain a new p-box B^{crit} where all the components of d must lie within the ranges given by Lemma 11 and c is fixed.

From Definition 4, the outer n-diameter of B^{crit} provides an upper bound on the Euclidean distance between any two n-dimensional points (d in this case) in B^{crit}. Using this observation, we combine (15) with the results obtained in Lemma 10 and Lemma 11, and substitute the value for the 2nd L_2 norm in the RHS of (29) to have

$$|z^* - z^e| \leq 2\|c'\|_2 [\sum_{i_1=1}^{m_{te}-n^{\min}_{d,B}} \left(\frac{G_{i_1,\max} + (N - WJ + 1)\zeta_{i_1}}{2} \right)^2 + \sum_{i_2=1}^{n^{\min}_{d,B}} \left(\frac{H_{i_2,\max} + (N - WJ + 1)\zeta_{i_2}}{2} \right)^2 + \sum_{i_3=1}^{n^{\min}_{d,B}} \left(\frac{(N-1)\zeta_{i_3} + x^*_{i_3,\min}}{2} \right)^2]^{\frac{1}{2}} \quad .$$

Simplifying the expression, we obtain the final form as

$$|z^* - z^e| \leq \|c'\|_2 [\sum_{i_1=1}^{m_{te}-n^{\min}_{d,B}} (G_{i_1,\max} + (N - WJ + 1)\zeta_{i_1})^2 + \sum_{i_2=1}^{n^{\min}_{d,B}} (H_{i_2,\max} + (N - WJ + 1)\zeta_{i_2})^2 + \sum_{i_3=1}^{n^{\min}_{d,B}} \left((N-1)\zeta_{i_3} + x^*_{i_3,\min}\right)^2]^{\frac{1}{2}} \quad . \tag{30}$$

■

Clearly, this error bound given by (30) is a function of the training set problem data, extent of similarity between the test and training set problems, our choice of the boosting tree model, and the level of discretization adopted to generate the training points. It also depends on the relative number of decision variables and constraints as a higher difference between the two means that there is a greater possibility of labeling basic variables differently between the test problem and ξ-similar training set problems. We use a conservative estimate of the error bound during pruning of BAB tree nodes to solve the overall ILP problem as mentioned in section 3.4 and report the conservative

values for the various problem domains in section 5. The estimate is based on the worst-case values of $\|c\|_2$ and m_{te} for a given problem domain, and the maximum error due to variation in the selection of the cluster of training set problems that are ξ-similar to LP_{te}.

We also infer that the performance guarantee can be improved in two ways for a given initial training set. The guarantee can be improved by building a more complex boosting tree model with more regression trees (W), or terminal regions in a tree (J), or both. However, using a complex model increases the inference time. The other alternative is to select a smaller value of $\zeta_i \forall i$, which reduces the values of the terms $(N - WJ + 1)\zeta_i$ and $(N - 1)\zeta_i$ even though N increases, such that the overall LP prediction error is always less than a certain pre-defined value up to a certain limit defined by $G_{i,\max}$, $H_{i,\max}$, $x^*_{i,\min}$, and m_{te}. Although this increases the learning time, inference time remains unaffected.

5. Experimental Results

We evaluated the performance of the Boost-LP algorithm in four representative problem settings: a benchmark library, a scheduling domain, a vehicle routing domain, and a vehicle control domain consisting of multiple problems taken from the robotics literature. The results were obtained on a Quad-Core Intel Xeon CPU, with a 2.93 GHz processor and 3.8 GB of RAM, in Ubuntu 9.0.4 OS, using C++ as the programming language, and COIN-OR Cbc version 2.0 and IBM ILOG CPLEX Optimization Solver Academic Edition version 12.2 as the optimization solvers. The default versions of COIN-OR Cbc and CPLEX were always used, and no problem-specific heuristics or cutting planes were applied.

With respect to the algorithm parameters, the shrinkage factor, ν, was constantly taken as 0.1, δ used in the Huber loss function given by (13) was 0.1 times the maximum value of all the response variables, and the penalization loss function parameter τ was 1e6. We set the rounding parameters, δ_u and δ_l, to be 0.95 and 0.05 respectively. The similarity parameter ξ_c and all the components of the similarity vector ξ_A were chosen as 0.1. Unless otherwise stated, all the components of the threshold vector ζ were selected to be 0.25 times the range of the corresponding response variable in the training data set. All the parameter values were chosen by varying them uniformly within pre-defined intervals that were either obtained from the literature or assumed to span very wide ranges, and noting the best performance in predicting LP problem solutions using 10-fold cross-validation. The reported data are the average values taken over all the test problems. Standard errors are not presented as they are of the order of 0.1% of the average. We do not report the specfic training times for any of the problems as our aim is solve new instances of repeated ILP problems as efficiently as possible assuming that sufficient time is available for offline regression model learning. Nevertheless, it is worth mentioning that the training times are never more than 46 hours, and other than the hard MIPLIB and large multi-vehicle control problems, always between 2 to 15.5 hours.

In addition to COIN-OR Cbc and CPLEX solvers, we compared the performance of our algorithm against several alternative ILP solution approaches listed below. First, the average optimum solution of all the training set problems that are ξ-similar to the test problem was selected as a baseline approach to help in concluding whether the performance benefits are solely due to the use of similar training problems or depends on our regression method. Note that this baseline method is identical to the nearest neighbor approach used for the results reported in Table 1. Second, we adopted the slack method given in Engau et al. (2009) for warm starting an interior-point LP solver at the BAB tree nodes using, (a) the training set problem that is most ξ-similar (closest) to the test

problem, and (b) the parent node in the BAB tree (same strategy followed in Benson and Shanno (2007)), as the warm start point respectively. Third, for randomized rounding, we used a simple approach of rounding any binary decision variable to 1 with a probability equal to its fractional value x_i^*, and 0 with a probability of $1 - x_i^*$. Fourth, the exact multiparametric ILP algorithm was taken from Dua and Pistikopoulos (2000), which used an iterative two-stage decomposition process of first solving a multiparametric LP problem exactly by fixing the binary variables, and then solving an ILP sub-problem by treating the parametric vector (θ) components as free variables. The bounds for the components of θ were chosen to match with the bounds of the constraint vector components present in the training set LP problems. The average values of all the constraint vector components were used to construct the constraint vector b for the multiparametric ILP problem. And last, for the approximate multiparametric method, we used the same overall algorithm, but replaced the exact multiparametric LP solution approach with an approximate one described as Algorithm 3.2 in Filippi (2004). The CPLEX solver was used to construct the BAB trees and solve the LP problems (if necessary) in all the alternative approaches. Comparisons with the nearest neighbor approach, as described in Section 3.3, were also performed for the benchmark problems.

5.1 Benchmark ILP Library

The first evaluation domain comprised of an electronically available benchmark library of real-world ILP problems, known as the MIPLIB. This library is popularly used for comparing any new optimizer with current state-of-the-art performance and provides a diverse set of problems ranging from flight scheduling to protein folding. In particular, we evaluated the performance of the Boost-LP algorithm on all the problems in the MIPLIB 2003 library for which the optimum solution of a problem instance could be computed in less than 5 hours by CPLEX. Specifications for each of the problems, namely the number of constraints, number of decision variables, number of integer and binary variables, objective function value, and the type of constraints (set packing, set cover, knapsack etc.) can be found in `http://miplib.zib.de/miplib2003/index.php`. Every problem is classified into one of the following three types: *easy*, *hard*, and *unknown solution*. We refer to the types as MIPLIB-Easy, MIPLIB-Hard, and MIPLIB-Unknown respectively. Hard problem refers to one that cannot be solved by any known method in less than an hour. Further details on the original sources of the problems and the structures of the constraint matrices are available in Achterberg et al. (2006).

For the sake of presentation brevity, we present sample results for three representative MIPLIB-Easy and MIPLIB-Hard problems. We perturbed the constraint vector and the objective function values without modifying the constraint matrix to generate 10 instances for each of the ILP problems. In each problem instance, 10% of the constraints and objective function coefficients were randomly chosen for perturbation, and an entirely different set of constraints and objective function coefficients were selected to generate the next problem instance. 8 of these instances were used for training and the remaining 2 for testing.

Our approach successfully estimates the solutions of the MIPLIB-Easy problems with low approximation errors of less than 4.0% in reasonably short time durations even with a simple boosting tree model consisting of 500 trees and 8 leaf nodes per tree; this is shown in Table 2. A more complex model comprising of 1000 trees and 16 leaf nodes per tree is required to obtain less than 5.0% approximation errors for the MIPLIB-Hard problems after consuming significantly more computation time. The solution estimation errors are much higher (greater than 12.5%), whereas the

inference times are much lower if simpler models consisting of 500 trees and 8 leaf nodes per tree are used instead. Both of these results are shown in Table 3. The theoretically predicted LP solution bounds (Theoretical LP solution error) are always less than twice that of the experimentally observed values (Observed LP solution error), showing that the bounds are reasonably tight.

The problems where the Boost-LP algorithm performs well share the common characteristic that the number of decision variables is far greater than the number of constraints when the problems are expressed in natural forms that do not contain any slack or surplus variables. This follows from the form of the error bound given by (30) and is briefly rationalized in the ensuing discussion in section 4.2. The coefficient matrices are also often irregular for the hard problems where our algorithm does not perform equally well. This is again expected from the terms inside the upper bounds presented in (22) and (23) that contribute to the overall estimation error.

Tables 2 and 3 show that, overall, our approach performs significantly better than all the other solution techniques. The computation times are markedly less (much more than an order of magnitude for the MIPLIB-Hard problems) than those for off-the-shelf exact solvers, warm start methods, and randomized rounding. The times for the warm start methods are, in fact, comparable to the exact solvers, which is consistent with the results reported in the literature (Benson and Shanno, 2007). The gain over randomized rounding is also expected as this metaheuristic approach solves the relaxed LP problems optimally and only derives some computational benefits over exact solvers by fathoming certain nodes using rounding operations. Note here that warm starts and randomized rounding have the advantage though of not requiring any computationally intensive pre-processing step to either construct the training sets, generate the regression models, or build polyhedral representations.

The computation times are also smaller than the baseline method and the two multiparametric techniques, although the differences are less pronounced in these cases. This reduction in computation time occurs due to fast predictor vector computation and boosting tree query operations. In contrast, a large number of LP problems need to be solved exactly for the baseline method as simply averaging the optimal solutions of ξ-similar training problems frequently leads to infeasible values even though the values lie in the vicinity of the true optimal. Loading memory-intensive data structures for the polyhedral decomposition of the parametric space in the multiparametric approaches is also quite time-consuming.

With respect to solution errors, the Boost-LP algorithm always performs better for the overall ILP problems, and marginally worse or somewhat better as compared to the other approaches for the relaxed LP problems. Although the baseline and warm start using closest training problem methods provide slightly better accuracy for the LP problems, their high computation times far outweight this marginal advantage.

5.2 Aircraft Carrier Deck Operations

The second problem is that of scheduling, where the objective function is to minimize the overall weighted time taken by a fleet of aircraft (both manned and unmanned) to complete a given sequence of tasks in naval carrier operations. The scheduling problem is an instance of a repeated ILP problem, where the problem data is expected to change from minute to minute without drastically altering the overall problem. Weights were assigned based upon the aircraft priorities, and

8. Newer version of CPLEX than the one available during the library creation solves this problem in less than an hour; nevertheless, computation time is still quite large and an order of magnitude more than the MIPLIB-Easy problems.

Table 2: Comparison of various ILP solution approaches on three MIPLIB-Easy problems where the Boost-LP algorithm outperforms all the other methods and provides an order of magnitude speed-up in computation time with less than 4.0% estimation error. The cross-validation errors are averaged over all the decision variables. The solution errors are computed for the objective functions. The metrics that always assume zero values, namely the solution errors for the exact approaches, LP solution error for randomized rounding, and the percentages of infeasible solutions for all the methods except the baseline, are omitted. The bold entries denote best performance among all the approaches (solution errors are only compared among the approximation methods) for the problem and metric under consideration.

Approach	Performance metric	Problem set		
		air04	air05	nw04
COIN-OR Cbc	Computation time (s)	126.58	40.36	11.75
CPLEX	Computation time (s)	25.46	23.35	2.65
Average solution of	Computation time (s)	12.91	12.11	2.04
ξ-similar training problems	Infeasible solutions (%)	34.6	29.8	33.3
(nearest neighbor	LP solution error (%)	**2.3**	**1.9**	2.0
estimation)	ILP solution error (%)	4.1	3.9	4.0
Warm starting	Computation time (s)	27.65	22.14	3.01
interior-point solver	LP solution error (%)	**2.3**	2.0	**1.7**
using closest training problem	ILP solution error (%)	**3.9**	3.7	3.8
Warm starting	Computation time (s)	21.26	18.59	2.18
interior-point solver	LP solution error (%)	3.4	3.3	3.3
using parent tree node	ILP solution error (%)	6.1	5.7	5.8
Randomized	Computation time (s)	16.78	15.45	1.51
rounding	ILP solution error (%)	8.2	8.0	7.4
Exact multiparametric	Computation time (s)	7.27	6.99	1.03
Approximate multiparametric	Computation time (s)	4.98	4.58	0.82
	LP solution error (%)	4.1	4.1	3.9
	ILP solution error (%)	7.4	7.3	6.9
Boost-LP Inference	Computation time (s)	**3.43**	**1.75**	**0.33**
	Cross-validation error (%)	1.8	1.7	1.6
	Theoretical LP solution error (%)	3.8	3.4	3.4
	Observed LP solution error (%)	2.4	2.1	2.1
	ILP solution error (%)	**3.9**	**3.5**	**3.7**

constraints were imposed in terms of the resources available to accomplish any particular task at any point of time, safety requirements, and other physical restrictions. A screenshot of the carrier simulation environment is shown in Figure 4 and further details of the system design that facilitates interactions between the user and the scheduler are available in Ryan et al. (2011). To handle the uncertainties corresponding to the various discrete failure modes modeled using Poisson distribu-

Table 3: Performance comparison on three MIPLIB-Hard problems where the Boost-LP algorithm does not perform as well as in Table 2 but still outperforms all the other ILP solution methods significantly. Two models are used by the Boost-LP algorithm and the larger model provides an estimation error $\leq 5.0\%$ at the expense of increased computation time.

Approach	Performance metric	Problem set		
		`momentum1`	`protfold`	`sp97ar`
COIN-OR Cbc	Computation time (s)	3701.32	3600.49	3601.52
CPLEX	Computation time (s)	2902.41[8]	3600.41	3609.36
Average solution of	Computation time (s)	1488.52	1479.27	1514.83
ξ-similar training problems	Infeasible solutions (%)	37.7	35.9	36.1
(nearest neighbor	LP solution error (%)	3.2	**3.1**	3.2
estimation)	ILP solution error (%)	5.4	5.2	5.2
Warm starting	Computation time (s)	3146.75	3532.79	3724.68
interior-point solver	LP solution error (%)	**3.0**	**3.1**	**3.1**
using closest training problem	ILP solution error (%)	5.2	5.3	5.1
Warm starting	Computation time (s)	2513.77	2769.15	2819.20
interior-point solver	LP solution error (%)	4.8	4.7	4.6
using parent tree node	ILP solution error (%)	8.2	7.9	7.7
Randomized	Computation time (s)	2144.38	2210.70	2225.06
rounding	ILP solution error (%)	19.2	16.1	16.5
Exact multiparametric	Computation time (s)	725.39	711.28	716.42
	Computation time (s)	426.55	398.25	401.59
Approximate multiparametric	LP solution error (%)	6.4	6.0	6.1
	ILP solution error (%)	9.7	8.8	8.9
	Computation time (s)	**74.43**	**65.75**	**69.08**
Boost-LP Inference	Cross-validation error (%)	6.2	5.9	5.8
smaller model	Theoretical LP soln. error (%)	13.9	13.5	13.6
(W = 500, J = 8)	Observed LP soln. error (%)	7.6	7.5	7.5
	ILP soln. error (%)	13.1	13.0	12.8
	Computation time (s)	185.33	165.03	172.35
Boost-LP Inference	Cross-validation error (%)	2.7	2.5	2.5
larger model	Theoretical LP soln. error (%)	6.2	6.2	6.0
(W = 1000, J = 16)	Observed LP soln. error (%)	3.4	3.3	3.2
	ILP soln. error (%)	**5.0**	**5.0**	**4.9**

tions, we cast the problem in a stochastic integer linear programming (SILP) form. A disjunctive formulation of the generalized Benders' decomposition approach was adopted from Sen and Sherali (2006) as the BAB solution method. We present the actual problem formulation in Appendix C.

We considered an operation involving 20 aircraft (with identical priorities) of 4 different types, 28 possible tasks, a planning time horizon of 25, and 4 different failure modes using a number of

Figure 4: Screenshot of aicraft carrier deck simulation environment

popular regression approaches and loss functions. The number of predictor variables p was equal to 35,628 and the cardinality of the training data set N was equal to 6,254. The number of regression trees, W, and the number of terminal regions in any tree, J, were empirically chosen as 500 and 8 respectively. All the 16 training and 4 test problem instances had the same initial conditions, the same number of emergency aircraft, the same geometric layout of the carrier deck, and the same sequence of tasks. However, the means of the Poisson distributions for the occurrences of failures, and the specific airborne aircraft that develop emergency fuel and hydraulic leak conditions were generated randomly.

Table 4 evaluates the performance of the boosting tree algorithm without and with avoidance of infeasibility (discussed in section 3.3) and addition of new training data points (described in section 3.2.1). As expected, the cross-validation and solution errors as well as the percentage of infeasible solutions are lower when additional training data points are used. There is no change in the computation time though as inference is performed on identical complexity models. We also observe that the penalization loss function, L_p, reduces the percentage of infeasible LP solutions as compared to the standard Huber loss function, L_H, considerably. Of course, eliminating the infeasible solutions, by checking for constraint violations and solving the LP problems with violations exactly, increases the computation time; however, this increase is relatively small.

5.3 Vehicle Routing with Time Windows

Our third example domain was taken from Solomon (1987), where a set of vehicles, originating from a central depot, have to serve a set of customers with known demands exactly once without exceeding the vehicle capacities. Additional constraints in the form of allowable delivery times or time windows are imposed for certain customers and the number of vehicles as well as the routes of the individual vehicles have to be determined by the solver. We adopted the ILP-based set partitioning formulation of the problem given in Desrochers et al. (1992). The exact problem formulation is provided in Appendix D. The routing problem is another instance of a repeated ILP problem. Practical application is found in any package shipment company operations, where vehicles are despatched to identical or nearby pick up and delivery sites every day within similar time frames.

9. Non-zero values do not imply that Boost-LP Inference returns infeasible solutions for LP problems that have feasible and bounded solutions; rather infeasible solutions are detected and the corresponding LPs are solved using COIN-OR Cbc - table entries show the cost and benefits of performing this additional step

Table 4: Performance evaluation of the Boost-LP algorithm on aircraft carrier deck scheduling problem with and without infeasibility avoidance and addition of new training data points.

	Performance metric	Infeasibility unavoided		Infeasibility avoided	
		L_H	L_p	L_H	L_p
	Computation time (s)	14.76	15.83	17.46	18.36
Training data points not added	Cross-validation error (%)	2.0	2.2	1.9	2.0
	Infeasible solutions[9](%)	7.1	1.2	0.0	0.0
	Theoretical LP solution error (%)	5.0	5.5	4.8	5.2
	Observed LP solution error (%)	2.8	3.0	2.8	2.9
	SILP solution error (%)	4.2	4.0	4.1	3.9
Training data points added	Cross-validation error (%)	0.9	1.0	0.8	0.9
	Infeasible solutions (%)	6.7	0.9	0.0	0.0
	Theoretical LP solution error (%)	4.1	4.6	3.8	4.0
	Observed LP solution error (%)	2.2	2.5	2.0	2.1
	SILP solution error (%)	3.3	3.2	3.2	3.1

Figure 5 plots the decrease in estimation error as a function of computation time (until the Boost-LP algorithm terminates) for all the different ILP solution approaches using three benchmark data sets, each consisting of problems that require servicing 100 customers. Identical data sets were used in Solomon (1987) and Desrochers et al. (1992), and we obtained them online at `http://www.idsia.ch/~luca/macs-vrptw/problems/welcome.htm`. The 2D coordinates of the customers and the time windows are generated in a clustered way in the data set C2, randomly following uniform distributions in the data set R2, and in a semi-clustered way in the third data set RC2. The data sets have relatively long scheduling horizons, allowing more than 30 customers per vehicle. Each dataset contains 8-12 separate problems with different proportions of customers having time windows. We used the last problem in every data set for testing the models learned over the LP problems occurring at the BAB nodes of all the other ILP problems present in that particular dataset. The number of regression trees, W, and the number of terminal regions in any tree, J, were chosen as 500 and 8 respectively.

The Boost-LP algorithm outperforms all the other methods significantly in every data set by providing much faster decay of estimation error and the smallest error magnitude at any time instant except occasionally toward the very beginning. Although best performance is obtained for the data set C2 and worst for R2, the change in performance quality from one data set to another is quite small. Thus, we show that our approach is capable of producing satisfactory results for problem domains with varying characteristics.

5.4 Vehicle Control

The final problem domain was taken from the robotics literature. While robot planning problems can be solved in many different ways, casting them in mixed integer linear programming (MILP)

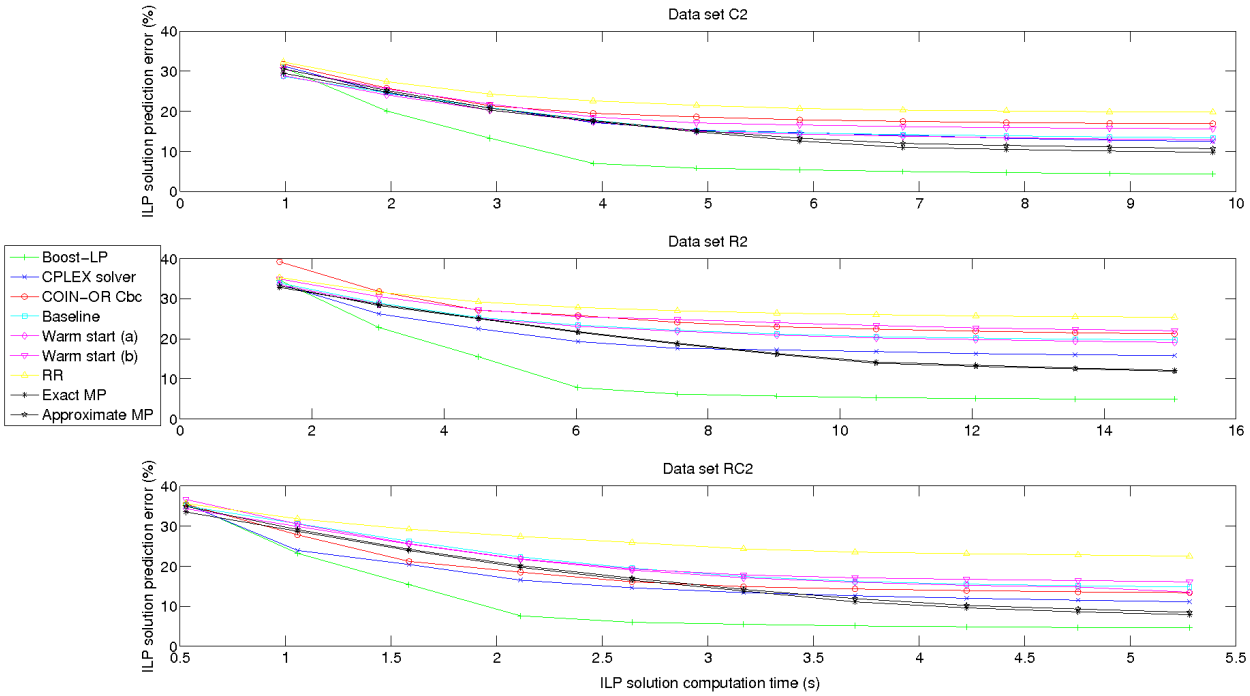

Figure 5: Decrease in solution prediction error as a function of computation time for different ILP solution approaches on vehicle routing problem. RR, Exact MP, and Approximate MP are acronyms for randomized rounding, exact multiparametric, and approximate multiparametric methods. Warm start (a) and (b) denote warm starting interior-point LP solver using closest ξ-similar training LP problem and parent BAB tree node LP problem respectively.

form is common in the controls community due to the ease of encoding system constraints such as time-varying vehicle dynamics. We considered both single-vehicle and multi-vehicle control problems, wherein 50 random instances of each problem were generated with the same parameter ranges as given in the source articles, out of which 40 were used for training and the remaining 10 for testing. The locations and dimensions of the obstacles, durations of assigned tasks to vehicles, and control input (force) and disturbance bounds were varied independently from one problem instance to the next. The number of vehicles, their initial locations and configurations, goal locations, the number and geometry of the obstacles, the number and types of assigned tasks for each vehicle, and the objective functions remained constant in all the problem instances. Thus, both the constraint matrix and vector differed among the constituent LP problems of the MILP instances, while still satisfying all the requirements to form the member set of a ξ-similar LP problem space. W and J were selected as 1000 and 16 respectively.

5.4.1 SINGLE-VEHICLE CONTROL

In this domain, a vehicle equipped with a three-motor omnidirectional drive needs to move to a goal location in 2D without colliding with stationary, circular obstacles of known radii. We considered the minimum-control-effort trajectory-generation problem within the framework of the iterative MILP time-step selection algorithm presented in Earl and Andrea (2005). The actual problem formulation is shown in Appendix E. Table 5 shows that the cross-validation error and the solution errors increase non-linearly with the number of obstacles. Even though different values of the threshold vector ζ components are chosen, with 0.4, 0.3, 0.2, and 0.1 times the range of the corresponding response variables being used for 5, 10, 15, and 20 obstacles respectively, ϵ increases with additional obstacle constraints such that the overall prediction error cannot be maintained at the same level. Importantly though, the computation time speed-up factor, as compared with CPLEX, the most widely used optimization solver, improves with the number of obstacles. This happens because the complexity of the boosting tree model remains unchanged in all the cases, thereby, keeping the inference time constant, and reducing the *rate* of exponential growth in the computation time.

Table 5: Performance evaluation of the Boost-LP algorithm on single-vehicle control problems with varying number of obstacles.

Performance metric	Number of obstacles			
	5	10	15	20
CPLEX computation time (s)	16.75	39.55	157.58	844.33
Boost-LP Inference computation time (s)	0.49	0.82	2.49	11.01
Speed-up factor	34X	48X	63X	77X
Observed LP solution error (%)	0.3	0.8	1.5	2.9
MILP solution error (%)	0.5	1.3	2.7	5.0

5.4.2 MULTI-VEHICLE CONTROL

We obtained the multi-vehicle problem from Richards and How (2004) that involved decentralized, cooperative control of UAVs with independent dynamics but coupled constraints. The exact problem formulation is presented in Appendix E. Figure 6 shows that the computation time speed-up factor improves with the number of vehicles as the complexity of the boosting tree model remains unchanged in all the cases. Figure 7 highlights the effect of model complexity on the computation time as well as the solution prediction error for the same number of vehicles (7 UAVs). Clearly, there is a trade-off between computational savings and prediction accuracy. Moreover, the prediction accuracy does not improve significantly beyond a certain level of model complexity (for a given training data set and ζ), indicating that learning a very complex regression model is of little use as it is only going to increase the computation time without significantly reducing the prediction error.

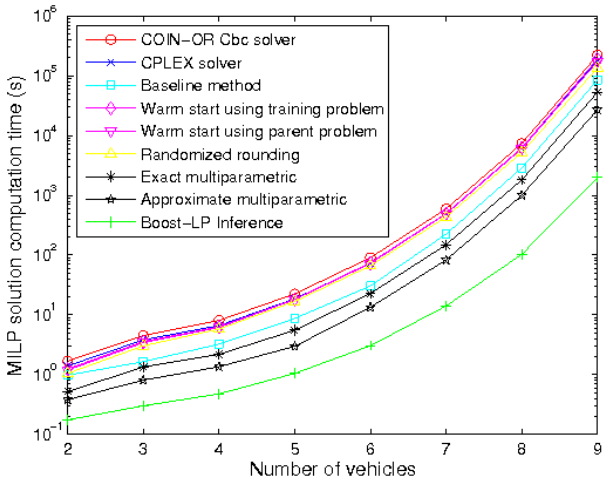

Figure 6: Computation time comparison of the Boost-LP Inference algorithm with alternative ILP solvers for decentralized UAV control problems involving varying number of vehicles. Note that the plot is in semi-log scale and error bars are not shown as they are negligibly small.

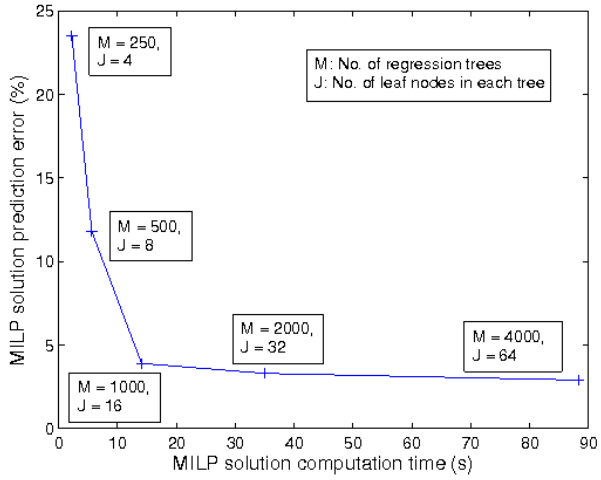

Figure 7: Computation time versus solution error performance curve of the Boost-LP Inference algorithm for decentralized UAV control problem.

6. Conclusions

We present a regression-based approach to obtain fast and provably-close-to-optimal solutions of repeated ILP problems with practical applications in many planning, scheduling, routing, and control domains, such that the objective function and constraint coefficients do not vary too much from

one problem to the other. We give a formal definition of ξ-similar LP problems that arise in the branch and bound tree nodes of such ILP problems and modify the gradient boosting tree algorithm to devise the Boost-LP algorithm for learning the solutions of the ξ-similar LP problems. We then use the notion of similarity, properties of boosting trees, and our construction of predictor variable vectors to provide absolute prediction error bounds for both the individual decision variables and the overall LP problem objective function. We also show that the Boost-LP algorithm infers the solutions of ILP problems in a fraction of the time required by both commercial and open-source optimizers as well as other existing approaches, while preserving reasonably good solution accuracy in all the four representative problem domains.

Future work includes investigating active learning from partially generated BAB trees and inferring solutions of harder problems consisting of higher-dimensional solution vectors from lower-dimensional ones. These directions will enable us to bypass the potentially time-consuming and perhaps infeasible step of constructing a large training set whenever we encounter a fundamentally different class of ILP problems that form the member sets of new ξ-similar LP problem spaces. Such extensions will also allow us to relax the requirement of identical solution vector size for all the LP problems and, thereby, permit comparisons with branch and cut, and branch, cut and price methods that use various cutting planes to speed up computations. It may also be interesting to learn the similarity metrics using kernelized sorting or other matching techniques, and then convert them to parametric kernels such as Gaussian processes to estimate the LP solutions directly from a library of pre-computed solutions of ξ-similar LP problems. Another useful research direction is to extend our work to quadratic objective functions, which will expand its applicability to a broader class of robust and optimal control problems arising in the model predictive and receding horizon control settings.

Acknowledgments

We acknowledge support for this project from the Office of Naval Research Science of Autonomy program (Contract No. N000140910625). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsoring agency. We are also grateful to Andrew Barnes, Geoffrey Gordon, Jonathan How, Sertac Karaman, Masahiro Ono, and Brian Williams for many helpful discussions and suggestions.

Appendix A.

Two child regions R_1 and R_2 are created at every internal parent node of a regression tree by selecting a splitting variable $o, 1 \leq o \leq p$ and a split value s that define a pair of half-planes

$$R_1 = \{v \mid v_o \leq s\}, R_2 = \{v \mid v_o > s\}$$

where v_o represents the the oth component of the vector v. We select o and s using a greedy stategy to minimize the residual sum of squares error that solves

$$\min_{o,s} \left[\min_{\gamma_1} \sum_{v_k \in R_1} (r_k - \gamma_1)^2 + \min_{\gamma_2} \sum_{v_k \in R_2} (r_k - \gamma_2)^2 \right] \tag{31}$$

where v_k is the predictor variable vector corresponding to LP_k. r_k is the target value for the problem LP_k and is chosen as the negative gradient of the loss functional that is given by $-\partial L(y_k, f(v_k))/\partial f(v_k)$. For any choice of o and s, the inner minimization in (31) is solved by

$$\hat{\gamma}_1 = \frac{\sum_{v_k \in R_1} r_k}{N_1}, \hat{\gamma}_2 = \frac{\sum_{v_k \in R_2} r_k}{N_2}$$

where N_1 and N_2 denote the number of training data points in R_1 and R_2 respectively. Each regression tree is grown by binary partitioning till the number of leaf nodes equals or exceeds the fixed size J that is chosen *a priori*. If required, the tree is then pruned using the technique of cost-complexity pruning described in Hastie et al. (2008) to reduce the number of leaf nodes to J.

Appendix B.

Expanding the quadratic equation involving $\epsilon_{i,tr}$ in section 4.1, we have

$$\epsilon_{i,tr}^2 + 2\epsilon_{i,tr} \sum_j a_{i,tr}^j + \sum_j (a_{i,tr}^j)^2 (1 - \xi_{i,r}^2) = 0 \quad .$$

The solution is then given by

$$\begin{aligned} \epsilon_{i,tr} &= \frac{-2\sum_j a_{i,tr}^j \pm \sqrt{4\sum_j (a_{i,tr}^j)^2 - 4\sum_j (a_{i,tr}^j)^2 (1 - \xi_{i,r}^2)}}{2} \\ &= \sum_j a_{i,tr}^j (-1 \pm \xi_{i,r}) \quad . \end{aligned}$$

Considering only the positive $\xi_{i,r}$ solution, we find that $\epsilon_{i,tr}$ is a small number close to 0 (either positive or negative depending upon the values of $a_{i,tr}^j$) as $\xi_{i,tr}$ is close to 1.

Appendix C.

We use a two-stage stochastic integer linear programming (SILP) framework, where the expected impact of decisions under uncertainty are minimized in the first stage and integral recourse actions are taken in the second stage to compensate for the errors in the decision-making process in the first stage. The first stage problem is formally represented as

$$\min E[h(\widetilde{\omega})]$$

where $E[.]$ denotes the expectation operator and $\widetilde{\omega}$ is a random variable vector. Then, for any realization ω of $\widetilde{\omega}$, the second stage problem is expressed as

$$\begin{aligned} h(\omega) : \min g^T y \\ \text{s. t. } Wy \geq r(\omega) \\ y \geq 0, y \in Z^n \quad . \end{aligned}$$

All the second stage decision variables are binary. We use the iterative $D^2 - BAC$ algorithm described in Section 3 of Sen and Sherali (2006) to solve this SILP problem and follow the formulation

II given in Möhring et al. (2001) to deal with temporal dependencies of tasks and irregular task starting time costs. The binary decision variables y_{ijk} are equal to 1 if aircraft $i \in \{1, \ldots, I\}$ begins task j at time step $k \in \{1, \ldots, K\}$, and 0 otherwise. Each aircraft has a user-specified priority p_i and a set of tasks T_i that it needs to accomplish, and the goal is to minimize the overall weighted time taken to complete all the sets of tasks.

The tasks consist of parking on deck, refueling in one of the five deck stations, loading weapons by docking at one of the two weapons elevators, going for maintenance by using one of the three hangar elevators, taxiing to various deck zones, launching or taking-off from one of the four catapults, going to mission, stacking up in a queue called the Marshal Stack after finishing mission, approaching the deck from the lowermost level of the Marshal Stack, landing, boltering due to failed landing if arresting wires are missed, and aerial refueling using tankers. Parking spots, refueling stations, launching catapults, landing strip (only one), hangar, and weapons elevators are together termed as the deck resources. Due to electro-mechanical failures, the deck resources (except the parking spots and the landing strip) may not be available at certain times during the operation. The failure rate of each type of deck resource is modeled by an independent Poisson distribution.

Moreover, an airborne aircraft may develop fuel leak and/or hydraulic failure, both of which are assumed to occur deterministically. Operation safety requirements mean that such an aircraft needs to land as soon as possible by moving down in the Marshal Stack and rearranging the levels of all the other aircraft accordingly. The geometric layout of the deck and its current configuration impose additional constraints, namely that certain parking spots may be occupied, two of the catapults (numbered 3 and 4) cannot be used when an aircraft is landing, and the two closely-located catapults (both 1 and 2 as well as 3 and 4) cannot be used simultaneously. There are four types of aircraft, two manned and two unmanned, each of which has a different fuel consumption rate, maximum speed, cruise speed, and approach speed. Thus, they take different times to complete the same task. The user can also specify own preferences to complete certain tasks for specific aircraft within desired time values. These preferences are handled as soft constraints that are satisfied by including violation margins.

The temporal dependency of tasks for each aircraft is represented by a weighted digraph $G_i = (V_i, A_i)$ with $A_i = \{(m_i, n_i) : d_{m_i n_i} > -\infty\}$, where $d_{m_i n_i}$ is the time lag such that the starting time for task n_i, S_{n_i}, is correlated to the starting time for task m_i, S_{m_i}, by an inequality of the form $S_{n_i} \geq S_{m_i} + d_{m_i n_i}$. The second stage problem is then written as

$$\min \sum_i \sum_j \sum_k p_i(t_{ij} + k\Delta_t) y_{ijk}$$

subject to the constraints

$$\sum_{k} y_{ij_ik} = 1, \quad \forall i \quad \text{and} \quad j_i \in T_i$$
$$\sum_{s=k}^{K} y_{im_is} + \sum_{s=1}^{k+d_{m_in_i}-1} y_{in_is} \leq 1, \quad \forall i, k \quad \text{and} \quad (m_i, n_i) \in A_i$$
$$\sum_{i} y_{ijk} \leq N_{jk}(\omega) \quad \forall j, k$$
$$\sum_{i} (y_{ij_{t1}k} + y_{ij_{t2}k}) \leq 1, \quad \sum_{i} (y_{ij_{t3}k} + y_{ij_{t4}k}) \leq 1, \quad \sum_{i} (y_{ij_lk} + y_{ij_{t3}k}) \leq 1, \quad \sum_{i} (y_{ij_lk} + y_{ij_{t4}k}) \leq 1 \quad \forall k$$
$$\sum_{k=1}^{S_{ci'}} y_{i'j_lk} = 1, \quad \forall i' \in I_c$$
$$\sum_{k=1}^{S_{pi''j''}+T_m} y_{i''j''k} = 1, \quad \forall i'' \in I_p, j'' \in T_{i''} \quad .$$

Here, t_{ij} is the estimated time to complete task j by aircraft i, $N_{jk}(\omega)$ denotes the number of available deck resources to perform task j at time k, and j_l, j_{t1}, j_{t2}, j_{t3}, and j_{t4} represent landing, and take-off from catapult 1, catapult 2, catapult 3, and catapult 4 respectively. I_c and I_p denote the set of critical aircraft and the set of aircraft with user-specified preferences respectively. $S_{ci'}$ is the permissible time limit within which a critical aircraft i' should start landing, $S_{pi''j''}$ is the user-stipulated time limits to complete task j'' for the aircraft $i'' \in I_p$, and T_m is the allowable time margin for violating the preference constraints. Note that all the actual time values are conservatively converted to integral values by taking the interval between two successive time instants (Δ_t) as a constant. Also note that if our initial value of T_m results in an infeasible solution, we increase it until the problem has optimal solutions.

Appendix D.

We follow the notation in Desrochers et al. (1992) and represent the vehicle routing problem as a graph $G = (N, A)$, where N is the set of nodes or customers and A is the set of route segments. Each edge $(i, j) \in A$ is associated with a cost c_{ij} given by the 2D Euclidean distance between the nodes i and j, and a duration t_{ij} that includes the servicing time at node i. Each customer has a known demand q_i and all the vehicles, with identical capacities equal to Q, originate and terminate at the same depot d. Servicing of a customer can begin at time T_i within the allowable time window defined by the earliest time a_i and the latest time b_i.

Let the set of feasible routes be denoted by $R \subseteq A$. Also, let δ_{ir} be a constant that assumes a value of 1 if route $r \in R$ covers customer $i \in N \backslash \{d\}$, and 0 otherwise. Let c_r be the cost associated with route r, which is equal to the sum of the costs of the edges of r. The binary decision variables are represented as x_r, that take values of 1 if route r is used, and 0 otherwise. The set partitioning

problem is then formulated as

$$\begin{aligned} &\min \sum_{r \in R} c_r x_r \\ &\text{s. t. } \delta_{ir} x_{ir} = 1, \quad i \in N \backslash \{d\} \\ &\qquad x_r \in \{0,1\}, \quad r \in R \quad . \end{aligned}$$

The key pre-processing step in solving this binary ILP problem is to identify the set R. Since identifying just the feasible set is computationally very challenging, we follow the second dynamic programming method presented in Desrochers et al. (1992) to obtain a superset of the exact feasible set with pseudopolynomial worst case time complexity. This method is a generalization of the multiple knapsack problem, and we refer the reader to sections 3 and 4.2 in Desrochers et al. (1992) for further details.

Appendix E.

Single-vehicle Control Problem

To cast the problem in mixed integer linear programming (MILP) form, the governing coupled and non-linear differential equations of the vehicle are simplified by restricting the admissible control set, defined by the time-dependent control input $(u_x(t), u_y(t), u_\theta(t))$, to lie within a cylinder whose radius is less than or equal to unity. The equations of motion for vehicle translation are given by

$$\begin{aligned} \ddot{x}(t) + \dot{x}(t) &= u_x(t) \\ \ddot{x}(t) + \dot{x}(t) &= u_y(t) \quad . \end{aligned}$$

After discretizing the control input in time and assuming that the input remains constant between time steps, the equations of motion can be written in discrete time-variant state-space form as

$$x_u[k+1] = A[k]x_u[k] + B[k]u[k] \tag{32}$$

where $t_u[k]$ is the time at step k, and $x_u[k] = x(t_u[k]) = (x_u[k], y_u[k], \dot{x}_u[k], \dot{y}_u[k])$, $u[k] = u(t_u[k]) = (u_x[k], u_y[k])$. Now, to incorporate the constraint that the admissible control set lies in a circle of fixed radius, the circle is approximated by a set of M_u linear inequalities that conservatively define an inscribed polygon. These linear constraints are given by

$$u_x[k] \sin \frac{2\pi m}{M_u} + u_y[k] \cos \frac{2\pi m}{M_u} \leq \cos \frac{\pi}{M_u}, \quad \forall m \in \{1, \ldots, M_u\} \quad . \tag{33}$$

Obstacle avoidance at discrete times is enforced in a similar manner by conservatively approximating each circular obstacle of radius R_{obst} and center coordinates $(x_{obst}[k], y_{obst}[k])$ with a polygon $O[k]$, defined using M_o linear inequalities of the form

$$O[k] = \left\{ (\overline{x}, \overline{y}) : (\overline{x} - x_{obst}[k]) \sin \frac{2\pi m}{M_o} + (\overline{y} - y_{obst}[k]) \cos \frac{2\pi m}{M_o} \leq R_{obst}, \quad \forall m \in \{1, \ldots, M_o\} \right\} \quad .$$

Since at least one constraint defining $O[k]$ needs to be violated in order to avoid the obstacle, an auxiliary binary variable $b_m[k] \in \{0,1\}$ is introduced to reformulate the M_o inequality constraints as

$$(\overline{x} - x_{obst}[k]) \sin \frac{2\pi m}{M_o} + (\overline{y} - y_{obst}[k]) \cos \frac{2\pi m}{M_o} > R_{obst} - H b_m[k], \quad \forall m \in \{1, \ldots, M_o\} \tag{34}$$

where H is a large positive number that is greater than the sum of the maximum dimension of the workspace and the obstacle radius. If $b_m[k] = 1$, the above inequality is trivially statisfied. On the other hand, if $b_m[k] = 0$, then the inequality becomes significant. To enforce that at least one of the constraints is significant, we have

$$\sum_{m=1}^{M_o} b_m[k] \leq M_o - 1 \quad . \tag{35}$$

We also have some boundary conditions that are given by the starting vehicle location $\mathbf{x}_0 = \mathbf{x}_s$, and we want to find the set of control inputs $\{\mathbf{u}[k]\}$, $k \in \{0, \ldots, N_u - 1\}$ that will move the vehicle to the final location $\mathbf{x}_{t_f} = \mathbf{x}_f$ while minimizing the cost function

$$J = \sum_{k=0}^{N_u-1} (|u_x[k]| + |u_y[k]|) \quad .$$

We introduce auxiliary continuous variables $z_x[k]$ and $z_y[k]$ to convert the absolute values in the cost function to linear form and add the following inequality constraints

$$\begin{aligned} -z_x[k] \leq u_x[k] \leq z_x[k] \\ -z_y[k] \leq u_y[k] \leq z_y[k] \quad . \end{aligned} \tag{36}$$

The cost function can then be written as

$$J = \sum_{k=0}^{N_u-1} (z_x[k] + z_y[k]) \quad . \tag{37}$$

The objective function (37), subject to the set of constraints given by (32), (33), (34), (35), and (36) form the overall MILP problem. We refer the reader to section III B in Earl and Andrea (2005) for details on the iterative time-selection algorithm that is used to solve the MILP problem with as few obstacle avoidance times as possible.

UAV Decentralized Control Problem

We use the Decentralized MPC algorithm presented in Richards and How (2004) for co-operative guidance of UAVs, where each vehicle only solves a sub-problem to generate its own plan, and then each of the sub-problems are solved only once per time-step without any iteration. Each vehicle p is modeled as a point mass subject to maximum force and speed limits given by F_p and V_p respectively. A bounded disturbance force with maximum value of $\tilde{F}_p$ is also included in the model to account for uncertainties. Thus, the linear, discrete-time dynamics model for each vehicle, with position $r_p \in \Re^2$, velocity $v_p \in \Re^2$, acted by a control force $f_p \in \Re^2$ in the presence of disturbance force $\tilde{f}_p \in \Re^2$, is expressed in discrete time-invariant state-space form as

$$x_p(k+1) = Ax_p(k) + Bu_p(k) + Ew_p(k)$$

with state $x_p(k) = [r_p(k\Delta_t)^T \quad \dot{r}_p(k\Delta_t)^T]^T = [r_p(k\Delta_t)^T \quad v_p(k\Delta_t)^T]^T$, control input $u_p(k) = f_p(k\Delta_t)$, and disturbance $w_p(k) = \tilde{f}_p(k\Delta_t)$, Δ_t being the constant interval between any two time

instants. The speed, force, and disturbance limits are then represented by the following system of linear constraints

$$\begin{aligned} [0 \quad G]x_p(k) &\leq 1V_p \\ Gu_p &\leq 1F_p \\ Gw_p &\leq 1\tilde{F}_p \end{aligned}$$

where G consists of rows of unit vectors. Collision avoidance is performed by constructing a square exclusion region of fixed length around every UAV, where no other vehicle is allowed to enter. The standard procedure of introducing binary variables and a large positive number is followed to enforce the collision avoidance constraints in the MILP framework. Some additional margin parameters are added to provide greater robustness; we refer the reader to section III in Richards and How (2004) for further details.

References

T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.

I. Adler and R. D. C. Monteiro. A geometric view of parametric linear programming. *Algorithmica*, 8(1-6):161–176, 1992.

A. A. Ageev and M. I. Sviridenko. Pipage rounding:a new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

A. Alessio and A. Bemporad. A survey on explicit model predictive control. In L. Magni, D. Raimonde, and F. Allgöwer, editors, *Nonlinear Model Predictive Control*, pages 345–369. Springer Berling / Heidelberg, 2009.

S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with application to dense graph arrangement problems. *Mathematical Programming:Series A*, 92(1): 1–36, 2002.

A. G. Banerjee, M. Ono, N. Roy, and B. C. Williams. Regression-based LP solver for chance-constrained finite horizon optimal control with nonconvex constraints. In *Proceedings of the American Control Conference*, San Francisco, California, USA, 2011.

J. E. Bell and P. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48, 2004.

H. Y. Benson and D. F. Shanno. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. *Computational Optimization and Applications*, 38(3):371–399, 2007.

D. Bertsimas, C. Teo, and R. Vohra. On dependent randomized rounding algorithms. *Operations Research Letters*, 24(3):105–114, 1999.

F. Borrelli, A. Bemporad, and M. Morari. Geometric algorithm for multiparametric linear programming. *Journal of Optimization Theory and Applications*, 118(3):515–540, 2003.

R. Brandenberg. Radii of regular polytopes. *Discrete & Computational Geometry*, 33(1):43–55, 2005.

R. Carr and S. Vempala. Randomized metarounding. *Random Structures & Algorithms*, 20(3): 343–352, 2002.

F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

V. Dua and E. N. Pistikopoulos. An algorithm for the solution of multiparametric mixed integer linear programming problems. *Annals of Operations Research*, 99(1-4):123–139, 2000.

M. G. Earl and R. D' Andrea. Iterative MILP methods for vehicle-control problems. *IEEE Transactions on Robotics*, 21(6):1158–1167, 2005.

J. Elizabeth and E. A. Yildrim. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, 41 (2):151–183, 2008.

A. Engau, M. F. Anjos, and A. Vannelli. A primal-dual slack approach to warmstarting interior-point methods for linear programming. In J. W. Chinneck, B. Kristjansson, and M. J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, pages 195–217. Springer, 2009.

A. Engau, M. F. Anjos, and A. Vannelli. On interior-point warmstarts for linear and combinatorial optimization. *SIAM Journal on Optimization*, 20(4):1828–1861, 2010.

C. Filippi. An algorithm for approximate multiparametric linear programming. *Journal of Optimization Theory and Applications*, 120(1):73–95, 2004.

T. Gal and J. Nedoma. Multiparametric linear programming. *Management Science*, 18(7):406–422, 1972.

M. X. Goemans and D. P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

T. F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, Boca Raton, Florida, USA, 2007.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, New York, USA, 2008.

K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

C. Y. Lee, S. Piramuthu, and Y. K. Tsai. Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, 35(4):1171–1191, 1997.

E. K. Lee and J. E. Mitchell. Integer programming: Branch-and-bound methods. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 525–533. Kluwer Academic Press, 2001.

Z. Li and G. Ierapetritou. Process scheduling under uncertainty using multiparametric programming. *AIChE Journal*, 53(12):3183–3203, 2007.

R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

S. J. Louis, X. Yin, and Z. Y. Yuan. Multiple vehicle routing with time windows using genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation*, Washington DC, USA, 1999.

R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. On project scheduling with irregular starting time costs. *Operations Research Letters*, 28(4):149–154, 2001.

R. Nallusamy, K. Duraiswamy, R. Dhanalakshmi, and P. Parthiban. Optimization of multiple vehicle routing problems using approximation algorithms. *International Journal of Engineering Science and Technology*, 1(3):129–135, 2009.

E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, editors. *Multi-parametric Programming: Volume 1: Theory, Algorithms, and Applications*. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, Germany, 2007.

P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

P. Raghavan and C. D. Thompson. Randomized rounding:a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

A. Richards and J. How. Decentralized model predictive control of cooperating UAVs. In *Proceedings of the IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, 2004.

J. C. Ryan, M. L. Cummings, N. Roy, A. Banerjee, and A. Schulte. Designing an interactive local and global decision support system for aircraft carrier deck scheduling. In *Proceedings of AIAA Infotech@Aerospace*, St. Louis, Missouri, USA, 2011.

S. Sen and H. D. Sherali. Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming:Series A*, 106(2):203–223, 2006.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999.

O. Telelis and P. Stamatopoulos. Combinatorial optimization through statistical instance-based learning. In *Proceedings of the 13th IEEE International Conference on Tools in Artificial Intelligence*, Dallas, Texas, USA, 2001.

D. Vladušič, A. Černivec, and B. Slivnik. Improving job scheduling in grid environments with use of simple machine learning methods. In *Proceedings of the 6th International Conference on Information Technology: New Generations*, Las Vegas, Nevada, USA, 2009.

D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, New York, USA, 2011.

E. A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

W. Zhang and T. G. Dietterich. High performance job-shop scheduling with a time-delay TD(λ) network. In *Advances in Neural Processing Information Systems*, Denver, Colorado, USA, 1996.