Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2010-045 September 18, 2010

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

Learning Solutions of Similar Linear Programming Problems using Boosting Trees

Ashis Gopal Banerjee and Nicholas Roy

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
ashis,nickroy@csail.mit.edu

Abstract. In many optimization problems, similar linear programming (LP) problems occur in the nodes of the branch and bound trees that are used to solve integer (mixed or pure, deterministic or stochastic) programming problems. Similar LP problems are also found in problem domains where the objective function and constraint coefficients vary due to uncertainties in the operating conditions. In this report, we present a regression technique for learning a set of functions that map the objective function and the constraints to the decision variables of such an LP system by modifying boosting trees, an algorithm we term the Boost-LP algorithm. Matrix transformations and geometric properties of boosting trees are utilized to provide theoretical performance guarantees on the predicted values. The standard form of the loss function is altered to reduce the possibility of generating infeasible LP solutions. Experimental results on three different problems, one each on scheduling, routing, and planning respectively, demonstrate the effectiveness of the Boost-LP algorithm in providing significant computational benefits over regular optimization solvers without generating solutions that deviate appreciably from the optimum values.

1 Introduction

Optimization plays an important role in many scheduling, planning, control, and navigation problems for autonomous systems. Various optimization techniques are used extensively in a wide variety of settings, ranging from flexible manufacturing systems and warehouse operations to guidance of unmanned aerial, ground, and underwater vehicles. Solving some form of a linear program, where both the objective function and the constraints are linear functions of the decision variables, lies at the core of a large number of such techniques.

Even while solving optimization problems that cannot be directly cast in linear programming (LP) form, relaxed LP versions of the problems are typically used in the nodes of the branch-and-bound (BAB) or branch-and-cut (BAC) trees to solve integer linear programming (ILP), mixed integer linear programming (MILP), as well as stochastic integer linear programming (SILP) problems. Relaxed LP problems often share multiple common constraints and always contain the same objective function and number of decision variables. Uncertainties

in the operating conditions may change some of the objective function and constraint coefficient values and result in the generation of some new constraints and/or deletion of old ones without altering the number of decision variables. Henceforth, we refer to problems that contain the same number of decision variables, similar objective function coefficients, and similar constraint coefficients with some variation in the number of constraints as a set of *similar* LP problems.

Although many commercial solvers such as CPLEX, MATLAB etc. as well as open-source software libraries like COIN-OR [4] provide efficient and robust implementations, it takes a fair amount of time to compute the solutions of ILP, MILP, and SILP problems involving tens of thousands of decision variables and comparable number of constraints. Usually, it takes few seconds to solve a particular LP problem; consequently, few minutes or even hours are needed to compute a solution to the overall IP problem. Moreover, the solution needs to be recomputed whenever even one of the coefficient value changes. This makes IP problems computationally intractable for large-scale, real-time systems.

We observed that the solutions of similar LP problems are usually quite similar themselves in the sense that most of the optimum decision variable values do not change much. Hence, we believe that supervised machine learning, and more specifically, regression, can be used to learn from the solutions of given LP problems to predict the solutions of new but similar LP problems. Fast inference can then be performed over such regression models at run-time to quickly estimate the LP solutions instead of computing them using standard optimizations solvers. This provides an alternate and practically useful route for solving large-scale optimization problems.

In this report, we present a boosting tree-based approach, named as the Boost-LP algorithm, to learn a set of functions that map the objective function and constraints to the individual decision variables of similar LP problems. Thus, the predictor variable vector consists of the objective function and constraint cofficients and the response variables are the decision variables themselves.

We provide absolute performance guarantees on the objective function and the decision variable values. Our prediction error bounds are based on a matrix transformation of the system of inequality constraints to a vector having the same dimensionality as that of the decision variables, and geometric properties of the axis-aligned hyperbox arrangement of the predictor variable space created by the boosting trees. The bounds consist of a parametric vector that can be chosen appropriately by generating additional training data points to restrict the errors within pre-defined thresholds, which can be lowered up to a certain limit as determined by the LP system parameters.

However, boosting trees in their standard form, cannot ensure that the predicted solution to the LP problem satisfies all the problem constraints. In order to bias the boosting trees to avoid predicting infeasible solutions, we modify the standard loss function to penalize selecting response values that lie outside the feasible region of the training set LP problems as the constant modeled value inside the boosting tree regions.

Our algorithm is tested on three real-world problems involving scheduling of aicraft deck carrier operations, vehicle routing with time window constraints, and vehicle planning and control respectively. Boosting trees are found to perform significantly better than many other standard regression techniques and achieve marked computational speed-up (up to almost 100 times) as compared to the conventional optimization solvers. Infeasible solutions are only generated in a tiny fraction (1-2)% of the cases and are avoided completely by switching over to actually solving the LP whenever such a situation occurs. Promising results are obtained in terms of the overall ILP, MILP, and SILP solutions, where the predicted values never deviate by more than 5% of the actual. We also observe that our approach is quite robust to variations in the problem parameters; however, the performance degrades as the number of distinctly different constraints increases for a fixed training set size.

2 Related Work

Relatively little work has been done in applying machine learning, especially non-evolutionary techniques, to combinatorial optimization problems. Zhang and Dietterich [17] applied temporal difference learning to job shop scheduling using a time-delay neural network architecture to learn useful search heuristics for obtaining better schedules in shorter periods of time as compared to other algorithms. Lee et al. [9] 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. Telelis and Stamatopoulos [13] showed that kernel regression-supported heuristic methodology performed better than problem-specific heuristic search techniques in knapsack and set partitioning problems. More recently, Vladušič et al. [16] employed locally weighted learning, naïve Bayes, and decision trees for job shop scheduling over heterogenous Grid nodes to obtain improvements over random, round-robin, and first available algorithms.

Evolutionary techniques have been popularly used for solving vehicle routing problems (VRP), where a number of cities have to 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. We only discuss a few representative papers here for the sake of brevity. Louis et al. [10] modified the merge crossover operators in a GA to achieve good performance in the multiple VRP (involving multiple vehicles) with clustered customer locations. Bell and McMullen [1] modified the standard ant colony optimization (ACO) technique to search for multiple routes of the VRP. They obtained solutions that were very close to the optimum in the case of small problems; however, the performance was worse as compared to single route-searching ACO for large-scale problems. Nallusamy et al. [11] recently applied k-means clustering to first convert the multiple VRP into individual VRPs and then used a GA along with heuristics to solve each instance separately. In terms of per-

formance, their approach provided a trade-off between optimality and usage of computational resources.

Thus, we can see that the existing machine learning-based approaches have had mixed success in terms of solving NP-hard optimization problems. In this report, we present an alternative constrained regression-based technique that enables us to come up with performance guarantees as opposed to heuristic and evolutionary approaches. Recently, Nguyen et al. [12] applied such a constrained regression technique for calibrating cost models. They obtained promising results as compared to standard least squares, lasso, and ridge regression techniques, which motivated us to follow a similar approach.

3 Boosting Tree-Based Learning Approach

3.1 Problem Formulation

We are given a set of N Linear Programming (LP) problems $\{LP_1, \ldots, LP_N\}$ and their solutions $\{x_1^*, \ldots, x_N^*\}$, where any LP_k can be represented in the generic form as:

$$\max z_k = c_k^T x, \tag{1}$$

$$\text{s. t. } A_k x \leq b_k, x \geq 0 \tag{2}$$

Here, $c_k, x \in \mathbb{R}^n$, $A_k \in \mathbb{R}^{m_k, n}$, and $b_k \in \mathbb{R}^{m_k}$ $\forall k$. So all the LP problems are similar in the sense that the decision variable vector x and the objective function vector c_k have identical dimensionality n. However, no restrictions are imposed on the number of constraints m_k in the different LP problems.

As discussed in Sect. 1, we are interested in developing a regression model of this LP system in order to predict the solution of any new but similar LP problem. A separate regressor function is used for inferring each component of the vector x to avoid the computational complexity associated with learning multiple response variables simultaneously. The inter-dependence of the decision variables is captured by incorporating all the problem constraints in the predictor variable vector (given by (5)) and also by modifying the loss function suitably (discussed in Sect. 3.2).

Thus, we want to learn a set of n functions

$$f_i : (A^s, b^s, c^s) \mapsto x_i, 1 \leq i \leq n \tag{3}$$

which are used to estimate the optimum x for any test LP problem. Any regression model requires a set of predictor variables (vectors denoted by v) and a set of response variables (scalars denoted by y). In our case, the optimum value of each LP decision variable x_i^* acts as the training set response variable for the corresponding function f_i. In order to generate the predictor variable vector for the training set, we first transform the first system of inequalities in (2) to a slightly different form as given by $A'_k x \leq b'_k$, where $A'_k \in \mathbb{R}^{m'_k, n}$ and $b'_k \in \mathbb{R}^{m'_k}$

represent truncated forms of A_k and b_k that only contain the m'_k *active constraints* at the optimum solution x^*_k. By introducing a new matrix W_k, we can transform the matrix constraint on x to a vector constraint that is given by

$$x \leq (W_k^T A'_k)^{-1} W_k^T b'_k = d_k \tag{4}$$

where $W_k \in \mathbb{R}^{m'_k, n}$ should be non-negative and the product matrix $W_k^T A'_k$ must be non-singular. The form of (4) is similar to a least square solution of (2); W_k is used instead of A_k to avoid the problem of singularity for non-unique x^*.

In order to construct the matrix W_k, let us represent it as $\{w_1, \ldots, w_{m'_k}\}^T$ and A'_k as $\{a_1, \ldots, a_{m'_k}\}^T$. We select each $w_i = e_{q_i}$, where $e_{q_i} \in \Re^n$ is a unitary vector and $q_i \in [1, n]$ denotes the position of the unity element such that $e_{q_i}^T a_i \neq 0 \forall i$. This ensures that $W_k^T A'_k$ is strongly non-singular as all of its principal submatrices are non-zero.

If multiple choices of q_i exist, then we select it in such a manner that at least one non-zero entry exists in every column of W_k. Ties are broken randomly. This heuristic enables us to associate *relevant* constraints for every x_i (constraints where coefficients of x_i in A'_k are non-zero), weigh the corresponding relevant constraint coefficients equally by unity, perform matrix division, and utilize the obtained value as the predictor variable vector value for the particular problem LP_k. If a column only contains zero elements, we eliminate this particular column from both W_k and A'_k and put $d_{i,k} = 0$, which takes care of the non-singularity problem for non-unique x^*. This is referred to later as *Condition 1* and is useful in bounding the LP solution prediction errors in Sect. 4.

Given the transformation (4), the common predictor variable vector v_k for all the functions f in a problem LP_k is formed by augmenting c_k with d_k. Thus, effectively, we are learning $f_i : v \mapsto x_i, 1 \leq i \leq n$, where a specific training problem predictor variable vector instance is given by

$$v_k = [c_{1,k}, \ldots, c_{n,k} \vdots d_{1,k}, \ldots, d_{1,n}]^T \tag{5}$$

Clearly, the size of v_k is always equal to $p = 2n$ for any value of k. Thus, transformation (4) not only provides a compact way of encoding all the LP problem parameters A, b, and c, it also results in a constant predictor variable vector size that is independent of $m_k, 1 \leq k \leq N$. The set of active constraints is, however, unknown for the test LP problem. So, the the entire matrix A and vector b is used for generating d. We also assume that the test LP problem parameters are such that all the components of the corresponding v vector lie within the range defined by the training set problems. If this is not the case, it is hypothesized that the LP problem is potentially infeasible and a standard optimizer is invoked to validate the hypothesis and obtain a feasible solution if necessary.

3.2 Algorithm Description

We have developed a modified version of the standard boosting tree algorithm, referred to as the boost-LP algorithm, to learn each regressor function f_i. For the

sake of completeness, we first present the standard boosting tree algorithm [8]. The multivariate regressor function is represented as a sum of trees $T(v;\Theta_m)$, each of which is given by

$$T(v;\Theta_m) = \sum_{j=1}^{J} \gamma_{jm} I(v \in R_{jm}) \tag{6}$$

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}$ encodes the parameters of the m-th regression tree having terminal regions $R_{jm}, j = 1, \ldots, J$, and the indicator function I is defined as

$$I(v \in R_{jm}) = \begin{cases} 1 & \text{if } v \in R_{jm}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

It can be seen from (6) that the response variable y is modeled as a constant within every tree region R_{jm}. Using the additive form, the overall boosted tree can then be written as the sum of M regression trees

$$f_M(v) = \sum_{m=1}^{M} T(v;\Theta_m) \tag{8}$$

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

$$R_1 = \{v \mid v^o \leq s\}, R_2 = \{v \mid v^o > s\} \tag{9}$$

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

$$\min_{o,s}[\min_{\gamma_1} \sum_{v_k \in R_1} (r_k - \gamma_1)^2 + \min_{\gamma_2} \sum_{v_k \in R_2} (r_k - \gamma_2)^2] \tag{10}$$

where v_k is the predictor variable vector corresponding to LP_k and r_k is the target value in each tree region. r_k can be either chosen as the response variable itself or 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 (10) is solved by

$$\hat{\gamma}_1 = \frac{\sum_{v_k \in R_1} r_k}{N_1}, \hat{\gamma}_2 = \frac{\sum_{v_k \in R_2} r_k}{N_2} \tag{11}$$

where N_1 and N_2 denote the number of training data points in R_1 and R_2 respectively. Each regressor 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 [8] to reduce the number of leaf nodes to J.

In order to prevent overfitting, we consider a slightly modified version of (8) in an iterative form as

$$f_m(v) = f_{m-1}(v) + \nu \sum_{j=1}^{J} \gamma_{jm} I(V \in R_{jm}), m = 1, \ldots, M \tag{12}$$

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

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{v_k \in R_{jm}} L(y_k, f_{m-1}(v_k) + \gamma) \tag{13}$$

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 that is given by [8]

$$L_H = \begin{cases} [y - f(v)]^2, & \text{if } |y - f(v)| \leq \delta, \\ 2\delta(|y - f(v)| - \delta^2), & \text{otherwise.} \end{cases} \tag{14}$$

Let us now present a couple of well-known concepts before discussing the modifications.

Definition 1. A p-dimensional *convex polytope* (or p-polytope) is defined as the bounded solution set (intersection) of a finite system of linear inequalities that represent closed half-spaces:

$$P = \{x \in \mathbb{R}^p \mid Ax \leq b\} \tag{15}$$

Remark 2. The solution of an LP problem as given by (1-2) lies on the boundary of a convex polytope whose facets are defined by the linear inequality constraints. The corresponding polytope is known the *feasible region* of the given LP problem.

In the Boost-LP algorithm, we consider a modified form of the standard loss functions that heavily penalizes selection of any γ_{jm} that lies outside the common feasible region of all the given LP problems for which v lie in R_{jm}. We refer to this as the *penalization loss* L_p and represent it as

$$L_p = L(y, f(v)) + h\gamma'_{jm} \tag{16}$$

Here, h is a very large positive number and $\gamma'_{jm} = \{y : y \notin P_c\}$, where P_c is the common feasible region of all the LPs whose $v \in R_{jm}$. Although this modification cannot guarantee generation of feasible solution for any new LP problem, it significantly increases the possibility of doing so (shown in Sect. 5.1). Infeasibilities are detected by looping through all the constraints and are handled by discarding the predicted values and actually solving the LPs using any standard optimizer.

If the range of d_i is large in the training data set, then more problems need to be generated so that solutions for additional values of those widely distributed parameters are utilized for learning. If the absolute difference between two consecutive values of $d_i, 1 \leq i \leq n$, is greater than a pre-defined threshold ζ_i, a minimum number of d_i^s are added with uniformly spaced values in between those points such that the new value of the absolute difference is less than or equal to ζ_i. Henceforth, N refers to the total number of training data points that includes both the original as well as the newly added ones.

Now, in order to generate complete LP problems corresponding to the newly created d_i values, we select the average values of the two original extreme points for all the other components of v excepting d_i. Although this directly generates the c vector, it does not yield a unique solution for A' and b'. This is taken care of by selecting the A' matrix and the corresponding W matrix for the lower d_i valued point and then solving for b' using (4). It may be noted that different threshold values can be selected for the different components of d and this choice is domain-dependent.

4 Theoretical Results

In this section, we present an absolute performance guarantee on the overall LP objective function in the form of Theorem 8. This result is based upon the performance guarantee for the individual decision variables (response variables of the Boost-LP algorithm) that is obtained in Lemma 7. The result also utilizes a geometric property (given by Remark 4) of the arrangement created by the boosting trees that is described in Lemma 5. Lemma 7 again builds upon the result derived in Lemma 6 on the existence of a non-negative vector that provides an upper bound on the difference between the vector d (given by (4)) and the response variable for any training set problem and uses Lemma 5.

Definition 3. 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. It may be noted here that R_p is the usual outer radius or circumradius.

Remark 4. Let B be a p-dimensional box (or p-box) in Euclidean space E^p with dimensions $2a_1 \times 2a_2 \times \ldots \times 2a_p$, such that $\{x \in E^p \mid -a_i \leq x_i \leq a_i, i = 1, \ldots, p\}$. It can be assumed without any loss of generality that $0 < a_1 \leq a_2 \leq \ldots \leq a_p$. Then R_j is simply given by [3]

$$R_j = \sqrt{a_1^2 + \ldots + a_j^2} \tag{17}$$

Lemma 5. Boosting trees transform the LP problems present in the training data set to an arrangement of higher-dimensional, axis-aligned boxes.

Proof. Combining the 2nd system of inequality in (2) and (4), it can be written that

$$0 \leq d < \infty \tag{18}$$

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 predictor variable space. The RHS inequality in (18) is required to prevent the decision variables from assuming infinite values, resulting in unbounded LP solutions (from (1)).

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 will 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.

□

Lemma 6. If W_k is chosen as E'_k (Condition 1), then there exists a vector ϵ with all non-negative components such that $d_{i,k} - y_k \leq \epsilon_i, 1 \leq i \leq n$, for any value of k and ϵ is solely a function of the system parameters, i. e. A_k, b_k, and c_k of the LP problems present in the training data set.

Proof. Irrespective of whether $m_k \leq n$ or $m_k > n$ in any given problem LP_k, the number of active constraints m'_k at the optimum solution x^*_k cannot exceed n, as otherwise the system will not have any solutions. Thus, the first system of inequalities in (2) can be reduced to

$$A'_k x^*_k = b'_k \tag{19}$$

where $A'_k \in \mathbb{R}^{m'_k, n}$ and $b'_k \in \mathbb{R}^{m'_k}$.

The general solution x_k of the above system of equations is given by Remark 2.1. in [7] as:

$$x_k = x_{m'_k+1} + H^T_{m'_k+1} q_k \tag{20}$$

where $x_{m'_k+1}$ is the exact solution of (19) and $q_k \in \mathbb{R}^n$ is an arbitrary vector. The *Abaffian* matrix $H_{m'_k+1}$ is again given by

$$H_{m'_k+1} = I - A'^T_k (E'_k A'^T_k)^{-1} E'_k \tag{21}$$

Now, from the basic definition of general solution, x_k will lie in the feasible region of LP_k, and without any loss of generality we can assume that $x_{m'_k+1}$ represents x^*_k. Based on this observation, (20) and (21) can be combined to yield

$$\begin{aligned} x_k &= x^*_k + (I - A'^T_k (E'_k A'^T_k)^{-1} E'_k)^T q_k \\ &= x^*_k + (I - E'^T_k (E'^T_k A'_k)^{-1} A'_k) q_k \end{aligned} \tag{22}$$

Hence,

$$\begin{aligned} q_k &= (x_k - x^*_k)(I - E'^T_k (E'^T_k A'_k)^{-1} A'_k)^{-1} \\ &= (x_k - x^*_k)(I + E'^T_k (E'^T_k A'_k + A'_k E'^T_k)^{-1} A'_k) \quad \text{from the Woodbury formula}^1 \\ &\leq \varepsilon_k (I + E'^T_k (E'^T_k A'_k + A'_k E'^T_k)^{-1} A'_k) = \eta_k \end{aligned} \tag{23}$$

as for any given LP problem, $|x_k - x^*_k| \leq \varepsilon_k$, since the feasible region is bounded.

[1] Also known as the Sherman-Morrison-Woodbury formula or the matrix inversion lemma

Since $d_k = (E_k'^T A_k')^{-1} E_k'^T b_k'$ from (4), we have,

$$(E_k'^T A_k')^{-1} = d_k[(b_k'^T E_k')[(E_k'^T b_k')(b_k'^T E_k')]^{-1}] \tag{24}$$

Abbreviating $(b_k'^T E_k')[(E_k'^T b_k')(E_k' b_k'^T)]^{-1}$ as F_k' and substituting this in Equation (22), we have,

$$x_k = x_k^* + (I - E_k'^T d_k F_k' A_k')q_k \tag{25}$$

Rearranging terms and using basic matrix operations, we get,

$$d_k = [(E_k' E_k'^T)^{-1} E_k'][q_k - (x_k - x_k^*)][(q_k^T A_k'^T F_k'^T)((F_k' A_k' q_k)(q_k^T A_k'^T F_k'^T))^{-1}] \tag{26}$$

Now, since $q_k - (x_k - x_k^*) \leq \eta_k + \varepsilon_k$, the above equation can be re-written as

$$d_k \leq [(E_k' E_k'^T)^{-1} E_k'](\eta_k + \varepsilon_k)[(q_k^T A_k'^T F_k'^T)((F_k' A_k' q_k)(q_k^T A_k'^T F_k'^T))^{-1}] = \widetilde{\epsilon_k} \tag{27}$$

The above equation holds as all the matrix product terms in the RHS are non-singular; the rank of the Gramian matrix $E_k' E_k'^T$ is same as that of E_k', which again must be equal to m_k' from its basic construction.

Here, $\widetilde{\epsilon_k}$ solely depends on the LP problem parameters A_k, b_k, and c_k as A_k' and b_k', and, hence, E_k', F_k', η_k, and ε_k all depend on those parameters. We can re-write the vector (27) in its component scalar form as

$$d_{i,k} \leq \widetilde{\epsilon_{i,k}}, 1 \leq i \leq n \tag{28}$$

Since the boosting tree response variable $y_k = x_{i,k}^*$ is constant for any component of x in a given data set, we can re-formulate the above equation as

$$d_{i,k} - y_k \leq \epsilon_i, \text{ for any } k \text{ and } 1 \leq i \leq n \tag{29}$$

where $\epsilon_i = \max\{\widetilde{\epsilon_{i,1}} - x_{i,1}^*, \ldots, \widetilde{\epsilon_{i,N}} - x_{i,N}^*\}$. Thus, ϵ_i is just a function of the system parameters A_k, b_k, and c_k as both $\widetilde{\epsilon_k}$ and the solution x_k^* depend only on them. Moreover, $\epsilon_i \geq 0, 1 \leq i \leq n$ from (4), and, hence, the lemma follows.

□

The absolute performance guarantee of the Boost-LP algorithm is measured in two ways: firstly, by the maximum error that can occur in inferring an optimum $y^* = x_i^*, i = 1, \ldots, n$ value for any arbitrarily new LP problem that is not present in the training data set and, secondly, by the maximum error incurred in computing the optimum $z^* = c^T x^*$ value.

Lemma 7. If x_i^e denotes the predicted value of x_i^* for any arbitrary LP problem with parameters A, b, and c, then $|x_i^* - x_i^e| \leq (N - MJ + 1)\zeta_i + \epsilon_i$, where $N \geq MJ$ and ϵ_i is obtained from Lemma 6.

Proof. From (6) and (8), it is evident that the maximum error in predicting any x_i^* will arise when the corresponding v vector lies in a region R_{jm}^{crit} such that the worst-case error in predicting the response variable by a constant value γ_{jm}^{crit} is the highest among all the tree regions. Mathematically speaking,

$$|x_i^* - x_i^e| = |y^* - f_M(v)| \leq |y^{crit} - \gamma_{jm}^{crit}| \tag{30}$$

where

$$y^{crit} = \arg\max_{y} |y - \gamma_{jm}^{crit}| \tag{31}$$

Clearly, at least one training data point needs to be present in every tree region. Since the total number of regions is equal to MJ, in the extreme case it may so happen that $(MJ-1)$ regions all have one data point and one region has all the remaining $(N-MJ+1)$ points. Now, if we look at the nature of the target values, r_k can be either equal to y_k, or it can be equal to $(y_k - f_m(v_k))$ when the negative gradient of the L_2 loss functional is taken, or it can be equal to ± 1 when the L_1 loss functional is used. r_k can also be equal to $\pm\alpha_t$ when the t-th quantile is used for the Huber loss functional.

Thus, it can be seen that child regions are iteratively created according to (9) and (10) to cluster the response variable based on similar values, similar deviations from predictions (γ_{jm}), 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 5) 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 r_k. Without any loss of generality, let us assume that this sorting is being done in a non-decreasing order.

So, if we have a region containing $N_{jm} = (N - MJ + 1)$ data points, the algorithm will model $y_l, \ldots, y_{l+N-MJ}$ response variable values by a single γ_{jm} where $1 \leq l \leq MJ$, such that $y_l \leq \ldots \leq y_{l+N-MJ}$. Now from Lemma 6, as $(d_{i,l} - y_l) \leq \epsilon_i$ when $W = E'$ for any value of l, for this region we have $\gamma_{jm}^{crit} \geq d_{i,l} - \epsilon_i$ and $y^{crit} \leq d_{i,l,N-MJ+1}^{max} = \max\{d_{i,l}, \ldots, d_{i,l+N-MJ}\}$. Since, $|d_{i,j+1} - d_{i,j}| \leq \zeta_i, 1 \leq j \leq N-1$, we have,

$$y^{crit} \leq d_{i,l} + (N - MJ + 1)\zeta_i \tag{32}$$

As it can be easily seen that $d_{i,l,N_{jm}}^{max} \geq d_{i,l,N'_{jm}}^{max}$ for any value of $N_{jm} > N'_{jm}$, we can re-write (30) as

$$\begin{aligned} |x_i^* - x_i^e| &\leq (N - MJ + 1)\zeta_i + \epsilon_i \\ &= d_{i,crit} \end{aligned} \tag{33}$$

Thus, the lemma follows.

□

From (33), it can be seen that the maximum error in predicting any x_i occurs in a boosting tree terminal region R_{jm}^{crit} such that the corresponding d_i lies within a fixed range given by $[d_{i,l^{crit}} - \epsilon_i, d_{i,l^{crit}} + (N - MJ + 1)\zeta_i]$, where $1 \leq l^{crit} \leq MJ$ represents the training data point corresponding to $d_{i,crit}$. This implies that R_{jm}^{crit} represents a p-box in v space, wherein d_i can only assume certain values and all the other predictor variables ($c_j, 1 \leq j \leq n$ and $d_j, 1 \leq j \leq n, j \neq i$) can take any values within its overall range given by the minimum and maximum values that occur in the training data set.

Theorem 8. If x^e denotes the predicted x^* for any arbitrary LP problem with parameters A, b, and c such that the objective function values are given by z^e and z^* respectively, then $|z^* - z^e| \leq \|c\|_2 \sqrt{\Sigma_{i=1}^{n}[(N - MJ + 1)\zeta_i + \epsilon_i]^2}$, where $N \geq MJ$, ζ_i is given by Lemma 7, and ϵ_i is obtained from Lemma 6.

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

$$|z^* - z^e| = |c^T x^* - c^T x^e| = |c.(x^* - x^e)| \tag{34}$$

Using Cauchy-Schwarz inequality, the error can be bounded by

$$|z^* - z^e| \leq \|c\|_2 \|x^* - x^e\|_2 \tag{35}$$

Based upon our discussion of Lemma 7, if we consider the maximum prediction error for all the components of x, then we can find n number of p-boxes, each constrained only in terms of a particular d_i. Thus, the maximum error in predicting the entire vector x will occur when c, A, and b are such that the generated v lies in R_{jm}^{crit} of each individual boosting tree. So, if we take the intersection of all the critical regions corresponding to this worst-case scenario in v space, we obtain a new p-box B^{crit} where all the components of d must lie within well-defined ranges as given by $d_{i,crit}$ and c is fixed.

From Definition 3, 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}. Utilizing this observation, we can combine (17) and (33) and substitute the value for the 2nd L_2 norm in the RHS of (35) to have

$$\begin{aligned} |z^* - z^e| &\leq 2\|c\|_2 \sqrt{(\frac{d_{1,crit}}{2})^2 + \ldots + (\frac{d_{n,crit}}{2})^2} \\ &= \|c\|_2 \sqrt{d_{1,crit}^2 + \ldots + d_{n,crit}^2} \end{aligned} \tag{36}$$

Substituting for $d_{i,crit}$, we obtain the final form

$$|z^* - z^e| \leq \|c\|_2 \sqrt{\Sigma_{i=1}^{n}[(N - MJ + 1)\zeta_i + \epsilon_i]^2} \tag{37}$$

It may be noted here that the monotonically increasing assumption of box widths in (17) is implicitly taken care of by considering all the box dimensions that have non-zero widths. This concludes the proof of the theorem.

□

Thus, from (37), it can be inferred that the LP solution performance guarantee can be improved in two ways. This can be done by building a more complex boosting tree model with higher number of regression trees (M) or terminal regions in a tree (J), or may be both. This will increase the product factor MJ, thereby, decreasing the maximum possible number of training data points $(N - MJ + 1)$ that are modeled by a constant response value in the critical region. However, this may not be desirable as it will increase the inference time. The other alternative is just to select a smaller value of $\zeta_i \forall i$ such that the overall LP prediction error is always less than a certain pre-defined value up to a certain limit defined by ϵ_i. This is evident from the cross-validation errors obtained in the next section.

5 Experimental Results

All the results presented in this section are obtained on a Quad-Core Intel Xeon CPU, having 2.93 GHz processor speed 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 as the optimization solvers. M, J, and ν are always chosen to be 500, 8, and 0.1 respectively.

5.1 Aircraft Carrier Deck Operations

Our first problem is that of scheduling, where the objective function is to minimize the overall weighted time taken by a set of aircrafts (both manned and unmanned) to complete a given set of tasks in naval carrier operations. Weights are assigned based upon the aircraft priorities. For example, consider a set of missions that need to be accomplished by the aircrafts without violating any of the constraints in terms of the resources (landing strip, launching catapults, weapons elevators, deck parking spaces etc.) that are available to accomplish any particular task at any point of time. Certain other constraints like fuel emergencies and flight times beyond the permissible limit for a day can force an aircraft to land sooner than expected. The scheduler must then construct a new plan to reduce the ensuing disruption in the existing schedule by again minimizing the time taken to complete the set of tasks with newly enforced deadlines and priorities. A screenshot of the carrier simulation environment is shown in Fig. 1.

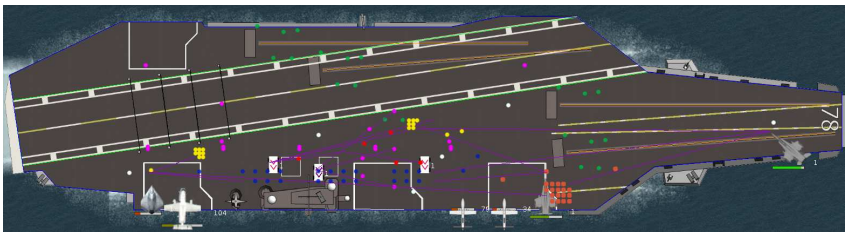

Fig. 1. Screenshot of aicraft carrier deck simulation environment

In order to handle uncertainties corresponding to the various failure modes, we cast the problem in a stochastic integer linear programming (SILP) form. Currently, we consider uncertainties in resource availability, successful task completion (different for each aircraft), and completion rates. A disjunctive formulation of the generalized Benders' decomposition approach is adopted from [14] to generate the training data set.

Table 1 presents a comparison of the average (taken over all the response variables) 10-fold cross-validation error in predicting the solutions of all the relaxed LP problems in the BAB nodes of the SILP problem corresponding to

an operation involving 20 aircrafts (with identical priorities), 9 possible tasks, a scheduling time horizon of 150, and 5 different failure modes using a number of popular regression approaches and loss functions. The number of predictor variables p is equal to 54,000 and the cardinality of the training data set N is equal to 5,764. No additional training data points are generated and identical ζ vector is chosen for all the approaches. Boosting tree regression significantly outperforms all the other regression approaches irrespective of the nature of the loss function being used. Although using polynomial (quadratic) regression improves the performance both with and without support vector machines (SVM) as compared to ridge regression, it is difficult to determine the most suitable choice for the polynomial order without evaluating all of them explicitly on any given problem.

Table 1. Average cross-validation error (in %) using different regression approaches

Approach	L_2	L_1	L_H	L_p
Ridge	10.4	11.1	10.9	11.5
Polynomial	7.9	8.2	8.0	8.3
SVM + ridge	10.5	11.1	10.8	11.4
SVM + polynomial	8.0	8.2	8.1	8.3
Boosting tree	1.9	2.2	2.1	2.3

Table 2. Performance evaluation of different boosting tree approaches with and without infeasibility avoidance and addition of new training data points (COIN-OR solver takes 1032.14 s of computation time)

	Performance metric	Infeasibility not avoided		Infeasibility avoided	
		L_H	L_p	L_H	L_p
Training data points not added	SILP prediction computation time (in s)	14.76	15.83	17.46	18.36
	Speed-up	70X	65X	59X	56X
	Average cross-validation error (%)	2.1	2.3	2.0	2.1
	SILP solution estimation error (%)	4.2	4.0	4.1	3.9
	No. of infeasible solutions[2](%)	7.1	1.2	0.0	0.0
Training data points added	Average cross-validation error (%)	1.0	1.1	0.9	1.0
	SILP solution estimation error (%)	3.3	3.5	3.1	3.2
	No. of infeasible solutions (%)	6.7	0.9	0.0	0.0

Table 2 evaluates the performance of the boosting tree algorithm without and with avoidance of infeasibility (by using the modified loss function given by (16)) and addition of new training data points for the same scenario as in Table 1. The same problem is used both for training and testing. As expected, the corresponding values for the average cross-validation and solution estimation 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 in the two cases as inference is done on models having identical complexities and hence the values are shown only once in the table. By selecting ζ appropriately during the learning phase, the cross-validation and solution estimation errors are kept below pre-defined thresholds of 1.5% and 4.0% respectively when additional training points are generated. It can also be seen that using penalization loss L_p significantly reduces the percentage of generated infeasible LP solutions as compared to the standard Huber loss function L_H. Even though this issue is avoided completely by actually solving the relaxed LP problem when the boosting tree generates an infeasible value during the inference phase, it comes at the expense of decreasing the speed-up factor.

5.2 Vehicle Routing with Time Windows

Our second example domain is taken from [15], 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 adopt the IP-based set partitioning formulation of the problem given in [5].

Table 3. Performance evaluation on data sets with varying modes of time window constraint generation

Performance metric	C1	C2	R1	R2	RC1	RC2
COIN-OR computation time (in s)	283.58	578.98	307.46	883.10	144.72	312.05
Boost-LP computation time (in s)	4.75	9.78	5.22	15.07	2.40	5.28
Speed-up	60X	59X	59X	59X	60X	59X
Average cross-validation error (%)	2.0	2.1	2.4	2.4	2.2	2.3
ILP solution estimation error (%)	4.2	4.3	4.9	4.9	4.5	4.6
No. of infeasible solutions[3](%)	1.4	1.4	1.6	1.7	1.5	1.5

[2] Non-zero values do not imply that Boost-LP returns infeasible solutions; rather infeasible solutions are detected and the corresponding LPs are solved using COIN-OR - table entries show the cost and benefits of performing this additional step

Table 3 summarizes the results for 6 benchmark data sets, each containing 100 customers, using the Boost-LP algorithm without performing infeasibility avoidance. Identical data sets were used in [15] and [5] and we obtained them online at `http://w.cba.neu.edu/~msolomon/problems.htm`. The 2D coordinates of the customers and the time windows are generated in a clustered way in the data sets C1 and C2, randomly using uniform distributions in the data sets R1 and R2, and in a semi-clustered way in the other two data sets RC1 and RC2. The data sets C2, R2, and RC2 also have longer scheduling horizons (allowing more than 30 customers per vehicle) as compared to C1, R1, and RC1 respectively (permitting 5-10 customers per vehicle). Each data set contains (8-12) separate problems with different proportion of customers having time windows. We use the last problem in every data set for testing the models learnt over the LP problems occurring in the BAB nodes of all the other IP problems present in that particular data set. It can be seen that although best performance is obtained for the data set C1 and worst for R2, the change in performance quality from one data set to another is relatively small. Thus, our approach is capable of producing satisfactory results for problem domains with varying characteristics.

5.3 Vehicle Planning and Control

Our final problem domain is taken from robotics, where 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 radius. While this motion planning problem can be solved in many different ways, framing it as a mixed integer linear programming (MILP) problem is common in the controls community due to the ease of incorporating complex, time-varying dynamics [2]. We compare the results for predicting the relaxed LP problem solutions in the minimum-control-effort trajectory-generation problem with those obtained by directly using the COIN-OR solver within the framework of the iterative MILP time-step selection algorithm presented in [6]. 50 random instances of the problem are generated with the same parameter ranges as given in [6] and the last 10 instances are used for testing whereas all the others are utilized for learning.

Table 4 shows that the average cross-validation error, average solution estimation error (taken over the 10 test problems), and the average percentage of infeasible solutions increase non-linearly with the number of obstacles. Even though different values of ζ are chosen, ϵ increases with additional constraints in the form of obstacles such that the overall prediction error cannot be maintained exactly at the same level. Interestingly though, the average speed-up factor (again taken over the 10 test problems) 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. We want

[3] We intentionally suppress solving actual LPs just to show that even otherwise our algorithm predicts infeasible solutions for few problems

Table 4. Performance evaluation on data sets with varying number of obstacles

Performance metric	Number of obstacles			
	5	10	15	20
Average COIN-OR computation time (in s)	20.27	48.25	196.98	1075.68
Average Boost-LP computation time (in s)	0.49	0.82	2.49	11.01
Average speed-up	41X	59X	79X	98X
Average cross-validation error (%)	0.2	0.5	1.1	2.0
Average MILP solution estimation error (%)	0.5	1.3	2.7	5.0
Average no. of infeasible solutions (%)	0.1	0.5	1.2	2.1

to add here that if the tree complexity is enhanced by increasing M and J to keep the prediction errors nearly constant in all the cases, then this reduction in exponential growth rate is no longer observed. Thus, for a given training data set size, if additional constraints are introduced in the system, then there is a trade-off between computational benefits and prediction accuracy.

6 Conclusions

In this report, we modify the standard boosting tree technique to develop an approach, which we term as the Boost-LP algorithm, for learning the solutions of similar LP problems. We provide absolute performance guarantees for both the individual decisions variables as well as the overall LP solution. The expressions clearly show that an user-defined vector can be adjusted to achieve better performance up to a certain limit that is governed by the LP system parameters. We conduct experiments in three different problem domains to highlight the fact that the proposed algorithm is capable of inferring the solutions of ILP, MILP, and SILP problems in a fraction of the time required by standard optimization procedures, without significantly compromising the solution accuracy.

Directions for future work include investigating theoretical bounds on the number of infeasible solutions and exploring the scalability of our approach in inferring solutions of problems consisting of higher-dimensional decision variable vectors. We also intend to quantify the robustness of our algorithm to variations in the training data set parameters and validate it on more complex planning domains consisting of dynamics constraints and polygonal obstacles.

References

1. J. E. Bell and P. McMullen. Ant colony optimization for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48, 2004.
2. J. Bellingham, Y. Kuwata, and J. How. Stable receding horizon trajectory control for complex environments. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Austin, Texas, USA, 2003.

3. R. Brandenberg. Radii of regular polytopes. *Discrete & Computational Geometry*, 33(1):43–55, 2005.
4. COIN-OR. `http://www.coin-or.org/index.html`.
5. 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.
6. M. G. Earl and R. D' Andrea. Iterative MILP methods for vehicle-control problems. *IEEE Transactions on Robotics*, 21(6):1158–1167, 2005.
7. H. Esmaeili, N. Mahdavi-Amiri, and E. Spedicato. Explicit ABS solution of a class of linear inequality systems and LP problems. *Bulletin of the Iranian Mathematical Society*, 30(2):21–38, 2004.
8. T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2008.
9. 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.
10. 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.
11. 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.
12. V. Nguyen, B. Steece, and B. Boehm. A constrained regression technique for COCOMO calibration. In *Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, 2008.
13. T. Orestis and S. Panagiotis. Combinatorial optimization through statistical instance-based learning. In *Proceedings of the 13th IEEE International Conference on Tools in Artificial Intelligence*, Dallas, Texas, USA, 2001.
14. S. Sen and H. D. Sherali. Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming:Series A and B*, 106(2):203–223, 2006.
15. M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.
16. 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.
17. 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.