Learning from Feasible Solutions to Optimisation Problems

Kelvin Davis

Abstract

In this study we hope to show whether or not it is possible to learn where to find good solutions for combinatorial optimisation problems. Many studies have looked at using machine learning as a means of finding better approximations for the components of the algorithms that solve these problems, but so far, no study has considered the bounds of possible information. The proposed experiment will show how much structural information can be extracted from the solutions space of optimisation problems under total omniscience and then we will show whether the information can learned without omniscience.

Introduction

Solving discrete optimisation problems can be a computationally intensive task. Much of the difficulty is knowing where to look for good solutions. In the past decade, research has been done in the way of using machine learning to improve the components of state of the art enumeration methods that solve these problems. However there are limits to how much information can be learned from these problem. In this review, we will explore what kind of learning techniques have been utilised in the field of discrete optimisation and why research needs to be undertaken to explore the bounds of how much information can be learned from these problems.

Discrete optimisations are ubiquitous. They exist in problems of trying to maximise profit when ordering stock, scheduling, and the knapsack problem.

The lot-sizing problem is an excellent example of a discrete optimisation problem solved every day. Let us say that a store owner wants to order stock for their products. The question posed in the lot-sizing problem is: how many of each item should we order in order to minimise the total cost from purchasing the products and holding the products.

In general, a discrete optimisation is any problem where we want to find a set of values for variables that minimises (or maximises) some objective function $f(\mathbf{x})$, such that it satisfies some constraints . In the lot-sizing problem, the objective function that we are trying to minimise is the total cost and the variables that we have control over are the number of each item that will ordering.

The difficulty of these problems

These problems are difficult in that each variable that you add to the problem multiplies the number of total possible feasible solutions by some factor. As another example, in solving the knapsack problem, you are trying to find the subset of items that has the greatest total monetary value but whose total weight is below below some real positive value. For each item that you take into account, the total number of subsets to consider doubles. For 10 items, there would be 1024 possible combinations to consider, but for 100 items there are 2^{100} possible solutions to consider which is more than the total number of stars in the known observable universe .

To get around this scale in complexity, many different approaches have been taken and are utilised in state of the art solvers today . These approaches are in branching and pruning, searching, removing constraints, propagating constraints, decompositions, improving approximations, and so on. The field of discrete optimisation is vast.

The last decade has seen an explosion in Machine learning research and it has brought along uses in improving the components of these solvers. Machine learning is the process of tuning parameters of a predictive model to find the best fit of those parameters through the training data. These models can used to act as approximate functions for other functions . There are multiple paradigms of machine learning from 2Super Much of the research focuses on the improvements in performance

Introduction

Solving discrete optimisation problems can be a computationally intensive task. Much of the difficulty is knowing where to look for good solutions.

Research has been done that utilizes adaptive strategies and to choose where to branch within the solution spaces of these discrete optimisation problems. Some of these methods use techniques from machine learning to guide the branching, searching, and bounding strategies. So far the studies on these adaptive strategies have not considered learning with respect to the total amount of information that can be extracted from these problems.

The Branch and Bound algorithm is a method for solving these problems that splits the solutions by branching on a particular variable at a time. In this project we will investigate the amount of information that can be learned about the which variables to branch on with and without complete prior knowledge of all feasible solutions in the solution space of a discrete optimisation problem.

In the research following from this review we will explore whether or not there exists information in the feasible solutions that can be exploited by a Branch and Bound tree. We hope that the research inspires new approaches to branching that utilise this information.

Background

Optimisation

Optimisation is at the heart of many of life's problems. It rears its head in problems like price setting, ordering stock, scheduling tasks, hunting for bargains, fleeing the Cave of Wonders and deciding which treasures to pack in into your Knapsack (Conforti, Cornuéjols, and Zambelli 2014, vol. 271, chap. 2; also Clements and Musker 1992). In this project, we choose to focus on discrete linear optimisation problems, which can be formulated as Mixed Integer Programming (MIP) problems.

These kinds of optimisation problems take the form $z = \mathbf{c}^{\mathrm{T}}\mathbf{x}$ such that $\mathbf{x}$ satisfies the linear constraints $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$. Here, z is some variable that we want to maximise (commonly referred as the objective value), $\mathbf{x} = (x_1, x_2, ..., x_n)$ is the vector of variables that we have control over where some of the variables are constrained to the integers, and $\mathbf{c} = (c_1, c_2, ..., c_n)$ is a vector of linear coefficients. The constraints are rules that determine what values our controlled variables can take.

Mixed Integer Programming problems are Linear Programming problems with integer constraints on some of the variables. Linear Programming problems are commonly solved using the simplex method or interior point method (Conforti, Cornuéjols, and Zambelli 2014), which is able to solve them efficiently by finding an optimal solution at one of the extreme points of the solution space. Here, we define the *solution space* as the set of all feasible solutions to the problem, the extreme points of which exist at the intersections of various constraints. However, when the feasible solutions are constrained to be integers, the geometry of the solution space cannot be as easily exploited by an algorithm like Simplex, as those extreme points may not be integer solutions.

The Class of Branch and Bound Algorithms

MIP problems belong to the class of NP-Hard (Conforti, Cornuéjols, and Zambelli 2014), which means that any problem in the class of NP can reduced to a MIP problem in polynomial time (Du and Swamy 2016, 16–17).

The general method for solving NP-Hard optimisation problems formulated as MIPs is called Branch and Bound, which is not a single algorithm, but a class of algorithms; all of which, as the name implies, have two key components (Clausen 1999):

1. **Branching** - splitting the solution space into smaller pieces
2. **Bounding** - calculating the lower/upper bound

In general, The Branch and Bound algorithm is an implicit enumerative search that explores a tree of subspaces by iteratively splitting a solution space into smaller subspaces, evaluating an upper bound (or lower bound for a minimisation problem) of the objective value for each of these subspaces, removing any subspaces whose bounds, by comparison against the optimal solution thus far, indicate that they are not worth considering, and then repeating this process with the next best subspace in the tree. Whenever feasible solutions are found by the algorithm, their objective values are checked against that of the current optimal solution, and the current optimal solution is updated if the new feasible solution is more optimal.

In the algorithm described thus far, the only information that is collected in this process is information on what the current optimal solution is; all other solutions are not used to guide the search, besides informing the algorithms of what has been searched before. No information is used regarding the values of the variables at those solutions is used. Infeasible solutions and infeasible subspaces inform the algorithm of where not to look.

All in all, information is collected about what has not been explored, what has been explored and what the lower and upper bounds are. We hope to show that more information exists in the structure of the feasible solutions and that this structure can be exploited.

As shown in the Literature Review section, adaptive strategies have been developed that do use information obtained from previous solutions, However, no approach has been considered as of yet that uses information collected from the whole solution space a priori.

Information

So how do we measure information? Informally, Chris Wallace (2005) defines information as "something the receipt of which reduces our uncertainty of the world".

In this respect, information about where the optimal solutions exist would reduce uncertainty about what objective values we could expect to find in different areas of the solution space.

We refer to uncertainty as the degree to which we would be unable to predict the outcome from a random process. Shannon Entropy is an excellent measure for this uncertainty.
Shannon defines entropy from a discrete process Y as $H = -\sum_i p_i \log p_i$, where $p_i = \mathbb{P}(Y = y_i)$ is the probability that the event y_i occurs (Shannon 1948).

There are many other methods to measure this uncertainty (as provided by Scikit-learn); including Gini, a measure commonly used for income inequality (Yitzhaki and Schechtman 2013); and metrics that represent the uncertainty in sampling from a continuum like Variance, used to measure the spread of a distribution; Mean Absolute Error, Mean Squared Error and Friedman Mean Squared Error (Hastie, Tibshirani, and Friedman 2009).

Information Gain provides a comparison of the uncertainty before and after splitting on a given variable. Information Gain (denoted IG) can be measured using

$$IG(Y, x) = I(Y) - \sum_i \mathbb{P}(x_i) I(Y|x_i),$$

where Y in our case is the set of objective values, I the particular criterion (entropy, gini index, etc.) used, x is the variable that we are splitting on, x_i is a unique element of x, and $P(x_i)$ the probability of x_i occurring in x.

Branching

In the exploration of each node within the Branch and Bound tree, an upper bound (of a maximisation problem) is obtained from the solution of the LP relaxation. If any of the variables within the solutions are fractional, the MIP problem represented by that node is split into two or more smaller sub problems. The naive approach to choosing a variable to branch on, as defined in the seminal paper (Land and Doig 1960), is to choose the variable whose value from the solved LP relaxation is the furthest from an integer. As shown by (Achterberg, Koch, and Martin 2005), this method can be shown to have worse performance then simply picking a variable to branch on at random. Two of the main approaches to branching are *strong branching* and psuedocost branching. Strong branching chooses a variable to branch on by testing if not all of the fractional variables then at least a subset of those fractional variables buy solving the LP relaxation is of the child nodes generated by branching on that variable, and evaluating a score based on the games in the bounce. Pseudocost branching, on the other hand, keeps track of the change in objective function of the solution the LP relaxation when a variable had been branched on in past exploration in the tree. The future costs can be defined as follows <–will define here–>. This branching rule can be quite uninformative at the start of the search and so strong branching is often performed toward the start of the search to get some history to go off of . <–introduce reliability branching as a means of finding the balance of pseudocost and strong–> Lookahead branching takes strong branching a level further by looking at not only the LP gains of the child nodes but of the grandchild nodes. In 2014, Gilpin and Sandholm developed branching methods that reduced the uncertainty about the value of the variables at the optimal solutions by computing the entropy of each variable and branching on the variable with the greatest changing entropy. However they use the fractional component of the solution found by solving the LP relaxation as the probability that the entropy is based on. This implies that the entropy that they are computing and the subsequent information that they are gaining from branching on those variables is not collected from the feasible solutions but rather from the fractional components of the LP relaxation.

The use of machine learning

Now on to decision trees. In our research, we are wanting to find branching strategies that exploit information found in the feasible solutions of MIPs. This is not so different from the training process of decision trees. The goal in training a decision tree is to find the best ways to split the dataset such that the uncertainty of the result is reduced. There are many different measures of uncertainty that can be used in the decision tree model like variance or entropy but the underlying process remains the same. In this light a branch and bound search tree can be likened to a decision tree where the decisions are which variables to branch on and the results are the objective values at those solution.

Support Vector machines may also be useful in this regard although less obviously. A support Vector Machine classifier also splits a data set to reduce uncertainty about the outcome but only one split is made using a Kernel that combines the variables in a linear or nonlinear way depending on which one you choose. This can be likened to that of a disjunctive cut <–reference to dijunctive–>. For more information about Machine Learning, we refer the reader to (Kotsiantis 2007).

Machine learning has been used extensively in the field of discrete optimisation to improve the components of branch and bound search. In 2014, He et al. used imitation learning to train mode selection strategies. To train these strategies they words observe the behaviour from an Oracle that new where the optimal solution was before training. In 2016, Khalil et al. developed a framework to learn an approximation algorithm for strong branching which would be more efficient to compute during the branch and bound search. As features, the learning algorithm used the coefficients of the objective function, the number of constraints, and statistics pertaining to the constraint degrees i.e. the number of variables that participated in a constraint the statistics included mean standard deviation minimum and maximum. Comment the following out These features don't necessarily provide information about the variables from the feasible solutions In a more recent paper Balcan et al. (Balcan et al. 2018) used machine learning to learn an optimal mixture of branching rules this mixture is a linear combination of several different branches of these include list them here! Their results showed that the optimal mixture learnt on one set of instances would result in substantially bigger tree sizes than on other instances and hence does not generalise well.

References

Achterberg, Tobias, Thorsten Koch, and Alexander Martin. 2005. "Branching rules revisited." *Operations Research Letters* 33 (1): 42–54. doi:10.1016/j.orl.2004.04.002.

Balcan, Maria-Florina, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. 2018. "Learning to Branch," 1–35. http://arxiv.org/abs/1803.10150.

Clausen, Jens. 1999. "Branch and bound algorithms-principles and examples." *Department of Computer Science, University of . . .*, 1–30. doi:10.1.1.5.7475.

Clements, Ron, and John Musker. 1992. "Aladdin." Buena Vista Pictures.

Conforti, Michele, Gérard Cornuéjols, and Giacomo Zambelli. 2014. *Integer Programming.* Vol. 271. Springer, Cham. doi:10.1007/0-306-48332-7_212.

Du, Ke-Lin, and M. N. S. Swamy. 2016. *Search and Optimization by Metaheuristics.* doi:10.1007/978-3-319-41192-7.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning.* Vol. 27. Springer Series in Statistics 2. New York, NY: Springer New York. doi:10.1007/978-0-387-84858-7.

Kotsiantis, Sotiris B. 2007. "Supervised machine learning: A review of classification techniques." *Informatica* 31: 249–68. doi:10.1115/1.1559160.

Land, A. H., and A. G. Doig. 1960. "An Automatic Method of Solving Discrete Programming Problems." *Econometrica* 28 (3): 497–520.

Shannon, Claude E. 1948. "A mathematical theory of communication." *The Bell System*

Technical Journal 27 (July 1928): 379–423. doi:10.1145/584091.584093.

Wallace, C S. 2005. *Statistical and Inductive Inference by Minimum Message Length*. doi:10.1007/0-387-27656-4.

Yitzhaki, Shlomo, and Edna Schechtman. 2013. *The Gini Methodology*. 1st ed. Vol. 272. Springer Series in Statistics. New York, NY: Springer New York. doi:10.1007/978-1-4614-4720-7.