Monash University

Literature Review

Implicit enumeration with dual bounds from approximation algorithms

Nelson Frew

supervised by

Dr. Pierre Le Bodic

June 4, 2018

Abstract

Implicit enumeration refers to a divide-and-conquer problem solving strategy where we can enumerate through a problem's solutions without specifically finding each solution's value. This has broad uses within many fields of Computer Science, one such example being in Mixed-Integer Programming's (MIP) Branch-and-Bound (B&B). To omit evaluation of solutions from our search, we analyse them with regards to bounds established on where the optimal value can reside, known as dual bounds. Traditionally, bounds are found by relaxing the problem formulation so that it can be solved efficiently, however such approaches may perform arbitrarily poorly according to problem type. In this literature review, we describe the research context with particular reference to MIP and B&B, linking with relevant research in the area that leads to our new approach of deriving dual bounds: approximation algorithms.

Contents

Introduction

The field of Discrete Optimisation studies methods to solve optimisation problems where variables may be constrained to take discrete values (for example, integers) to be valid for a solution. The field is closely related with both Combinatorial optimisation and Integer Programming, as it is often the case that if one can be formulated as a discrete optimisation problem, it can also be formulated as an integer or combinatorial optimisation problem. Generally, the difference in the nomenclature is due to an emphasis on particular problem origins, but the essence remains the same. Consequently, research in the field addresses a wide variety of problems types. Classic problems in Discrete Optimisation include scheduling, shortest path problems, and the cutting stock problem.

One widely used method of solving such problems is by using Mixed-Integer Programming (MIP). MIP refers to a set of modelling and solving techniques for problems where at least one of the variable must be integer, by formulating problems as a *Mixed-Integer Program*. In such a case, we formulate it as follows: we want to maximise/minimise $z = cx$, where c contains problem data, and vector x contains variables to be optimised. Constraints on optimising the objective function are generally linear normally denoted by an inequality on a matrix A and vector b, and the variables can be specified to be integer or continuous. For further reference on the formulation of a Mixed-Integer Program, we refer the reader to [12].

Since we can formulate Discrete Optimisation problems as Mixed-Integer Programs we examine the state of the art method for solving such formulations: the Branch-and-Bound (B&B) approach. Introduced by Land and Doig (1960)[1] with further work done by Dakin (1965)[2], B&B optimises the objective by a divide-and-conquer approach on the searchspace, systematically locating a solution in a process known as implicit enumeration. To implicitly enumerate something, we make inferences about where a valid solution can reside from information garnered through the search. To demonstrate how this might be done, we describe the B&B approach.

Branch-and-bound

Overview

B&B finds an optimal solution by establishing *primal* and *dual* bounds on the optimal value. If we are maximising a value, the dual bounds are an upper bound on the optimal and the primal bounds are the lower bounds. To establish these, we relax the problem by removing a 'hard' constraint, and solve this version. We then constrain a variable to a feasible value in the problem (known as branching) and solve the relaxations of these to find the dual bounds, we have bounds on where optimal can be within these constraints. If, for maximisation, the dual bounds are less than the original problem's primal bounds, we know that constraining variables in this way cannot yield an optimal result, so we remove it from our search (known as bounding). If the relaxation gives a feasible solution to the original problem, and this is higher than our best primal bound, we know that the optimal value is at least this much, and use it as our new primal bound. If the dual bounds from this constrained solution are not associated to a feasible solution, but are still above the highest known primal bounds, we branch further by constraining another variable to feasible values. We continue this process of choosing a partially enumerated solution and solving relaxations until we implicitly enumerated through the solution space.

Questions that arise from this description may include: how do we choose a variable to constrain? How do we relax our solution and obtain dual bounds? How do we decide which partially enumerated solutions to solve next? These questions are in fact non-trivial, and comprise a large component of the research in B&B. We briefly discuss research conducted which aims to improves the performance of a B&B in terms of its *branching*, *bounding*, and *searching* strategies.

Branching strategies

As search continues down a search tree, the level of partial enumeration increases on a solution by constraining another variable at each level of depth. The issue that immediately arises is deciding what the best method for this is. A basic strategy may come to mind is to simply branch on the variable with the largest fractional component, however Achterberg et al. (2005) [14] has showed this to be as bad as random selection. Tomlin (1970) [16] introduced the concept of *use penalties* to inform the branching process, where "up penalties" and "down penalties" were calculated for each variable based on their respective fractional components, and the variable with the largest penalty in either direction was chosen. Mitra (1973)[13] conducted computational experiments which demonstrated such approaches may have limited use, but acknowledging that the similar strategies proposed by Benichou (1971)[9] involving "pseudocosts" on variables may advantages. Morrison et al. (2016) [4], in a survey of branching strategies, proposed a distinction between branching strategies based on the constraints on decision variables: *binary* and non-binary, or *wide*, strategies. Between each category of branching, there are benefits and drawbacks related to each with regards to tree depth and problem type. There has been concentrated research on branching strategies for many years, and is still an active area of work. A detailed survey of the literature and the current questions in research are provided in [4] where the reader is directed for further reading.

Methods of bounding an optimal solution

Another highly important component of B&B's success is the method of bounding partially enumerated solutions. In general, this is done by *relaxing* the problem in some way, then solving the resultant problem. A relaxation of a problem is the removal of any constraints in its formulation in order to manifest a computationally simpler, but nevertheless closely related subproblem. By removing constraints from the original problem, the relaxed feasible region will naturally subsume the region of the original problem. We want to optimise the objective in this relaxation in order to bound the optimal solution, so finding relaxations that minimise this region's growth is key.

Within the context of MIP, there have been various methods to achieve this: with Linear Programming (LP) relaxations, Semidefinite Programming (SDP) relaxations (see Vandeberghe & Boyd (1996) [**?**], Alizadeh (1993), and Lovasz & Shrijver (1991) [8]), and Lagrangian relaxations [6]. LP is similar to IP in that they are both solving optimisation problems that are constrained by linear expressions, with the difference them being IP's intregrality constraint. Solving Linear Programs has been done with efficiency ever since the introduction of the Simplex by Dantzi (1947) [3]. Due to their similarity, LP is a often used to obtain a relaxation of a given IP, by removing the intrality constraint. Despite being widely used, however, LP relaxations are hardly universally applicable: for example, linear relaxations for Vertex Cover generally provide very little information with regards to the optimal solution.

Extensions, improvements, and related techniques

The state of the art methods for solving Mixed-Integer Programs does not normally rely on the B&B alone. One common extension to the method include the introduction of cutting-planes [7] to produce the Branch-and-Cut algorithm. In the cutting plane approach, introduced by Dantzig et al. (1954) [5] and generalised by Gomory (1958) one solves a linear relaxation of a program and formulates a constraint (a cutting plane) that separates this solution from the search space. A Branch-and-Cut [10] [11] is simply a branch and bound approach that can decide to add these cutting planes instead of branching on a solution. For a more detailed treatment, the reader is referred to [12]. Other key methods for extending the B&B are presolving techniques (REFERENCE?) and primal heuristics(REFERENCE).

Another related concept is the idea of *warm starting a solution* in linear programming. In short, warm starting is a method to exploit information gained about the problem from previous computations to inform future ones. Ralphs and Guzelsoy (2006) [15] linked this concept specifically in regards to Integer Programming.

Approximations

Overview

Related to the study of computationally complex problems, the field of Approximation Algorithms (AAs) arose to provide polynomial-time approximations where no efficient optimal solution algorithm was known. Specifically, AA research involves finding algorithms with solution values provably within a factor of the optimal value. We refer to an algorithm as an α-approximation algorithm if regardless of instance it will return a value within a constant factor *alpha* of optimal. For a detailed treatment, we refer the reader to the book by Vazirani (2001) [17].

Case study: Finding an approximation scheme for Knapsack

To illustrate the nature of AA research we demonstrate with a description of an approximation to the classic problem in computer science, the Knapsack Problem (KP). KP is defined as follows: given n items with associated weights w_i and values v_i, and Knapsack capicity W, our program is

maximise $\sum_{i=1}^{n} v_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i \leq W$ and $x_i \in \{0, 1\}$

To solve this problem, we refer to the Dynamic Programming algorithm (DP) as described by [17]: let P be the value of the most valuable object in the item set, and let $A(i, p)$ be the minimal weight of the solution to KP with only the first i items available, where the total value is exactly p. If no such set can exist (that is, the first i items cannot yield a value of p and so cannot have an associated minimal weight), $A(i, p) = \infty$. The DP recurrence can be defined as follows:

$$A(i+1, p = \begin{cases} \min\{A(i,p), w_{i+1} + A(i, p - v_{i+1})\}, \text{ if } v_{i+1} < p \\ A(i+1, p) = A(i, p) \text{ otherwise} \end{cases} \tag{1}$$

The solution is given, then by finding $max\{p|A(n, p) \leq W\}$. This provides exact computation in $O(n^2 P)$, as shown by [17].

From this DP, we now derive a *Fully Polynomial Time Approximation Scheme* (FPTAS) for KP, by introducing an error parameter ϵ. We call an algorithm an FPTAS if its runtime is bounded by a polynomial both in its instance size and $1/\epsilon$. In the case of KP, we construct an FPTAS by ignoring some amount of the least significant bits of the profit values, parameterised by ϵ.

Algorithm 1: FPTAS for Knapsack

Data: A KP problem instance, and error parameter ϵ.
Result: The Weight and value pair of the optimal solution

1 Given ϵ, let $K = \frac{\epsilon P}{n}$;
2 For each object i, define adjusted values $v' = \left\lfloor \frac{v_i}{K} \right\rfloor$;
3 Run a DP with these adjusted item values;
4 Return the approximation solution set, S', provided by the DP;

The guarantee associate with this FPTAS is that the values returned by this algorithm are guaranteed to be, for optimal value OPT, at least $(1 - \epsilon) \cdot OPT$. The proof for this result is as follows: let $z_{S'}$ be the value of the solution set S' obtained from the FPTAS, and z_O be the value of the optimal solution set O. Also, let $z'_{S'}$ be the value of the solution set with adjusted/truncated values value the FPTAS's solution set S', and z'_O be the optimal solution set with similarly adjusted value for all items.

Because of the flooring operation truncating off at most K we know that the difference between z_O and $K \cdot z'_O$ would be at most nK. Further, since $z'_{S'}$ is the optimal value for the adjusted profits, z'_O cannot exceed it. So,

$$z_O - nK \leq K \cdot z'_O \leq K \cdot z'_{S'} \leq z_{S'} \leq z_O$$

Since $K = \frac{\epsilon P}{K}$,

$$z_O - nK = z_O - \epsilon P$$

Finally, since $P \leq z_O$,

$$(1 - \epsilon) \cdot z_O \leq z_{S'} \leq z_O$$

For further details and the proof associated with the time complexity of this scheme, refer to [17].

Problem structure and the general LP formulation

An important observation of the FPTAS for KP is that it is achieved specifically by exploiting the problem structure. It is for this reason that LP programming can perform arbitrarily badly: it is inherently a general formulation. Further, there are algorithms where solutions are just easier to obtain combinatorially than with LP formulations. For example, for finding the maximum matching of a graph, the combinatorially oriented Blossom Algorithm performs better than formulating an LP with blossom inequalities (CITE). Simply, the main drawback to LP is that there is no provable guarantee on solution quality with LP techniques.

Relating approximations to LP values

As far as we know, the sole dedicated treatment that has been given to both AAs and LPs within the scope of implicit enumeration was provided by Wolsey (1980) [**?**]. Wolsey provided a general analysis technique for AAs to relate them to LP relaxation solutions, with the objective of reflecting the relation between primal and dual bounds in a B&B. For an approximate solution value Z^H, and LP relaxation solution value Z^{LP}, he derived inequalities of the form $Z^H \leq rZ^{LP} + s$ for $r \geq 1$. This analysis was extended to formulations on the Bin Packing Problem, Longest Undirected Hamiltonian Tours, Minimum Length Eularian Tours and the Chinese Postman Problem. From these components, Wolsey showed that it was possible to achieve implicit enumeration in the form of a Branch-and-Bound algorithm, with the observation that AAs are usually computed with partial enumeration. Specifically, he showed that AAs, when used in this way, would monotonically approach the value of the relaxation as the level of enumeration increased. The product is the only implicit enumeration scheme leveraging AA guarantees to find an optimal value that we are aware of.

Implicitly enumerating with approximations

With the work from Wolsey, we can see that AAs can be used to achieve implicit enumeration with LP relaxations operating as both the dual bounds and as a point of convergence for the solution values of the AAs. However, we can omit the use of LP entirely by observing the following properties following from the analysis on KP where we saw:

$$(1 - \epsilon) \cdot z_O \leq z_{S'} \leq z_O$$

Since

$$(1-\epsilon)z_O \leq z_O - nK \leq K \cdot z'_{S'}$$

We find upper and lower bounds on the optimal value:

$$z_O \in [z_{S'}, K \cdot z'_{S'} + nK]$$

Conclusion

In this document we have overviewed the relevant components to understanding the relevant questions posed by using AAs to find dual bounds. We have seen that using such mechanisms allow us to get Dual Bounds with guarantee, and the steps that follow relate to seeing how we can use these components in light of previous research in MIP. There are several questions that remain: what searching and branching strategies does such an approach benefit from? How can we use the concept of Warm Starting in tandem with such components as the DP's that are computed to get dual bounds? How does such an approach compare to current state of the art methods?

Importantly, we hope to build on the work done by Wolsey and help to attract attention to find more applications for AAs in the future.

References

[1] A.G. Doig A. H Land. An automatic method of solving discrete programming problems. 1960.

[2] R. J. Dakin. A tree search algorithm for mixed integer programming problems. 1965.

[3] George Dantzig. Maximization of a linear function of variables subject to linear inequalities: in t.c koopmans (ed.). 1947.

[4] Jason J. Sauppe Edward C. Sewell David R. Morrison, Sheldon H. Jacobson. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. 2016.

[5] S. Johnson. G. Dantzig, R. Fulkerson. Solution of a larg-scale traveling-salesman problem. 1954.

[6] A.M. Geoffrion. Lagrangean relaxation for integer programming. 1974.

[7] R.E Gomory. Outline of an algorithm for integer solutions to linear programs. 1958.

[8] A. Schrijver L. Lovasz. Cones of matrices and set-function and 0-1 optimisation. 1991.

[9] P. Girodet G. Hentges G. Ribiere O. Vincent M. Benichou, J.M. Gauthier. Experiments in mixed-integer linear programming. 1971.

[10] G. Rinaldi M. W. Padberg. Optimization of a 532-city symmetric travelling salesman problem by branch and cut. 1987.

[11] G. Rinaldi M. W. Padberg. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. 1991.

[12] Giacomo Zambelli Michele Conforti, Gerard Cornuejols. *Integer Programming*. Springer International Publishing Switzerland, 2014.

[13] G. Mitra. Investigation of some brancha nd bound strategies for the solution of mixed integer linear programs. 1973.

[14] A. Martin T. Achterberg, T. Koch. Branching rules revisited. 2005.

[15] Menal Guzelsoy Ted Ralphs. Duality and warm starting in integer programming. 2006.

[16] J. A. Tomlin. Branch and bound methods for integer and non-convex programming. 1970.

[17] Vijay Vazirani. *Approximation Algorithms.* Springer-Verlag Berlin, Heidelberg, 2001.