(DRAFT) Implicit enumeration with dual bounds obtained from approximation algorithms

Nelson Frew

1 Introduction

The field of discrete optimisation (CITE) studies problems that model discrete decision making. Mixed-Integer Programming refers to a set of modelling and solving techniques for such discrete problems (CITE) by formulating them as Mixed-Integer Programs (MIPs). For many problems, this approach is the most efficient known method to find a solution. The standard MIP formulation, for maximisation, is as follows:

$$\begin{array}{ll} \text{maximise} & cx + hy \\ \text{subject to} & Ax + Gy \leq b \\ & x \geq 0 \text{ integral} \\ & y \geq 0 \end{array}$$

Where x and y are the sets of decision variables that we assign values to with the purpose of maximising the objective function $cx + hy$. Generally, objective functions are constrained by linear inequalities, and variables can be specified to be integer or continuous.

Optimisation problems with linear constraints and continuous variables are known as Linear Programs (LPs), which are both solvable in polynomial time and very efficiently in practice (CITE). In this report, we consider maximisation problems whose constraints are all linear. MIPs, on the other hand, are characterised by having one or more variables constrained to be integer, generally resulting in an NP-Hard problem (CITE). By removing the integer constraints on variables in a MIP, we obtain its associated *LP relaxation*.

Because the feasible region of continuous solutions naturally subsumes the set of feasible solutions for integers, we know, for optimal value for the LP relaxation z_{LP} and optimal value to the MIP z, that

$$z \leq z_{LP}$$

Since z_{LP} is at least as large as our optimal value, z is provided with an upper bound, which we refer to as the *dual bound*. The state-of-the-art method for solving MIPs uses this relationship to search for what are known as *primal solutions* that are feasible to the MIP, and employs dual bounds to implicitly enumerate through these. We find primal solutions by forming *subproblems*, constrained versions of the original problem, and solving the LP relaxation of these. If the solved LP relaxation is feasible to the original MIP, then we have found a primal solution, and so have an associated lower bound on our optimal value, which we refer to as the *primal bound*.

The process of choosing a variable to constrain is known as *branching*, and is generally informed by information obtained from the LP relaxation's solution. Branching can form a number of subproblems, which are all considered *branches* of the parent problem. If we find that a given subproblem's dual bound is lower than the MIP's primal bound, then we can safely *prune* this branch as we know further evaluation and constraining of this subproblem cannot yield a better primal solution. Otherwise, if this subproblem has a dual bound above our primal bound, we continue constraining and solving the relaxation until a new feasible solution is found, or a further constrained subproblem of this is pruned.

This constitutes the core of the Branch and Bound approach, which we formally describe below: let (x^i, y^i) be x and y values associated with the optimal solution to linear program LP_i, (x^*, y^*) denote an optimal solution to the MIP; $\underline{z}$ denote the lower bound on the optimal value and z^* the optimal solution of the MIP,

and let $\mathcal{L}$ denote the list of nodes of the Branch and Bound tree yet to be solved (i.e. not pruned nor branched on).

Algorithm 1: Branch and Bound algorithm for Mixed-Integer Programming

Data: A Mixed-Integer Program.
Result: The optimal solution value, and the associated solution

Set $\mathcal{L} := N_0$, set $\underline{z} := -\infty$, set $(x^*, y^*) := \emptyset$;
while $\mathcal{L}$ *is not empty* **do**
  Select a node N_i from $\mathcal{L}$, by some node selection scheme, deleting it from $\mathcal{L}$;
  Solve LP_i to get solution value z_i and (x^i, y^i) if it exists;
  if LP_i *is infeasible, i.e.* $z_i = -\infty$ **then**
    Go to step 2;
  end
  else
    Let (x^i, y^i) be an optimal solution of LP_i and z_i be its objective value
  end
  if $z_i \leq \underline{z}$ **then**
    Go to step 2;
  else
    if (x^i, y^i) *is feasible to the MIP* **then**
      Set $\underline{z} := z_i$;
      Set $(x^*, y^*) := (x^i, y^i)$;
      Go to step 2;
    end
  end
  else
    Branching: From LP_i construct 2 linear programs LP_{i1}, LP_{ik} with smaller feasible regions whose union does not contain (x^i, y^i), but contains all the solutions of LP_i with $x \in \mathbb{Z}$, by choosing a variable to constrain.;
    Add the corresponding new nodes N_{i1}, N_{ik} to $\mathcal{L}$ and go to step 2.
  end
  Return $\underline{z}$;
end

While finding optimal solutions with Branch and Bound is in worst case exponential-time, for some problems there exist Approximation Algorithms (AAs) which can provide feasible solutions in as fast as polynomial-time. To achieve this, AAs exchange a guarantee on optimality for a guarantee the *quality* of the solution, within some factor of the optimal value.

An α-approximation is an AA that guarantees that the solution value will always be within a constant factor $\alpha < 1$ of the optimal solution; i.e. for an approximate solution value z_A, it must be that $\alpha z \leq z_A \leq z$. As a result, for a maximisation problem, we can analytically derive an upper bound on the optimal value from the lower bound guarantee on the optimal solution by simple algebraic manipulation. In this way, we can use AAs in lieu of an LP relaxation to obtain valid dual bounds on our optimal solution. Because we are leveraging guarantees on the optimal value, the dual bounds we obtain are also guaranteed to be within a level of accuracy. This is distinct from LP relaxations, which can provide arbitrarily poor dual bounds.

When conducting a Branch and Bound with AAs, an important observation is that the general strategies which have contributed to the success of the standard Branch and Bound with LPs must be addressed. In particular, methods of branching and node selection do not have a clear translation with respect to AAs. In this project, we investigate methods of effectively using dual bounds provided by AAs for the 0,1 Knapsack. To do this, we investigate known approximation schemes and methods to construct high quality dual bounds from them, as well as devising branching strategies within its Branch and Bound.

2 Literature Review

We now give a brief literature review of the research and improvements to the Branch and Bound approach, as well as an overview of details and approaches with AAs that are central to the premise of this project.

2.1 Branch and Bound improvements and strategies

While our previous description of the Branch and Bound approach captured the core components, there still remain many aspects which are uncertain: which variables to constrain for branching, alternative methods to relax a solution, and node selection schemes. These problems are in fact non-trivial and represent active research areas within Mixed-Integer Programming. In light of this, we provide an overview of known strategies and the body of research that comprises the current understanding in the field.

2.1.1 Branching Strategies

Deriving useful subproblems is central to the Branch and Bound approach, as it provides us with the partial enumeration required to establish bounds on an optimal solution value. A basic strategy is to branch on the variable with the largest fractional component, however [1] has shown this to be as bad as random selection. It is possible to make inferences based on the properties of each variable's fractional component, as shown by [23] by the proposal of "use penalties". This was used to inform the branching process where "up penalties" and "down penalties" were calculated for each variable based on their respective fractional components, after which the variable with the highest penalty in any direction was chosen. However, [17] showed with computational experiments that such approaches may have limited use, while approaches involving issuing "pseudocosts" as proposed by [2] may have advantages.

In an attempt to provide a categorisation for branching methods, [18] proposed two groupings: *binary/non-binary* strategies, and *wide* strategies. Both problem types and tree depth provided the ideas central to these classifications.

A detailed survey of the literature and the current open questions in research is also provided by [18], where the reader is directed for further reading.

2.1.2 Node selection schemes

Talk about best bound?

2.1.3 Bounding optimal solutions

As we have seen, a natural method of finding dual bound on optimal solutions can be done with LP relaxations. Solving LPs has been efficiently possible since the introduction of Dantzig's Simplex method [6]. Despite its efficiency, though, LP relaxations can provide arbtirarily poor bounds in this context: such an approach for finding the Vertex Cover of a graph will be generally far from optimal (CITE?). However, this is only one of many known ways to obtain valid dual bounds within Mixed Integer Programming: other examples include Semidefinite Programming relaxations (see [15, 24]), and Langrangian relaxations [8].

2.1.4 Extensions, improvements, and related techniques

MIP solvers in the state-of-the-art often combine Branch and Bound strategies with various extensions to improve performance. One common extension to the method includes using cutting planes [9] to produce the Branch-and-Cut algorithm. The cutting plane approach, introduced by [5], and extended to general Integer Programs (IPs) in [9], involves solving a LP relaxation of a program and formulates a constraint, a

cutting plane, that separates this solution from the rest of the search space. A Branch and Cut approach [20, 21], then, is a Branch and Bound with the ability to choose to add a cutting plane instead of branching on variable. For a more detailed treatment on this topic, the reader is referred to [4]. Other key methods for extending the Branch and Bound are with presolving techniques [16] and using primal heuristics [3].

Another related concept is the idea of warm starting a solution in linear programming linked to integer programming by [22]. In short, warm starting is a method to exploit information gained about the problem from previous computations to inform future ones. Ones simple method for this is to use previous computation in finding a starting primal bound in a Branch and Bound.

2.2 Approximation algorithms for the Knapsack Problem

As mentioned, the case study chosen for experimentation within this project is the 0,1 Knapsack Problem (KP). Known to be NP-Hard (CITE), KP has been studied extensively both in to solving optimally and approximately (see, for example [13, 10, 12, 14, 7, 19]), and as such, it serves well as a topic for analysis. We describe the formulation for KP as follows: given n items associated weights w_i and values v_i, and Knapsack capacity W, our program is

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq W, \text{ and } x_i \in \{0,1\} \end{aligned}$$

It is well known that KP lends itself well to a Dynamic Programming (DP) approach, although there are many ways to conduct this. One of the most well known examples, performing a DP by profits as presented by [25], can find a solution in $O(n^2P)$ time for the most profitable item's value P, which we now describe: 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 exists[1], then $A(i,p) = \infty$.

The DP recurrence can then 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 optimal solution value can then be found be finding the value $\max\{p \mid A(n.p) \leq W\}$.

Being NP-Hard, KP does not admit a polynomial time algorithm, however we can construct a *polynomial time approximation scheme* (PTAS). A PTAS is guaranteed to give a solution value within a factor of $(1-\epsilon)$ of optimality, and have running time bounded by a polynomial in n, for fixed values of ϵ. A classic example of a PTAS was presented by (cite Sahni), with time complexity $O(n^{1/\epsilon})$.

If we restrict the definition of a PTAS further, so that the running time is bounded by a polynomial in both n and ϵ, we obtain a *fully polynomial time approximation scheme* (FPTAS). This means we can choose an ϵ that gives us a high quality solution while still being polynomially bounded.

It is common for FPTAS's for KP to to achieve polynomially bounded run time by adjusting the input problem to reduce its complexity, and using a DP to then solve this adjusted problem. As such, the construction of an FPTAS normally involves two distinct steps: a preprocessing step to form the simpler problem, and a step where this is solved to find an approximate value for the original problem. We now give an overview of the range of FPTAS's described that led to the state of the art.

[1] that is, the first i items cannot yield a value of p and so cannot have an associated minimal weight

2.2.1 Ibarra and Kim's FPTAS (1975)

Ibarra and Kim's FPTAS: with DP by profits

What is now a standard method of constructing an FPTAS for KP was first described by Ibarra and Kim [11]. While there are several components at play in their proposal algorithm, the core of what creates the approximation involves a scaling the values of all items down to by a factor dependant on ϵ. This distillation of their algorithm, as described by [25], is presented below:

Algorithm 2: 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 $v' = \lfloor \frac{v_i}{K} \rfloor$;
3 Run a DP with these adjusted item values;
4 Return the approximation solution set, S', provided by the DP;

By using the DP recurrence (1) described in the previous section, we can obtain an FPTAS for the Knapsack. We now present the argument for this as described by [25].

Lemma 1. *The set, S', output by the algorithm satisfies:*

$$value(S') \geq (1 - \epsilon) \cdot OPT$$

Proof. Here is my proof □

Theorem 1. *Algorithm 2 is an FPTAS for KP*

Proof. Assume that it is the case. Therefore by Proof By Assumption we have proven it to be true. □

Use Rhee's thesis to help me do this

By refining our approach, we can improve our runtime further.

- Preprocessing: Talk about dynamic programming on expensive items
- Computing: Talk about greedy approach on cheap items
- Results in $O(mP')$ time algorithm

The approach described by [11] builds on this core approach by improving the nuance of the preprocessing step, which we know describe. The first improvement results from acknowledging that while a given input problem will potentially have many items with very small value, the bulk of the value of an optimal solution will likely be from high value items THIS DOESN'T SEEM LIKE THE ACTUAL REASON.

2.2.2 Lawler (1978)

What he changed about the DP.

- Solution pairs as presented by WS
- Amended rounding procedure
- Complexity

2.2.3 Magazine and Oguz (1981)

More changes to the last two

- Based on the DP recurrence (1)
- Builds on both Lawler and Ibarra & Kim
- Space and time complexities

2.2.4 Kellerer and Pferschy (1999)

Use both Rhee and their informal description to fuel the discussion of this part

2.2.5 Kellerer and Pferschy (2001)

Vector merging procedure

3 Design and Analysis

of the project that is

Stuff to include:

- A priori
- A posteriori 1
- A posteriori 2

4 Implementation and results

Benchmarks lol

4.1 What is a logical way to lay all this out?

5 Todos

(**Put warm starting in future work**)

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[3] Timo Berthold. Primal heuristics for mixed integer programs. 2006.

[4] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Integer programming, volume 271 of graduate texts in mathematics, 2014.

[5] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[6] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *New York*, 1951.

[7] RS Garfinkel and GL Nemhauser. Integer programming, 1972.

[8] Arthur M Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to integer programming*, pages 82–114. Springer, 1974.

[9] Ralph E Gomory et al. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958.

[10] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[11] Oscar H Ibarra and Chul E Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, 1975.

[12] Giorgio P Ingargiola and James F Korsh. Reduction algorithm for zero-one single knapsack problems. *Management science*, 20(4-part-i):460–463, 1973.

[13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack problems. 2005.

[14] Peter J Kolesar. A branch and bound algorithm for the knapsack problem. *Management science*, 13(9):723–735, 1967.

[15] László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0–1 optimization. *SIAM journal on optimization*, 1(2):166–190, 1991.

[16] Ashutosh Mahajan. Presolving mixed-integer linear programs. *Preprint ANL/MCS-P1752-0510, Mathematics and Computer Science Division*, 2010.

[17] Gautam Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4(1):155–170, 1973.

[18] David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[19] George L Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[20] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[21] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[22] TK Ralphs and M Güzelsoy. Duality and warm starting in integer programming. In *The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference*. Citeseer, 2006.

[23] JOHN A Tomlin. Branch and bound methods for integer and non-convex programming. *Integer and nonlinear programming*, 1:437–450, 1970.

[24] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

[25] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.