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

Nelson Frew

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. This has broad uses within many fields of Computer Science, one such example being in Mixed-Integer Programming's Branch and Bound. 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 report, we describe our investigation into a new approach for deriving dual bounds: approximation algorithms.

1 Introduction

1.1 Background

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.

1.2 Research Context

Following the inception of Dantzig's [**?**] Simplex method in 1947, theoretical interest in finding optimal solutions shifted to harder problems. Solving discrete optimisation problems, known as Integer Programs (IPs), began to thrive as a focal point of research. Discrete optimisation gave rise to two new active research areas: exactly solving IPs, and active pursuit of polynomial time algorithms for such hard problems and their variants.

Research into optimally solving IPs led to Land and Doig's initial Branch and Bound method [**?**] in 1960, but had little direct applicability due to limitations in computing hardware at the time. On the other hand, as a response to the many discrete optimisation problems for which polynomial-time algorithms were not known, Computational Complexity Theory research began[**?**]. Concerns with finding polynomially bounded algorithm run-times led to sacrificing optimality for efficiency, in what became AA research through the 1970s. Approximations for the Knapsack Problem[**?**], Travelling Salesman Problem [**?**], Facility Location problem [**?**], and many others were devised through the decade. Until 1980, discrete problem algorithm design and AA design remained disconnected.

It wasn't until Wolsey [**?**] attempting to unify AAs of the 1970s with the Branch-and-Bound algorithm of 1960 that the fields met again. Wolsey provided a general analysis technique relating approximation worst-cases with optimal LP relaxation solutions, and devised a Branch-and-Bound procedure from these components. However, since this point, to the best of our knowledge, no further work has been done in investigating the relationship between approximations and Branch and Bound. This project investigates whether a stronger link exists and can be leveraged to improve performances.

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 high quality bounds on an optimal solution's value. When using LP relaxations, a generally effective strategy is to branch on the variable with the largest fractional component in the LP's solution, and while [1] has shown that this can be as bad as random selection, it is still regarded as the most reliable branching strategy (CITE?).

It is possible to make inferences based on the properties of each variable's fractional component, as shown by [28] 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, [20] 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, [21] described 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 [21], where the reader is directed for further reading.

2.1.2 Node selection schemes

Having chosen a variable to branch on, we will have generated a number of subproblems which are represented as *nodes* at the frontier of our search tree. Because of this representation, the method of node selection is also known as the *search strategy* for the tree.

Consider diagram here

As in our description of the Branch and Bound, we will refer to the set of nodes which are to be explored as $\mathcal{L}$. To continue our search, it is clear that we must choose a node to explore, however the choice of method for doing this is not necessarily straightforward. There are many methods for conducting such a tree search, the most common of which we now describe.

The *depth-first search* (DFS) strategy is a standard and well studied approach to both Branch and Bound and general tree traversal [10, 27]. In this strategy, one arranges the unexplored nodes in a first-in first-out (FIFO) manner, by storing $\mathcal{L}$ as a stack. While this approach may seem naive, it can be modified to have very low memory costs. There are many potential drawbacks to DFS, as well as many derivations to counteract these, and the reader is directed to [21] for a more comprehensive overview of these concepts and techniques.

The *breadth-first search* (BrFS) strategy is likewise a notorious and well studied search strategy trees. The strategy employed by BrFS is often considered to be opposite to DFS, as it employs a last-in first-out (LIFO) search, often by storing $\mathcal{L}$ in a queue. Despite being a well known strategy, BrFS is not often used within Branch and Bound searches.

The final main category of search we will consider is known as the Best-First Search (BFS). As its name indicates, instead of having the order which nodes are created govern the order which they are chosen, we define a way to appraise the "quality" of nodes to inform our choice. One can conduct such a search by storing $\mathcal{L}$ as a priority queue based on the "quality" of a node. The definition of "quality" is clearly important, but within a Branch and Bound a basic but useful indicator would any information about a node's dual bound. **HELP** BFS is known to be highly effective; [7] showed that BFS could provably explore the lowest number of subproblems in certain families of scenarios, although it is also relevant to note that [26] demonstrated potential flaws in the scheme.

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 [18, 29]), and Langrangian relaxations [9].

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 [11] to produce the Branch-and-Cut algorithm. The cutting plane approach, introduced by [5], and extended to general Integer Programs (IPs) in [11], 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 [23, 24], 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 [19] and using primal heuristics [3].

Another related concept is the idea of warm starting a solution in linear programming linked to integer programming by [25]. 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 [15, 12, 14, 16, 8, 22]), 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

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

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 values as presented by [30], 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:

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

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

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 $O(n^{1/\epsilon})$.

While PTAS's are indeed bounded by a polynomial in n, they can be exponential in terms of ϵ, so long as ϵ is fixed. If we restrict the definition of a PTAS further, then, 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. For our ends, we consider the examples within the literature which have led to what are now textbook examples of FPTAS's to show the performance of well established approximations when used in a Branch and Bound scheme.

2.2.1 Ibarra and Kim's FPTAS (1975)

What is now a standard method of constructing an FPTAS for KP was first described by Ibarra and Kim [13]. 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. The distillation of their algorithm, as described by [30], 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 $v' = \left\lfloor \frac{v_i}{K} \right\rfloor$;
3 Run a DP with these adjusted item values;
4 Return the approximation solution set, S', provided by the DP;

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 [30].

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

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

Proof. 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. Furthermore, 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 $P \leq z_O$,

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

□

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. □

This constitutes the core of what is one of two textbook approaches to constructing an FPTAS for KP. There are nuanced optimisations which supplement this original algorithm's approach given by [13], however for simplicity we omit such details here.

2.2.2 Lawler (1978)

The second standard construction for obtaining an FPTAS for KP was presented originally by [17], which we present as described by [31]. Building on the contributions of [13], this approach describes an alternative FPTAS which still maintains the basic approach shown in Algorithm 2. Specifically, we construct this algorithm by changing our DP approach.

For $j \leq n$, let A_j be an array which contains representations of solutions to KP for the first j items. A solution is represented by an ordered pair (v, w) in A_j if there is a set of the first j items with value v and weight $w \leq W$. Instead of explicitly storing all partial solutions, we resolve to only store partial solutions which are not *dominated* by other partial solutions. We say a solution (v, w) dominates solution (v', w') if $v \geq v'$ and $w \leq w'$.

The DP algorithm, as presented by [31], then follows:

Algorithm 3: Alternative dynamic programming for Knapsack

Data: A KP problem instance
Result: The value of the optimal solution

```
for j ← 2 to n do
    for each (v, w) ∈ A_j do
        if w + w_j ≤ W then
            Add (v + v_j, w + w_j) to A_j
        end
    end
    Remove dominated pairs from A_j
end
return max_{(v,w)∈A_j} v
```

It can be shown that this algorithm correctly finds the optimal value to KP, and the reader is directed to either [17] or [31] for the proof of this result. By using this DP algorithm in Algorithm 2, we obtain another valid FPTAS for KP.

The original version of this algorithm, presented in [17], contains further subtle refinements to this algorithm by improving details of the FPTAS provided by [13].

2.3 Approximations, Linear Programming and Branch and Bound (find better section title)

All that now remains to review is the existing literature which addresses AAs in an implicit enumeration setting. As far as we are aware, the only focused treatment of this was presented by Wolsey [32].

Acknowledging that LPs are tightly knitted to the state of the art in Branch and Bound, Wolsey demonstrated the use of a framework which allows one to emulate characteristics of linear duality with AAs. In this way, the first and only provided method for finding dual bounds was presented.

For an approximate solution value z_A, an LP relaxation solution value z_{LP}, Wolsey showed we can construct inequalities of the form:

$$z_A \leq r \cdot z_{LP} + s$$

for $r \geq 1$. Further, he showed that we can extend this analysis to formulations on the Bin Packing Problem, Longest Undirected Hamiltonian Tours, Minimum Length Eularian Tours and the Chinese Postman Problem.

Following this analysis, Wolsey demonstrated that we can construct a Branch and Bound where the value of z_A would monotonically increase as enumeration progressed, inevitably convering to optimality. The product is the only implicit enumeration scheme leveraging AA guarantees that we are aware of.

Note to self: I hadn't spoken of linear duality up to this point.. Should I have?

3 Design and Analysis

In light of the approaches described in the literature to both Branch and Bound and AAs, our objective is to now find a method of constructing high-quality dual bounds. Intuitively, the guarantees which help define given FPTAS's for KP are not proven with such applications in mind, so there are potential extensions to the analysis which we can use to find stronger dual bounds. In this section, we demonstrate analysis methods which have been undertaken to this end.

3.1 Analytic derivation of dual bounds a priori

To form a basis to our discussion, we reintroduce the bounds obtained by the FPTAS originally presented by Ibarra and Kim in [13]. In particular, recall that we found

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

for the optimal value:

$$z_O \in [z_{S'}, \frac{z'_{S'}}{1 - \epsilon}]$$

TODO: Something something something truncation branching (does this need its own section?)

This provides us with the following first attempt at an amendeded Branch and Bound algorithm:

Algorithm 4: Branch and Bound with Approximation Algorithms

Data: A MIP instance, and an AA for the given problem.
Result: The Weight and value pair of the optimal solution

Let $\bar{z}$:= approximate solution;
while $\mathcal{L}$ is not empty **do**
    Choose a node N_i from $\mathcal{L}$, and delete it from $\mathcal{L}$;
    Run the AA, deriving upper bound (UB) and lower bound (LB) in the process;
    if UB $\leq \bar{z}$ **then**
        Go to step 3;
    end
    else
        if LB$> \bar{z}$ **then**
            $\bar{z}$:= LB;
        end
    end
    if for N_i's parent's UB, UB_p, $UB > UB_p$ **then**
        Set $UB := UB_p$
    end
    else
        Choose a variable to branch on, then generate and enqueue new child nodes;
    end
    Return $\underline{z}$;
end

Here are some LP benchmarks lol

	Branching Strategy	Running time	Memory used	Node count	Avg. time per node
instance 1	Fractional	10	10	10	10
instance 1	Random	10	10	10	10
instance 2	Fractional	10	10	10	10
instance 2	Random	10	10	10	10

Implementing this approach, we can see how this performs on given instances of KP

	DP	Branching Strategy	Running time	Memory used	Node count	Avg. time per node
instance 1	I&K	Truncation	10	10	10	10
instance 1	I&K	Random	10	10	10	10
instance 2	Lawler	Truncation	10	10	10	10
instance 2	Lawler	Random	10	10	10	10

Likewise we obtain the aggregate stats for these problems of similar form:

	DP	Branching Strategy	Avg. running time	Avg. memory used	Avg, node count	Mean avga time per node
instance 1	I&K	Truncation	10	10	10	10
instance 2	I&K	Random	10	10	10	10
instance 3	Lawler	Truncation	10	10	10	10
instance 4	Lawler	Random	10	10	10	10

3.2 Derivation of dual bounds a posteriori

We can obtain improvements to our dual bounds by observing information obtained following completion of the FPTAS, however, i.e. *a posteriori*. To do this, we first observe that the approximate solution set S' will have a profit of $p(S')$. This value will be the value of the adjusted profits $p'(S')$ scaled up by K, plus any profit ω lost from the flooring operation, i.e.

$$p(S') = Kp'(S') + \omega$$

Todo the same analysis but with omega involves.

This analysis leads to the insight that

$$OPT \leq \frac{(p(S')\omega)}{1-\epsilon}$$

Which gives us

$$OPT \leq p(S') + Kn - \omega$$

Implementing this approach, we can see how this performs on given instances of KP

	DP	Branching Strategy	Running time	Memory used	Node count	Avg. time per node
instance 1	I&K	Truncation	10	10	10	10
instance 1	I&K	Random	10	10	10	10
instance 2	Lawler	Truncation	10	10	10	10
instance 2	Lawler	Random	10	10	10	10

Likewise we obtain the aggregate stats for these problems of similar form:

	DP	Branching Strategy	Avg. running time	Avg. memory used	Avg, node count	Mean avga time per node
instance 1	I&K	Truncation	10	10	10	10
instance 2	I&K	Random	10	10	10	10
instance 3	Lawler	Truncation	10	10	10	10
instance 4	Lawler	Random	10	10	10	10

3.3 Improved a posteriori dual bounds

Another approach is if we round up. We get these bounds, see?

$$OPT \leq p(S'') + \omega$$

Implementing this approach, we can see how this performs on given instances of KP

	DP	Branching Strategy	Running time	Memory used	Node count	Avg. time per node
instance 1	I&K	Truncation	10	10	10	10
instance 1	I&K	Random	10	10	10	10
instance 2	Lawler	Truncation	10	10	10	10
instance 2	Lawler	Random	10	10	10	10

Likewise we obtain the aggregate stats for these problems of similar form:

	DP	Branching Strategy	Avg. running time	Avg. memory used	Avg, node count	Mean avga time per node
instance 1	I&K	Truncation	10	10	10	10
instance 2	I&K	Random	10	10	10	10
instance 3	Lawler	Truncation	10	10	10	10
instance 4	Lawler	Random	10	10	10	10

4 Future work

Things like:

- Warm starting
- State of the art KP FPTAS
- Christofides algorithm for metric TSP

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] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

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

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

[10] Solomon W Golomb and Leonard D Baumert. Backtrack programming. *Journal of the ACM (JACM)*, 12(4):516–524, 1965.

[11] 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.

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

[13] 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.

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

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

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

[17] Eugene L Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

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

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

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] Edward C Sewell and Sheldon H Jacobson. A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing*, 24(3):433–442, 2012.

[27] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

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

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

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

[31] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[32] Laurence A Wolsey. Heuristic analysis, linear programming and branch and bound. In *Combinatorial Optimization II*, pages 121–134. Springer, 1980.