A Column and Constraint Generation Algorithm for the Dynamic Knapsack Problem with Stochastic Item Sizes

Daniel Blado*[1] and Alejandro Toriello†[2]

[1]Global Data Insights and Analytics, Ford Motor Company, Dearborn, Michigan 48126
[2]H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

October 8, 2019

Abstract

We consider a version of the knapsack problem in which an item size is random and revealed only when the decision maker attempts to insert it. After every successful insertion the decision maker can choose the next item dynamically based on the remaining capacity and available items, while an unsuccessful insertion terminates the process. We propose an exact algorithm based on a reformulation of the value function linear program, which dynamically prices variables to refine a value function approximation and generates cutting planes to maintain a dual bound. We provide a detailed analysis of the zero-capacity case, in which the knapsack capacity is zero, and all item sizes have positive probability of equaling zero. We also provide theoretical properties of the general algorithm and an extensive computational study. Our main empirical conclusion is that the algorithm is able to significantly reduce the gap when initial bounds and/or heuristic policies perform poorly.

1 Introduction

The deterministic knapsack problem is a fundamental discrete optimization model that has been studied extensively in mathematical optimization, computer science, operations research, and industrial engineering. Recently, knapsack models under uncertainty have been the subject of much work, both to model resource allocation problems with uncertain parameters, and as subproblems in more general discrete optimization problems under uncertainty, such as stochastic integer programs. Dynamic knapsack problems, where decisions occur in sequence and problem parameters may be revealed dynamically, have particularly been a topic of ongoing interest; such models have seen many applications, such as in scheduling [11], equipment replacement [12], and machine learning [15, 23].

The dynamic knapsack model variant that we analyze here has stochastic item sizes that are only revealed to the decision maker after they attempt to insert an item. After every successful insertion the decision maker can choose the next item dynamically based on the remaining capacity and available items, while an unsuccessful insertion terminates the process. This differs from more static models, in which the decision maker decides on a particular subset of items a priori, essentially

*dblado *at* ford *dot* com
†atoriello *at* isye *dot* gatech *dot* edu

attempting to insert the set simultaneously, and the question is whether that set fits the knapsack with a specified probability, e.g. [14].

Having the flexibility to choose subsequent items in light of the revealed sizes of previous items can intuitively lead to a higher overall expected value than a more static approach. However, this added freedom raises the complexity of the problem both practically and theoretically. A feasible solution to this problem is a policy that dictates which item the decision maker attempts to insert under any possible state, as opposed to simply a subset of items to insert a priori. Such added difficulty motivates the development of reasonably tight, tractable relaxations and high-quality, efficient policies; see for instance [3, 5, 6, 11]. However, there is no known method to systematically improve such relaxations to eventually arrive at an optimal solution, even under the assumption of integer size support; our goal in this paper is thus to provide a computationally efficient, exact algorithm with such convergence guarantees. Our contributions can be summarized as:

i) We analyze the special case where the capacity is zero: We prove that a straightforward optimal policy exists, and that the variables corresponding to the optimal value function have closed-form solutions. Such results provide both insights and a valid starting bound applicable to the general capacity case.

ii) We develop a finitely terminating general algorithm that solves the dynamic knapsack problem under integer sizes and capacity within an arbitrary numerical tolerance, using a dynamic value function approximation approach. We present both theoretical analysis and an extensive computational study, and show that the algorithm is able to significantly reduce the gap precisely when the initial bounds or heuristic policies perform poorly.

The remainder of the paper is organized as follows. We conclude this section with a brief literature review. Section 2 states the problem formulation and preliminaries, including relevant previous results. Section 3 provides the complete analysis of the zero capacity case. Section 4 then states the general algorithm and examines structural results, while Section 5 summarizes our computational study. Section 6 concludes, and an appendix contains plots and computational experiment data not included in the main article.

1.1 Literature Review

The knapsack problem and its generalizations have been studied for over sixty years, having wide-reaching applications in areas including budgeting, finance, and scheduling; see [18, 25]. Various versions of the knapsack problem under uncertainty have specifically received much attention; [18, Chapter 14] surveys some of these results. Generally, optimization under uncertainty can be modeled via a *static* approach that chooses a solution a priori, or a dynamic (or *adaptive* [9, 10, 11]) approach that chooses a solution sequentially based on realized parameters. A priori knapsack models with uncertain item values include [7, 16, 27, 31, 32], a priori models with uncertain item sizes include [13, 14, 19, 20], and an a priori model with both uncertain values and sizes is examined in [26]. Dynamic models for the knapsack problem with uncertain item sizes include [4, 9, 11, 12, 15], and [17] study a dynamic model with uncertain item values. Another variant, the *stochastic and dynamic* knapsack, assumes items are not initially given but rather arrive dynamically according to a stochastic process [21, 22, 28] .

The dynamic variant of the stochastic knapsack problem examined here was first studied in [12], where the item sizes are exponentially distributed; in this particular case, the natural greedy policy that inserts items based on their value-to-mean-size ratio is optimal, and recent work [3, 6] establishes that this policy is in fact asymptotically optimal more generally. The first study of the

problem in its full generality came in [9, 11], where the authors provide two linear programming bounds with polynomially many variables. They show both bounds are within a constant multiplicative gap using greedy heuristic policies. Later, [15, 23, 24] investigated bounds under the additional assumption that the capacity and item sizes are integers, evaluating the performance of LP relaxations of pseudo-polynomial size and developing randomized policies based on their optimal solutions. Their results also apply to variants beyond the problem studied here, such as correlated random item values, preemption, and the multi-armed bandits problem with budgeted learning.

As this variant only allows for at most one failed insertion, the problem setup is motivated by applications with a hard capacity, where attempted insertions consume capacity regardless of their ultimate success. The clearest example of this is scheduling, e.g. in cloud computing, where the capacity may represent either time or a computing resource on a server (CPU, for instance), and items are computing jobs whose resource requirement is not known with precision until they are executed; [30] explicitly examine the stochastic knapsack problem as a subroutine for dynamic resource provision in cloud environments. Another potential application is budgeting, where the knapsack capacity represents a budget, and items are investment opportunities whose precise capital requirements are not known beforehand. In both cases, it is likely that a failed insertion commits resources that cannot then be reassigned after the failure.

Using value function approximations to obtain relaxations and policies for dynamic programs dates back to [1, 8, 29, 33]. The results in [5] provide the technique's first application to a stochastic knapsack model. Our exact algorithm uses a successive refinement of value function approximations to converge to optimality; the only similar work we are aware of is [2], which studies the joint replenishment problem and its generalizations.

2 Problem Formulation and Preliminaries

We have a knapsack with integer capacity $b > 0$ and item set $N := \{1, 2, \ldots, n\}$, where every item i has a deterministic value $c_i > 0$. Item sizes are independent random variables $A_i \geq 0$ with integer support; each distribution is arbitrary but known to the decision maker. An item size is only realized after an attempted insertion. When attempting to insert i, the decision maker faces two possible outcomes: either i fits, and we collect the value c_i and update the remaining capacity; or i is too large and the process ends, i.e. we allow at most one failed insertion. A *policy* stipulates what item to insert and can depend on both the remaining items and remaining capacity. The objective is to maximize the expected value of successfully inserted items.

This problem can be modeled as a *dynamic program* (DP), where every possible state is defined by the set of remaining items $M \subseteq N$ and remaining capacity $s \in [0, b] := \{0, 1, \ldots, b\}$. For a given state (M, s), the allowed possible actions consist of attempting to insert an item $i \in M$. If we define $v^*_M(s)$ as the optimal expected value at state (M, s), the Bellman recursion is

$$v^*_M(s) = \max_{i \in M} \mathsf{P}(A_i \leq s)(c_i + \mathsf{E}[v^*_{M \setminus i}(s - A_i) | A_i \leq s]), \tag{1}$$

with the base case $v^*_\varnothing(s) = 0$. Intuitively, we collect nothing if the attempted item i has size greater than s, but if the item does fit, we not only collect the item's value c_i, but also the optimal expected value of the subsequent state, $(M \setminus i, s - A_i)$. The *linear programming* (LP) formulation of this problem is

$$\min_{v} \; v_N(b) \tag{2a}$$

$$\text{s.t. } v_{M \cup i}(s) \geq \mathsf{P}(A_i \leq s)(c_i + \mathsf{E}[v_M(s - A_i) | A_i \leq s]), \tag{2b}$$

$$i \in N,\ M \subseteq N \setminus i,\ s \in [0, b] \tag{2c}$$
$$v_M : [0, b] \to \mathbb{R}_+, \quad M \subseteq N. \tag{2d}$$

For every $M \subseteq N$, v_M can be thought of as a vector in $\mathbb{R}_+^{b+1}$, or equivalently as a real-valued function over the integers in $[0, b]$.

Notation To ease notation, we denote an item size's cumulative distribution function by $\mathsf{F}_i(s) := \mathsf{P}(A_i \leq s)$ for $i \in N$, and its complement by $\bar{\mathsf{F}}_i(s) := \mathsf{P}(A_i > s)$. The *mean truncated size* of item $i \in N$ at capacity $s \in [0, b]$ is the quantity $\tilde{\mathsf{E}}_i(s) := \mathsf{E}[\min\{s, A_i\}]$ [9, 11]. Intuitively, given remaining capacity s, we do not care about the distribution of the item size past s, because in all such cases we end up with a failed insertion.

2.1 Value Function Approximations - a Survey

In general, the LP (2) cannot be solved efficiently and essentially reduces to running the Bellman recursion (1) in $\Theta(b^2 n 2^n)$ time. However, any feasible solution to the LP provides a valid upper bound on the optimal solution; earlier work in [5] examines the bound that results from approximating the optimal value function as the affine value function approximation (AFF)

$$v_M(s) \approx qs + r_0 + \sum_{i \in M} r_i, \tag{3}$$

where r_i intuitively represents the inherent value of having an item i available to insert, q is the marginal value of the remaining capacity s, and r_0 is the intrinsic value of keeping the knapsack available to the decision maker.

Proposition 2.1 ([5])**.** *The best possible bound given by affine approximation* (3) *is the solution of the LP*

$$\begin{aligned} \min_{q,r}\ & qb + r_0 + \sum_{i \in N} r_i \\ \text{s.t. } & q\tilde{\mathsf{E}}_i(s) + r_0 \bar{\mathsf{F}}_i(s) + r_i \geq c_i \mathsf{F}_i(s), && \forall\ i \in N, s \in [0, b] \\ & r, q \geq 0. \end{aligned}$$

A Quadratic LP (Quad) introduced in [6] improves on AFF by introducing quadratic terms that encode the notion of diminishing returns:

$$v_M(s) \approx qs + r_0 + \sum_{i \in M} r_i - \sum_{\{k,\ell\} \subseteq M} r_{k\ell}. \tag{4}$$

AFF and Quad can be formulated without the assumption of integer support, and are efficiently solvable for many common distributions, such as those whose cumulative distribution function is piecewise convex. Both are also asymptotically tight, in that the ratio of the LP upper bound and a known lower bound (the non-adaptive version of the natural greedy policy provided in Section 5.1, that inserts items according to their profitability ratios) tends to 1 as the number of items tends to infinity, under certain assumptions [6].

Finally, assuming again that item sizes have integer support, [23, 24] proposes a pseudo-polynomial (PP) bound,

$$\max_x \sum_{i \in N} \sum_{s=0}^{b} c_i x_{i,s} \mathsf{F}_i(s)$$

$$
\begin{aligned}
\text{s.t. } & \sum_{i\in N}\sum_{s=\sigma}^{b} x_{i,s}\bar{\mathsf{F}}_i(s-\sigma) \le 1, && \sigma = 0,\dots,b \\
& \sum_{s=0}^{b} x_{i,s} \le 1, && i \in N \\
& x \ge 0,
\end{aligned}
$$

which [5] show arises from the value function approximation

$$v_M(s) \approx \sum_{i\in M} r_i + \sum_{\sigma=0}^{s} w_\sigma. \tag{5}$$

When available, PP is provably tighter than AFF [5] and serves as a benchmark to gauge the performance of polynomially solvable bounds.

The literature up to this point thus illustrates a rather comprehensive picture for the stochastic knapsack problem. The asymptotic result in [6] allows us to look to AFF as a practical and accurate bound when the number of items is very large. When the asymptotic result does not yet apply, the Quad and PP bounds offer provable strengthening. However, these bounds may still exhibit large gaps in many cases. Furthermore, to our knowledge no exact method exists for this problem beyond naively applying the Bellman recursion, so our aim is to design such an algorithm, which can also serve as a proof of concept for other dynamic combinatorial optimization problems.

2.2 A General Reformulation

Our approach pursues the value function approximation route, generalizing the relaxations we have examined thus far. Letting w be defined in the same space as v, consider the value function reformulation

$$v_M(s) = \sum_{U\subseteq M}\sum_{\sigma\le s} w_U(\sigma), \tag{6}$$

which encodes all possible information about a particular (M, s) state: all available remaining capacity states up to s, all available remaining subsets of current subset M, and the interactions between them. Such a representation is notably not unique; for instance, equivalent formulations include

$$v_M(s) = \sum_{U\supseteq M}\sum_{\sigma\le s} w'_U(\sigma), \text{ and } v_M(s) = \sum_{U\subseteq M}\sum_{\rho> s} w''_U(\rho).$$

These transformations all provide different ways of reformulating the state space, as for any function v, we can determine a unique set of w, w', and w''.

Indeed, it is easy to see that (6) is a true reformulation (as opposed to an approximation) by exhibiting a one-to-one correspondence between v and w. Starting with the base term $v_\varnothing(s) = w_\varnothing(\sigma) = 0$, we can recursively solve for w in terms of v by repeatedly using the relation (6) above. For example, we have

$$
\begin{aligned}
& v_i(0) = w_\varnothing(0) + w_i(0) = w_i(0) \\
& v_i(1) = w_\varnothing(0) + w_\varnothing(1) + w_i(0) + w_i(1) \implies w_i(1) = v_i(1) - v_i(0) \\
& v_i(2) = w_i(0) + w_i(1) + w_i(2) \implies w_i(2) = v_i(2) - v_i(1) \\
& \quad \vdots
\end{aligned}
$$

$$v_{ij}(1) = \sum_{U \subseteq \{i,j\}} \sum_{\sigma \leq 1} w_U(\sigma) \implies w_{ij}(1) = v_{ij}(1) - v_i(1) - v_j(1) + v_i(0) + v_j(0) - v_{ij}(0),$$

and so on. The full one-to-one mapping between v and w can be found in a similar manner.

The reformulation (6) generalizes the earlier approximations (3), (4), and (5), that led to the AFF, Quad, and PP bounds, respectively by choosing a subset of the w's to comprise the value function approximation. For instance, the pseudo-polynomial approximation (5) can be rewritten as

$$v_M(s) \approx \sum_{i \in M} w_{\{i\}}(0) + \sum_{\sigma \leq s} w_{\varnothing}(\sigma),$$

which is the special case of (6) that sets $w_U(\sigma) = 0$ if either $|U| = 1$ and $\sigma \neq 0$ simultaneously hold, or if $|U| \geq 2$. This provides a vehicle to develop an exact algorithm to solve the original formulation (2).

Directly using (6) in full would yield an unwieldy LP that is just as difficult as solving the original DP formulation; however, carefully selecting which w's to include in the value function approximation could lead to a tight bound (or a bound within a desired numerical tolerance). In the spirit of cutting plane algorithms used in solving the deterministic knapsack problem, then, one could dynamically improve the value function approximation to systematically reach stronger relaxations with certain algorithm termination. For example, defining the state space as $\mathscr{S} := \{(M, s)\}_{M \subseteq N, s \in [0,b]}$, let us choose as a starting point some $\tilde{\mathscr{S}} \subseteq \mathscr{S}$ to provide the approximation

$$v_M(s) \approx \sum_{\substack{(U,\sigma) \in \tilde{\mathscr{S}} \\ U \subseteq M, \sigma \in [0,s]}} w_U(\sigma) \tag{7}$$

Let $W(\tilde{\mathscr{S}})$ denote the optimal value of the associated approximation LP for (7) under set $\tilde{\mathscr{S}}$; clearly, $W(\tilde{\mathscr{S}}) \geq W(\mathscr{S})$. We determine whether the approximation is tight with a pricing problem

$$\max_{(M,s) \in \mathscr{S} \setminus \tilde{\mathscr{S}}} \big\{ W(\tilde{\mathscr{S}}) - W(\tilde{\mathscr{S}} \cup (M, s)) \big\},$$

or its weaker decision counterpart,

$$\exists\ (M, s) \in \mathscr{S} \setminus \tilde{\mathscr{S}} : W(\tilde{\mathscr{S}} \cup (M, s)) < W(\tilde{\mathscr{S}})? \tag{8}$$

If the current bound is not tight, we can thus identify and add a state to $\tilde{\mathscr{S}}$ and update the current value function approximation, repeating until a tight bound is found. Being able to solve (8) systematically in effect guarantees finite termination of the algorithm since $\mathscr{S}$ is finite. We thus develop a general algorithm based on this framework and present its results in Section 4.

Prior to the algorithm proper, however, we first examine the special case where the capacity is zero, $b = 0$. The motivation for this is at least threefold. First, this case presents a simple scenario to explore theoretically, and any insights, approaches, and results may be applicable to the more general capacitated case. Second, the zero capacity case is always a valid restricted subproblem of the general capacitated case, as all $w_{U,0}$ variables are still present in the general case. From a practical standpoint then, various algorithm heuristics can also be preliminarily tested on the zero capacity subproblem. Finally, the general algorithm requires a good starting bound. In addition to the bounds covered in the previous section, the zero capacity case provides an alternate starting point, and we verify further below that the bound is a useful initialization for certain computational experiments.

3 Zero Capacity Case

We now limit ourselves to the case with zero capacity, $b = 0$. Without loss of generality, we assume all items have a positive probability of having size 0. The item sizes can thus be thought of as Bernoulli random variables, although the analysis below does not require this. Throughout this section, we consider the simplified notation $p_i := \bar{\mathsf{F}}_i(0)$ and $q_i := 1 - p_i = \mathsf{F}_i(0)$.

3.1 Structure of the Optimal Policy

We first observe that any deterministic policy simply prescribes an item to insert given any subset of remaining items. Therefore, an optimal (deterministic) policy is a sequence of item insertion attempts that continues until a failure. This simplified structure allows us to determine closed-form solutions for both the policy and the value function.

Lemma 3.1. *Suppose items are indexed to satisfy*

$$\frac{q_i}{p_i} c_i \geq \frac{q_{i+1}}{p_{i+1}} c_{i+1}, \quad i = 1, \ldots, n-1.$$

The policy that attempts to insert items in this order is optimal. Furthermore, for the same problem restricted to a subset of items $M \subseteq N$, the same policy restricted to the subsequence defined by M is optimal.

Proof. Let $V(1, \ldots, n)$ denote the expected value of the sequence $1, \ldots, n$. It is clear that

$$V(1, \ldots, n) = q_1(c_1 + q_2(c_2 + \cdots + q_n c_n)) = q_1 c_1 + q_1 q_2 c_2 + \cdots + q_1 q_2 \ldots q_n c_n.$$

Suppose we interchange items i and $i+1$ to form a new sequence. Then the change in value from the original sequence (new versus old) would be

$$\begin{aligned} \Delta V &= r_{i-1} q_{i+1} c_{i+1} + r_{i-1} q_{i+1} q_i c_i - r_{i-1} q_i c_i - r_{i-1} q_i q_{i+1} c_{i+1} \\ &= r_{i-1}(q_{i+1} c_{i+1} + q_{i+1} q_i c_i - q_i c_i - q_i q_{i+1} c_{i+1}) \\ &= r_{i-1}(q_{i+1} p_i c_{i+1} - q_i p_{i+1} c_i), \end{aligned}$$

where $r_i := q_1 q_2 \ldots q_i$. Thus, $\Delta V \leq 0$, and the original sequence $1, \ldots, n$ would be better, if

$$\frac{q_i}{p_i} c_i \geq \frac{q_{i+1}}{p_{i+1}} c_{i+1}.$$

Since N is a finite set, we can thus perform a finite number of such exchanges from any starting sequence. Since any deterministic policy corresponds to some item sequence, the result follows. The same argument applies to any subset $M \subseteq N$. □

Let the *optimal ordering* refer to labeling the items such that the ratios $\frac{q_i}{p_i} c_i$ are non-increasing. This optimal greedy policy is slightly different from a greedy heuristic policy studied in the capacitated case in past work, e.g. [5, 11], which uses the profitability ratio $c_i \mathsf{F}_i(b) / \tilde{\mathsf{E}}_i(b)$. The latter does not apply here since the denominator goes to 0; however, we can intuitively think of the optimal ordering in the zero-capacity case as the "limit" of the heuristic policy as $b \to 0$. For sufficiently small b, we have

$$\tilde{\mathsf{E}}_i(b) = \mathsf{F}_i(b) \mathsf{E}[A_i | A_i \leq b] + b \bar{\mathsf{F}}_i(b) \approx b \bar{\mathsf{F}}_i(b),$$

with equality holding under the integer support assumption for $b < 1$. Thus the ratio orderings are preserved. The results in [3, 6] show asymptotic optimality of the greedy policy, i.e. optimality as

b grows large; coupled with Lemma 3.1, this implies the natural greedy policy is optimal at both extremes, zero and very large capacity.

We conclude this section with an auxiliary structural result on the optimal value function that follows from the analysis in Lemma 3.1. Though we do not use it in our later results, it may be of independent interest.

Corollary 3.2. *The optimal expected value function $v^*_M(0)$ is submodular, i.e.*

$$v^*_{M\cup i}(0) - v^*_M(0) \geq v^*_{M\cup\{i,j\}}(0) - v^*_{M\cup j}(0), \quad M \subsetneq N, i, j \in N \setminus M.$$

We omit the proof for brevity.

3.2 The Value Function Reformulation

Recall the reformulation $v^*_M(0) = \sum_{U\subseteq M} w^*_U(0)$, a special case of (6). As it turns out, there also exists an explicit closed form solution for the w variables in the zero-capacity case.

Proposition 3.3. *Let w^* refer to the corresponding w variables computed from the optimal value function v^*, and let the set of items $N = \{1, \ldots, n\}$ be indexed according to the optimal ordering. Then*

$$w^*_N(0) = (-1)^{|N|+1} q_n c_n \prod_{i\in N\setminus n} p_i. \tag{9}$$

Additionally, the same corresponding closed form applies for any $M \subseteq N$.

The proof of this proposition can be found in the Appendix. Intuitively, the proposition says that the w_M's exponentially decay in value as we add more items to M, as long as the highest indexed element is the same. In other words, the impact or marginal value diminishes as one adds more elements to the starting set, which is consistent with the intuition that the initial decisions are the most important. Smaller sets thus "matter" more in the value function LP; this further motivates why we start with lower cardinality sets in all of our computational experiments.

The results in this and the previous subsection solve the zero capacity case completely from both the bound and policy sides, and helps us quickly benchmark the general algorithm on the zero-capacity case.

4 The Algorithm

Recall the problem reformulation (6), $v_M(s) = \sum_{U\subseteq M}\sum_{\sigma\leq s} w_U(\sigma)$. Plugging in this value function formulation into the value function LP (2) yields constraints with left hand sides of the form

$$\begin{aligned} v_{M\cup i}(s) - \mathsf{F}_i(s)\mathsf{E}[v_M(s-A_i)|A_i \leq s] &= \sum_{U\subseteq M\cup i}\sum_{\sigma\leq s} w_U(\sigma) - \sum_{s'\leq s}\mathsf{P}(A_i = s')\Big(\sum_{U\subseteq M}\sum_{\sigma\leq s'} w_U(\sigma)\Big) \\ &= \sum_{U\subseteq M\cup i}\sum_{\sigma\leq s} w_U(\sigma) - \sum_{U\subseteq M}\sum_{\sigma\leq s}\mathsf{F}_i(s-\sigma)w_U(\sigma) \\ &= \sum_{U\subseteq M}\sum_{\sigma\leq s} w_{U\cup i}(\sigma) + \bar{\mathsf{F}}_i(s-\sigma)w_U(\sigma). \end{aligned}$$

Thus, our master dual LP is

$$\min_w \sum_{U\subseteq N}\sum_{\sigma\leq b} w_U(\sigma) \tag{10a}$$

$$\text{s.t.} \sum_{U \subseteq M} \sum_{\sigma \leq s} w_{U \cup i}(\sigma) + \bar{\mathsf{F}}_i(s-\sigma) w_U(\sigma) \geq c_i \mathsf{F}_i(s), \qquad \forall\, i \in N, M \subseteq N \backslash i, s \in [0, b] \tag{10b}$$

$$w_{\varnothing,\sigma} = 0, \qquad \forall\, \sigma \leq b. \tag{10c}$$

Note the empty set variables in (10) are required to be 0 via reformulation (6), as the original formulation assumes as a base case that $v^*_\varnothing(s) = 0$, i.e. we cannot assign a positive value to states without any items to insert. The remaining variables are unrestricted. The corresponding primal problem is

$$\max_x \sum_{i \in N} \sum_{M \subseteq N \backslash i} \sum_{s \leq b} c_i \mathsf{F}_i(s) x_{iMs} \tag{11a}$$

$$\text{s.t.} \sum_{i \notin U} \sum_{\substack{M \subseteq N \backslash i \\ M \supseteq U}} \sum_{s=\sigma}^{b} \bar{\mathsf{F}}_i(s-\sigma) x_{iMs} + \sum_{i \in U} \sum_{\substack{M \subseteq N \backslash i \\ M \supseteq U \backslash i}} \sum_{s=\sigma}^{b} x_{iMs} = 1, \qquad \forall\, \varnothing \neq U \subseteq N, \sigma \leq b \tag{11b}$$

$$x \geq 0. \tag{11c}$$

Instead of solving these LP's directly, we propose an algorithm that uses column and constraint generation. Before formalizing the algorithm, we first examine both the separation and pricing problems. Given a solution w to (10), the separation problem for a fixed pair $(i \in N, s \leq b)$ is

$$\min_{M \subseteq N \backslash i} \sum_{U \subseteq M} \sum_{\sigma \leq s} w_{U \cup i}(\sigma) + \bar{\mathsf{F}}_i(s-\sigma) w_U(\sigma). \tag{12}$$

Since s is fixed, once an item j is included in the set M, all $w_U(\sigma)$ variables with $\sigma \leq s$ are included in the objective. Thus, we can rewrite problem (12) as an integer program with binary decision variables corresponding to whether or not an item belongs to M. Let

$$\tilde{w}_U = \sum_{\sigma \leq s} w_{U \cup i}(\sigma) + \bar{\mathsf{F}}_i(s-\sigma) w_U(\sigma).$$

Then, (12) is equivalent to

$$\min_{y,z} \sum_{M \subseteq N \backslash i} z_U \tilde{w}_U \tag{13a}$$

$$\text{s.t. } z_U \leq y_j, \qquad \forall\, U \subseteq N \backslash i, j \in U \tag{13b}$$

$$z_U \geq \sum_{j \in U} y_j - (|U| - 1), \qquad \forall\, U \subseteq N, \tag{13c}$$

$$y_i, z_U \in \{0, 1\}, \tag{13d}$$

where the constraints and z_U variables need only be defined for sets U with nonzero $\tilde{w}_U$. Our next result shows that we cannot hope to be more efficient in separation than solving this IP formulation.

Proposition 4.1. *Problem* (12) *is NP-hard, even in the case where only the $\tilde{w}_U$ variables with $|U| \leq 2$ are nonzero.*

The proof follows a fairly standard reduction from the max-cut problem, and we omit it for brevity. To further clarify the proposition statement, the w values, which may be exponentially many in n, are part of the input, as in set packing/cover problems. In our experiments, we find this number to typically be of reasonable size compared to the number of items.

On the other hand, the pricing problem for (10) involves both a maximization and minimization version, because (11) has equality constraints. Given a solution x, for each fixed $\sigma \in \{0, 1, \ldots, b\}$, the maximization problem is

$$\max_{U \subseteq N} \sum_{i \notin U} \sum_{\substack{M \subseteq N \setminus i \\ M \supseteq U}} \sum_{s=\sigma}^{b} \bar{\mathsf{F}}_i(s-\sigma) x_{iMs} + \sum_{i \in U} \sum_{\substack{M \subseteq N \setminus i \\ M \supseteq U \setminus i}} \sum_{s=\sigma}^{b} x_{iMs}, \tag{14}$$

and the minimization problem is its analogue. Both problems can be modeled as integer programs in a similar fashion to (13), and this is how we solve them. We have not been able to establish their complexity, though we conjecture that they are also NP-hard.

The main idea behind the algorithm is to iteratively generate $w_U(\sigma)$ variables and solve the corresponding restriction of the master LP (10), to provide increasingly tighter upper bounds on the optimal solution until we reach the desired numerical tolerance. Recall that any feasible w to the master problem (10) gives a valid upper bound, that is, the tentative w must satisfy all of the constraints in problem (10). Hence, in each iteration of the algorithm we must check for feasibility via constraint generation through several solves of subproblem (12). It is therefore additionally necessary to ensure feasibility during the algorithm initialization phase, prior to any column generation. On the primal side, however, we can also iteratively test heuristic policies (based on the candidate dual solution), which provide valid lower bounds. Recall that any candidate solution w of (10) implies also an approximate value function v through the linear transformation (6); this value function approximation in turn has an associated policy given by using it inside the Bellman recursion. That is, given any value function approximation $v_M(s)$, at each state (M, s) we insert

$$\arg\max_{i \in M} \mathsf{F}_i(s)(c_i + \mathsf{E}[v_{M \setminus i}(s - A_i) | A_i \leq s]).$$

Using the reformulation (6), we can rewrite the conditional expectation above as

$$\mathsf{E}[v_{M \setminus i}(s - A_i) | A_i \leq s] = \frac{1}{\mathsf{F}_i(s)} \sum_{\rho \leq s} \mathsf{P}(A_i = \rho)\, v_{M \setminus i}(s - A_i)$$

$$= \frac{1}{\mathsf{F}_i(s)} \sum_{\rho \leq s} \mathsf{P}(A_i = \rho) \sum_{U \subseteq M \setminus i} \sum_{\sigma \leq s-\rho} w_U(\sigma) = \frac{1}{\mathsf{F}_i(s)} \sum_{\sigma \leq s} \sum_{U \subseteq M \setminus i} w_U(\sigma) \mathsf{F}_i(s - \sigma),$$

which in turn simplifies the policy into

$$\arg\max_{i \in M} \mathsf{F}_i(s) c_i + \sum_{\sigma \leq s} \mathsf{F}_i(s - \sigma) \sum_{U \subseteq M \setminus i} w_U(\sigma). \tag{15}$$

Thus, at each candidate solution, we can evaluate the associated policy via simulation and track the best policy value found so far. Having both an updated bound and policy value allows us to calculate an optimality gap at each iteration of the algorithm. Algorithm 1 below formalizes our discussion thus far.

We know Algorithm 1 must terminate, since there are a finite number of w variables in total.

Theorem 4.2. *Algorithm 1 terminates with an optimal solution in finitely many iterations.*

Our guarantee of termination in finite time is notably weaker than the $\Theta(b^2 n 2^n)$ complexity of the Bellman recursion. Nevertheless, our aim is to reach a reasonable numerical gap in a practical running time, especially for instances in which DP is intractable. We demonstrate this in computational experiments below.

Algorithm 1 Column and Constraint Generation Algorithm for the Stochastic Knapsack Problem

Inputs:
Subsets of state space $\tilde{\mathscr{S}} \subseteq \{(U, \sigma)\}_{U \subseteq N, \sigma \leq b}$ and constraint space
$\tilde{\mathscr{C}} \subseteq \{(i, M, s)\}$
Primal numerical tolerance ϵ and dual numerical tolerance δ
Initialize:
$\text{ALP}_{start} \leftarrow$ problem (10) restricted to variables in $\tilde{\mathscr{S}}$ and constraints in $\tilde{\mathscr{C}}$
ALP $\leftarrow \text{ALP}_{start}$ with generated constraints via problem (12) (for each (i, s) until primal feasible)
Soln $\leftarrow$ solution to ALP_{start}
Pol $\leftarrow$ simulated associated policy value according to (15)
while $\frac{Soln}{Pol} > 1 + \epsilon$ **do**
  Generate columns for ALP via pricing problems (14) (for each σ) and update $\tilde{\mathscr{S}}$
  if $\nexists$ generated columns **then**
    Declare optimal, return Soln
  else
    Soln $\leftarrow$ solution to ALP.
    if ALP unbounded **then**
      Return extreme ray, generate constraints via separation problem (12) (for each (i, s))
    else
      Return incumbent solution, generate constraints via problem (12) (for each (i, s))
    if $\nexists$ generated constraints **then** declare primal feasible
      TempPol $\leftarrow$ simulated policy (15) with current w
      if TempPol $>$ Pol **then** Pol $\leftarrow$ TempPol
      go to 3
    else
      Incorporate new constraints into ALP, resolve ALP.
      Soln $\leftarrow$ solution to ALP, **go to** 9
Return Soln, Pol

Even though Theorem 4.2 guarantees dual optimality, the primal side is more complicated. We cannot necessarily guarantee that the heuristic policies are monotonically non-decreasing, or even that the policy corresponding to the optimal solution is also optimal. We illustrate with the following simple example: Consider a two-item problem in which both items have deterministic size b and $c_1 > c_2$. Then (10) becomes

$$\begin{aligned} \min\ & v_N(b) \\ \text{s.t. } & v_N(b) - v_1(0) \geq c_2 \\ & v_N(b) - v_2(0) \geq c_1 \\ & v_1(0),\ v_2(0) \geq 0, \end{aligned}$$

which has as one extreme point optimal solution $v_N^*(b) = c_1, v_1^*(0) = c_1 - c_2, v_2^*(0) = 0$. However, policy (15) with this value function cannot distinguish between inserting item 1 or item 2 for the first insertion. In the event that the policy chooses to insert item 2 first, it will yield the suboptimal profit $c_2 < c_1$. In this simple example, the problem can be avoided by checking the corresponding primal optimal solution, which disambiguates the apparently equally good choices and identifies item 1 as the real optimal choice. However, it is unclear whether the primal x solutions can be similarly leveraged in approximate cases, when the dual bound is not tight. Furthermore, our computational experiments provide empirical evidence that the algorithm policies are *not* guaranteed to systematically provide non-decreasing lower bounds in larger, more complex instances.

5 Computational Experiments

We executed several computational experiments to empirically evaluate the algorithm above. To obtain stochastic knapsack instances, we used deterministic knapsack instances as a "base". From each deterministic instance, we generated seven stochastic ones by varying the item size distribution and keeping all other parameters the same. Given that a particular item i has size integer size a_i in the deterministic instance, we generated seven discrete probability distributions:

D1 0 or $2a_i$ each with probability 1/2.

D2 0 or $2a_i$ each with probability 1/4, a_i with probability 1/2.

D3 0 with probability 2/3 or $3a_i$ with probability 1/3.

D4 0 with probability 3/4 or $4a_i$ with probability 1/4.

D5 0 with probability 4/5 or $5a_i$ with probability 1/5.

D6 $a_i - \lceil a_i/5 \rceil$ or $a_i + \lceil a_i/5 \rceil$ each with probability 1/2

D7 $a_i - \lceil a_i/3 \rceil$ or $a_i + \lceil a_i/3 \rceil$ each with probability 1/2

All the distributions are designed so an item's expected size equals a_i. Our motivation for testing the first five distributions is at least threefold. First, these were all tested in [6], comparing the AFF, Quad and PP bounds described in Section 2; we thus wish to use the same instances under the algorithm for the sake of consistency. Second, in preliminary experiments, we observed that the Bernoulli-type instances (D1, D3, D4, D5) exhibited a significant starting gap of 20-30%, while the other instances with smaller variance (D2, D6, D7) tend to have a smaller starting gap of less than 10%; these distributions allow for a sound range of initial gaps. Lastly, we included distributions

D6 and D7 to eliminate the possibility of items with size zero (which was always present in previous experiments), and more generally to evaluate the algorithm in less extreme cases.

The deterministic base instances were generated from the advanced knapsack instance generator from `www.diku.dk/~pisinger/codes.html`. We generated thirty total instances: ten with 10 items, ten with 20 items, and ten with 30 items. Within each item number category, five instances had correlated sizes and profits, while the other five had uncorrelated sizes and profits. The fill rate varies between 2 and 6, and the sizes and capacity are scaled appropriately such that the capacity is always 50. The motivation for these item numbers is to evaluate the algorithm under various circumstances. In preliminary experiments, the original DP formulation can solve 10-item instances in a few minutes; the algorithm tests here help identify areas where the algorithm performs best. For 20 items, the DP formulation takes around 24 hours to complete and is effectively the practical limit for this method. Lastly, examining the 30-item instances provides an environment where the DP formulation is effectively impossible. As such, the 10- and 20-item instances only run Algorithm 1 from the bound side and are compared to the true optimal solution taken from the DP, while the 30-item instances run Algorithm 1 from both the bound and policy sides.

We ran preliminary experiments using CPLEX 12.6.1 for all LP solves on a MacBook Pro with OS X 10.11.4 and a 2.5 GHz Intel Core i7 processor. The preliminary experiments suggest that Algorithm 1 may take several hours depending on the instance; for example, 20-item instances typically completed anywhere between 14-17 master loops in 24 hours, where a master loop is defined as a complete column and constrain generation iteration of the algorithm. We ran our full experiment suite in parallel on the Georgia Tech ISyE Computing Cluster using Condor, with different machines of varying processing speeds and memory. All 10-item instances were run under an eight-hour time limit and 0.1% optimality gap threshold, stopping whenever either was reached. On the other hand, every 20- and 30-item instance ran Algorithm 1 for 16 master loops, regardless of time limit, to provide for a more fair comparison and to compensate for the hardware differences due to parallelization. Prior to discussing the computational results, we first further elaborate on the algorithm heuristics in the following subsection, to provide a more complete picture of the algorithm parameters utilized.

5.1 Implementation Details

Regarding the starting bound used to initialize the algorithm, after preliminary experiments we chose the PP bound for all instances except those given by distributions D2. Recall that this bound stems from restricting the reformulation (6) into (5), $v_M(s) = \sum_{i \in M} w_{i,0} + \sum_{\sigma \leq s} w_{\varnothing,\sigma}$. As introduced in [23, 24] and shown in [5], this starting bound is known to be a feasible solution to the master problem (10) and thus does not require initial constraint generation. In particular, PP arises when we choose:

$$\tilde{\mathscr{S}} = (\{i\}, 0)_{i \in N} \cup (\varnothing, \sigma)_{\sigma \in [0,b]}, \text{ and } \tilde{\mathscr{C}} = (i, \varnothing, s)_{i \in N, s \in [0,b]}.$$

Alternatively, for instances given by distributions D2, we first solve the corresponding subproblem restricted to the zero-capacity case examined in Section 3. In other words, we wish to solve for all $w(0)$, that is,

$$v_M(0) = \sum_{U \subseteq M} w_{U,0}.$$

However, as solving the corresponding linear program would require exponentially many constraints, we instead apply the exact same Algorithm 1 to the case that $b = 0$ to generate variables and

constraints. To achieve this, we start with the initial value function approximation

$$v_M(s) = w_{\varnothing,0} + \sum_{i \in M} w_{i,0}$$

and proceed to run Algorithm 1 under the restriction that $b = 0$; that is,

$$\tilde{\mathscr{S}} = (\{i\}, 0)_{i \in N} \cup (\varnothing, 0) \text{ and } \tilde{\mathscr{C}} = (i, \varnothing, 0)_{i \in N}.$$

A time limit of thirty minutes was imposed for running the algorithm at this initialization step, as we are merely generating starting variables and constraints for the original problem with nonzero capacity b. After initialization, if the resulting bound for the zero capacity case is still infeasible under the general capacitated case, we then also perform additional constraint generation to find a feasible solution prior to continuing with column generation. Finally, we also considered the Quad bound given by restriction (4) as a potential starting point; however, the bound performed relatively poorly in preliminary experiments.

It is worth nothing that we are already starting with the best performing bounds from the literature; [5, 6] have favorably compared the pseudopolynomial bound (5) to several other bounds in literature and show how PP is the better bound. However, their computational experiments often demonstrated a loose gap remaining. Using such starting points provides as fair an evaluation as possible for our algorithm, as we are immediately comparing it to the state-of-the-art bound (as opposed to many of the previously studied looser bounds).

For the 30-item instances, we simulated two initial heuristic policies to generate a starting lower bound. The first is the Greedy policy outlined in Section 3; the second is an adaptive version of this heuristic that recomputes profitability ratios every time it must make a decision, using the current remaining capacity. This natural *adaptive greedy* policy does not fix an ordering of the items, but rather at every encountered state (M, s) computes the profitability ratios at current capacity $c_i \mathsf{F}_i(s)/\tilde{\mathsf{E}}_i(s)$ for remaining items $i \in M$ and chooses a maximizing item. This is equivalent to resetting the greedy order by assuming (M, s) is the initial state. This latter policy is shown to be computationally effective in [5].

We also considered various heuristics for column and constraint generation. For column generation, these included the following ideas. We can choose to only solve the pricing problem for a subset of σ values instead of all $b + 1$ choices in some rotating or staggered fashion (e.g. all even integers in one iteration, all odd integers the next), to reduce the time that a single iteration may take. Additionally, we attempted column deletion, where we either limit the absolute maximum number of variables, the maximum number of variables per σ, and/or the maximum number of iterations that a variable can remain non-basic. In every column deletion test, we only removed non-basic variables, even if this caused us to exceed the stipulated limit. Ultimately, our preliminary experiments suggested that column deletion may have the temporary benefit of faster initial loops, but exhibits slower overall progress in later loops. We observed a similar tradeoff when running staggered pricing problems. In addition, column deletion did not always work as planned, since a significant fraction of the variables were usually basic. We thus did not implement column deletion or staggered pricing in the final experiments.

We assessed similar heuristics for constraint generation. Analogously to choosing a subset of σ values to solve the pricing problem, for each fixed i we can choose a subset of s values to solve the separation problem, again in some rotating or staggered manner. Additionally, we can also delete constraints, limiting the maximum number of constraints for each item i, the maximum number of constraints for any given (i, s) pair, or the maximum number of consecutive iterations that a constraint remains inactive. Preliminary experiments suggested that both bounding constraints

with respect to i only and the number consecutive inactive iterations did not significantly affect performance. On the other hand, bounding the number of constraints for each (i, s) pair seemed to improve overall performance by preventing an intermediate LP solve from taking too long; we thus implemented a bound of anywhere between 50 and 75 depending on the instance.

To help with computational tractability, we also considered a natural greedy heuristic as an alternative to solving the separation and pricing problems' IP formulations. Considering either problem as an optimization over subsets M, we apply the greedy heuristic often used in submodular optimization: Starting with $M = \varnothing$, we repeatedly add the remaining item that most improves the objective, until no item does so, and return this final set. Unfortunately, under preliminary tests, the greedy heuristics often took longer to finish; it appears that repeatedly evaluating the objective function for different sets is already too time-consuming compared to solving the corresponding IP. We thus opt to solve the IP formulations in our computational experiments below.

Lastly, we record the various numerical tolerances. We used the primal numerical tolerance $\epsilon = 0.1\%$ as the optimality gap threshold. We used dual numerical tolerance $\delta = 0.01\%$ to determine whether or not a constraint is violated when determining feasibility under constraint generation. Finally, we used a 0.1% threshold when performing the pricing problem to determine whether a prospective variable should be included.

5.2 Discussion

Tables 1, and 2 provide summary results for the 20- and 30-item experiments, respectively; summary results for the 10-item instances are in the appendix for brevity since the instances are relatively small. For all tables, the initial and final gaps refer to the geometric mean of the gaps across all instances of a particular distribution; thus, the closer the number is to 0%, the better the bound. The *relative remaining gap* (RRG) refers to how much of the initial gap was closed over the course of the algorithm (for example, if we start with an initial gap of 50% and end with a final gap of 20%, the relative remaining gap is 40%). Because the initial gap, which is the best empirically performing bound in literature, can often vary by number of items and item size distribution, we consequently also look to RRG (in addition to absolute gaps) to better compare the instances. Recall that all 10-item instances were run under an eight-hour time limit and a 0.1% optimality gap threshold, stopping whenever either was reached. The motivation behind such instances was to gain insight into the types of instances for which the Algorithm performs well; as such, we note that uncorrelated instances, as well as including 0 in the support and smoothing the distribution, can all make closing the final gap difficult. Full data can be found in the Appendix.

Every 20- and 30-item instance ran Algorithm 1 for 16 "master" loops, regardless of time limit, to provide for a more fair comparison and to compensate for the hardware differences due to our parallelized experiment runs using Condor. As such, instead of run time, we provide the "Average Inner Loops" metric, which is the average number of constraint generation loops needed for a given master loop of the algorithm; this is an alternate way to compare the instance difficulty across distributions. From Tables 1 and 2, the inner loop averages do not seem to have a clear correlation with the relative remaining gap. Instead, generally speaking, we observe that higher-variance distributions correlate with a smaller relative remaining gap. In fact, we see that D4 and D5 have their initial gaps being cut by more than half, while D2 and D3 see an improvement of at least a 25% relative gap decrease. With respect to absolute gap improvements, the more extreme Bernoulli D3, D4, and D5 instances ended below a final gap of 10%, many of which also observed an absolute gap decrease of 10%. The other, lower variance distributions in turn saw a 1 to 2% decrease in gap; these instances also exhibited a smaller starting gap. Intuitively speaking, these high variance distributions are the most different from the deterministic counterpart. Thus, our

Table 1: 20-Item Instances Summary

Dist	Initial Gap	Final Gap	RRG	Avg. Inner Loops
D1	7.83%	6.27%	81.79%	19.79
D2	5.12%	3.64%	71.99%	22.84
D3	13.93%	8.45%	58.83%	26.34
D4	19.51%	7.74%	30.78%	21.38
D5	19.53%	9.63%	38.01%	19.12
D6	3.31%	2.18%	65.44%	18.18
D7	2.81%	2.15%	76.21%	24.00

algorithm seems to work well for instances that most deviate from the deterministic case, where perhaps a less complex algorithm or heuristic may suffice.

Recall that the 30-item instances do not have an optimal solution to benchmark against and must run the corresponding policy for each tentative value function approximation after every master loop. Hence, the "bound gap closed" and "policy gap closed" columns in Table 2 refer to the relative gap closed when we only observe the progress made via the bounds and policies, respectively (the higher the percentage, the more progress made). As with the 20-item instances, there does not seem to be a strong relationship between the inner loop averages and the distribution, although a larger number of primal loops roughly corresponds to a smaller relative remaining gap. Regarding absolute gap, as with 20-item instances, the more extreme Bernoulli distributions D3, D4, and D5 saw the most improvement and ended within a 10 to 20% final gap. The other less extreme distributions, which again exhibit a smaller starting gap, close the gap by 1 to 2% and see a final absolute gap of less than 10%. What is also particularly interesting about Table 2 is that the correlated instances see significantly better improvement than the uncorrelated instances, and that the majority of the improvement comes from the policy side (as much as 40 to 50% RRG, or 10 to 30% absolute gap). As correlated instances deterministically have their size and value aligned in some way, they are more likely to have items with similar value-to-size ratios. This makes it more difficult for our natural greedy policies to discriminate between items to insert for a given state, and they can thus perform rather poorly. Therefore, our algorithm has the ability to provide significantly better policies when the natural policies are insufficient. On the other hand, the uncorrelated instances generally do see a better improvement from the bound side in that the uncorrelated bound gap closed is strictly better across all distributions from their correlated counterparts. All in all, our algorithm is able to improve the gap in the areas that need it the most, depending on the whether the initial bound or initial policy is more lacking.

To help narrow down the types of instances that allow for better progress via Algorithm 1, Figures 1 through 3 examine various parameters for the 20-item instances against the relative remaining gap. These plots suggest that higher fill rates, higher distribution variance, and a higher initial gap are all positively correlated with a lower relative remaining gap. Regarding fill rate, higher fill rates imply that an individual item will have a greater impact on the solution. Intuitively, then, the problem is less complex in that fewer item insertions on expectation are needed to fill the knapsack; this is reflected in the algorithm observing greater progress per master loop. The same intuition applies to the distribution variance, as the higher the variance of each item size, the greater the impact an item potentially has. Additionally, these more extreme distributions tend to have a higher observed starting gap. Since our algorithm is general-purpose, it is able to best improve the gap when there is a greater initial gap to close, i.e. when the original bounds do not perform as well. Such behavior is often observed with other exact algorithms in practice,

Table 2: 30-Item Instances Summary

Type	Dist	Initial Gap	Final Gap	RRG	Bnd. GapClosed	Pol. GapClosed	AvgInnerLoops
Cor	D1	13.45%	7.30%	58.06%	2.28%	39.89%	6.16
	D2	4.53%	4.42%	97.31%	2.69%	0.00%	-
	D3	23.37%	11.14%	52.89%	3.57%	43.59%	9.29
	D4	34.84%	14.56%	51.25%	4.52%	45.00%	9.87
	D5	48.34%	17.62%	46.98%	3.27%	49.70%	15.10
	D6	3.52%	3.38%	95.25%	4.75%	0.00%	14.03
	D7	3.18%	2.91%	92.50%	2.23%	5.38%	16.22
Uncor	D1	9.29%	9.00%	97.04%	2.66%	0.29%	11.22
	D2	3.61%	3.29%	90.79%	8.69%	0.56%	-
	D3	13.07%	12.45%	95.40%	4.60%	0.00%	12.29
	D4	20.85%	18.23%	88.14%	11.86%	0.00%	7.56
	D5	25.02%	21.06%	84.73%	15.27%	0.00%	25.48
	D6	3.06%	2.47%	83.80%	8.05%	8.87%	10.57
	D7	2.39%	2.23%	92.55%	3.46%	4.00%	9.68

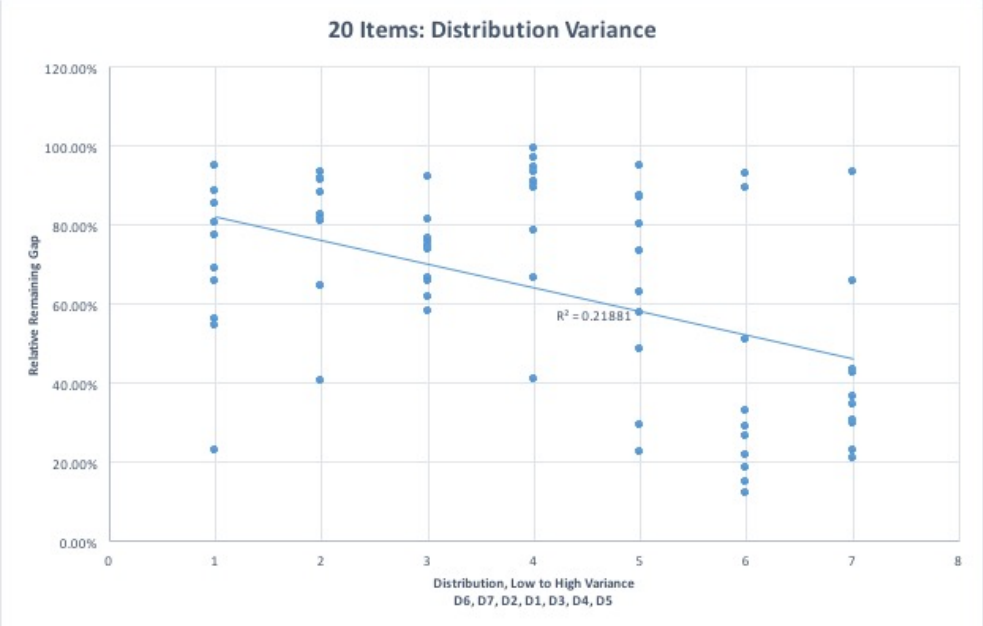

Figure 1: 20 Items - Distribution Variance vs. Relative Remaining Gap

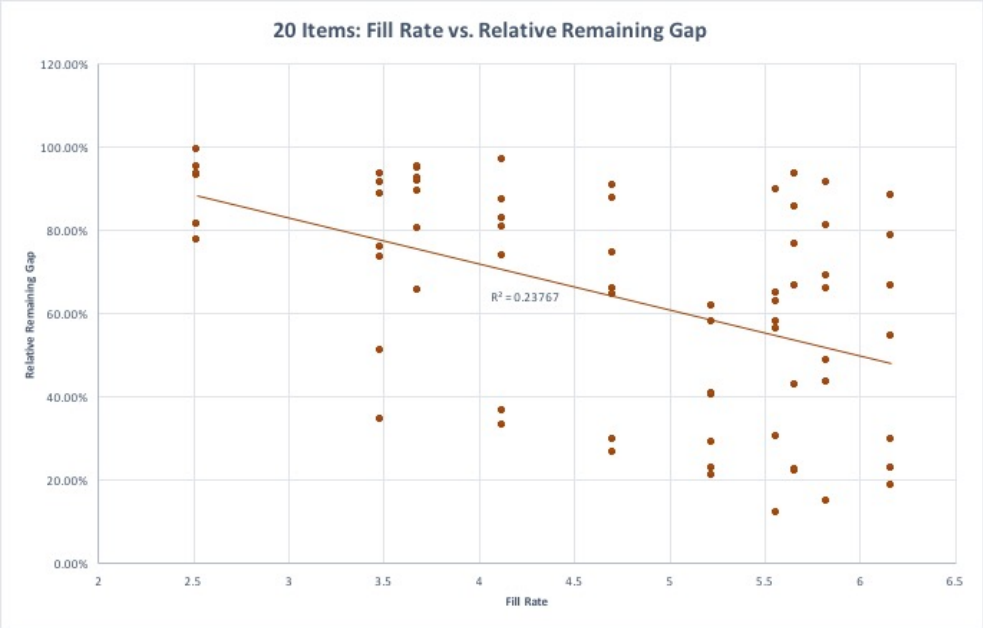

Figure 2: 20 Items - Fill Rate vs. Relative Remaining Gap

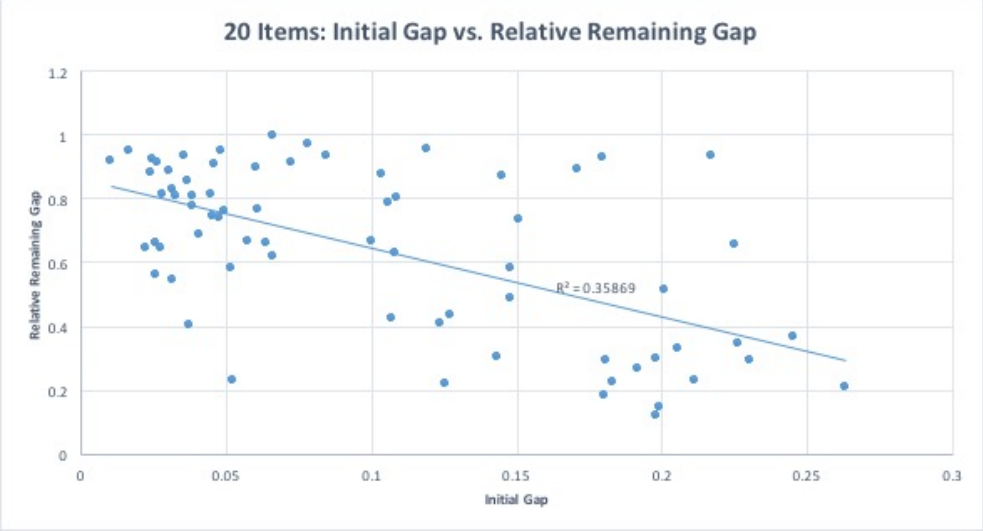

Figure 3: 20 Items - Initial Gap vs. Relative Remaining Gap

such as exact algorithms for (deterministic) integer programs, where decreasing the gap becomes increasingly difficult as we approach optimality. The same plots for the 30-item instances reflect similar results and are thus presented in the Appendix for the sake of brevity.

Another metric we consider is the relative remaining gap closed per loop, which is the total relative gap closed divided by the number of master loops performed in the algorithm run. This metric provides for a fairer comparison between the instances that were unable to complete the specified 16-loop limit due to machine-specific use limitations under the Georgia Tech ISyE Computing Cluster. We also compared and plotted the fill rates, distribution variance, and initial gap to this second metric, but the results are very similar and thus omitted here. For these additional plots, please refer to the Appendix.

We also consider the cardinality of set sizes $|U|$ of all generated $w_U(\sigma)$ variables at the end of the algorithm. For brevity, discussion for the smaller 10-item instances can be found in the Appendix. However, we do note here that our main inference from such instances is that, at optimality, less extreme, non-Bernoulli instances favor variables that correspond to larger item set sizes. This is because individual items have a relatively smaller impact, as opposed to the more extreme Bernoulli instances.

Figures 4 examine set size distributions for the 20- and 30-item instances. These instances did not exhibit a noticeable difference between the Bernoulli and non-Bernoulli distributions and are thus each presented as a single summary plot. As these instances did not finish at optimality, their plots speak more to the initially generated columns and can provide some insight into better starting bounds. Keeping in mind that the starting approximation already includes all singleton set variables (which explains the disproportionately large second column in the plots), both figures seem to portray a bimodal distribution. The modes are roughly one-sixth and two-thirds the number of items (4 and 13 for 20 items, 5 and 20 for 30 items), although separation between modes is more pronounced for the 30-item instances. Such behavior suggests that including more variables of such cardinalities would make for a better starting bound. Intuitively, given the information provided by the variables at these modes, one can "fill in the gaps" and approximate the incremental value to states corresponding to the intermediate set sizes; this can be a more efficient method than beginning with variable set sizes that are uniformly or normally distributed, for example.

Algorithm 1 provides for a systematic way to further reduce the gap arising from otherwise efficiently solvable bounds and policies, and it can reduce the gap significantly depending on where the largest areas of improvement lie. In essence, our algorithm performs well in the situations where we need it to the most; we observe the best progress when the initial bounds and/or heuristic policies perform the most poorly. It remains a valid alternative to the DP formulation for larger instances that still see noticeable gaps from not only a time perspective, but from a space perspective as well; our algorithm typically never required more than 1.5 GB of memory throughout its run, whereas the DP would require upwards of 15 GB memory for even the smaller 20-item instances.

6 Conclusion

We have studied a dynamic version of the knapsack problem with stochastic item sizes. We developed a finitely terminating exact algorithm that solves the dynamic knapsack problem within numerical tolerance, under the assumption of integer item sizes and capacity. The algorithm incorporates a combination of column and constraint generation to iteratively improve a value function approximation based on a reformulation of the original dynamic program. We provided theoretical results that solve the zero-capacity case in full from both the bound and policy sides, providing insight towards the general case, and we examined the hardness of subproblems encountered in the

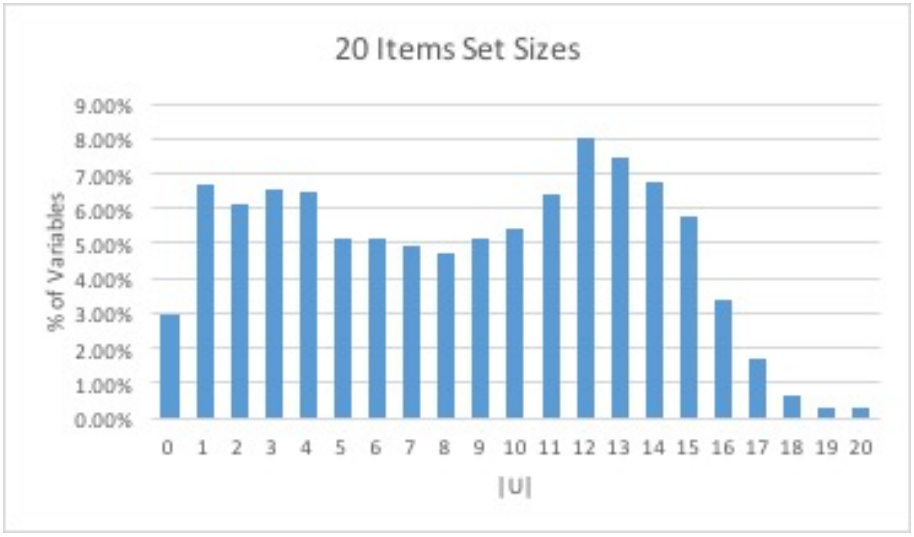

Figure 4: 20 Items: $w_U(\sigma)$ Set Size Frequencies

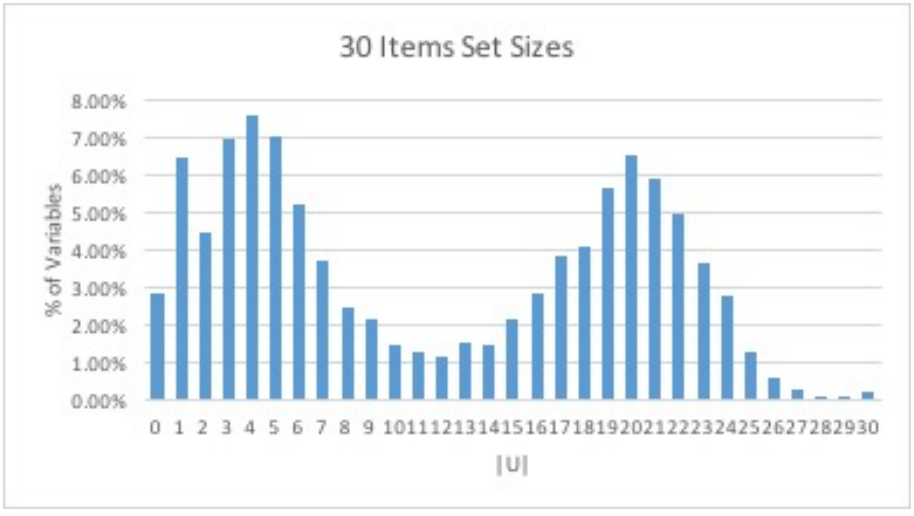

Figure 5: 30 Items: $w_U(\sigma)$ Set Size Frequencies

more general capacitated case. An extensive computational study points to the types of instances that see the greatest relative improvement. In particular, the algorithm significantly closes the gap from the policy side when natural heuristic greedy policies are lacking, while we also observe a steady gap closure from the bound side. The bounds derived by the algorithm also perform particularly well when the initial gap is relatively large.

Our results motivate additional questions. The exact algorithm provides theoretical guarantees from the bound side, while the policy side could be investigated further; moreover, more complex algorithm heuristics may improve computational performance. Other potential future work includes narrowing down the types of instances where the algorithm works best (such as examining different distributions), fine-tuning our heuristics (such as smarter or more complex column generation) for faster progress, and improving certain theoretical guarantees (such as establishing whether the pricing problem is NP-complete). As we do assume integer support in our analysis, it is also of interest to explore how we can develop an algorithm for continuous distributions, akin to the exact algorithm presented in [2] for the generalized joint replenishment problem, which has continuous state and action spaces. Finally, in an even more general sense, the knapsack problem is fundamental to the development of linear and integer programming. In a similar vein, it would be of interest to consider whether our methods — including value function approximations and a systematic algorithm — can be applied to other combinatorial optimization problems under uncertainty, e.g., the multi-row knapsack and other more general packing problems.

Acknowledgements

D. Blado's work was partially supported by an NSF Graduate Research Fellowship Program under Grant No. DGE-1650044. Both authors' work was partially supported by the National Science Foundation via grant CMMI-1552479.

References

[1] D. Adelman, *Price-Directed Replenishment of Subsets: Methodology and its Application to Inventory Routing*, Manufacturing and Service Operations Management **5** (2003), 348–371.

[2] D. Adelman and D. Klabjan, *Computing Near-Optimal Policies in Generalized Joint Replenishment*, INFORMS Journal on Computing **24** (2011), 148–164.

[3] S.R. Balseiro and D.B. Brown, *Approximations to stochastic dynamic programs via information relaxation duality*, Operations Research (2018), Forthcoming.

[4] A. Bhalgat, A. Goel, and S. Khanna, *Improved Approximation Results for Stochastic Knapsack Problems*, Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2011, pp. 1647–1665.

[5] D. Blado, W. Hu, and A. Toriello, *Semi-Infinite Relaxations for the Dynamic Knapsack Problem with Stochastic Item Sizes*, SIAM Journal on Optimization **26** (2016), 1625–1648.

[6] D. Blado and A. Toriello, *Relaxation Analysis for the Dynamic Knapsack Problem with Stochastic Item Sizes*, SIAM Journal on Optimization **29** (2019), 1–30.

[7] R.L. Carraway, R.L. Schmidt, and L.R. Weatherford, *An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns*, Naval Research Logistics **40** (1993), 161–173.

[8] D.P. de Farias and B. van Roy, *The Linear Programming Approach to Approximate Dynamic Programming*, Operations Research **51** (2003), 850–865.

[9] B.C. Dean, M.X. Goemans, and J. Vondrák, *Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity*, Proceedings of the 45th Annual IEEE Symposium on the Foundations of Computer Science, IEEE, 2004, pp. 208–217.

[10] ———, *Adaptivity and Approximation for Stochastic Packing Problems*, Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2005, pp. 395–404.

[11] ———, *Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity*, Mathematics of Operations Research **33** (2008), 945–964.

[12] C. Derman, G.J. Lieberman, and S.M. Ross, *A Renewal Decision Problem*, Management Science **24** (1978), 554–561.

[13] A. Goel and P. Indyk, *Stochastic load balancing and related problems*, Proceedings of the 40th Annual IEEE Symposium on the Foundations of Computer Science, IEEE, 1999, pp. 579–586.

[14] V. Goyal and R. Ravi, *A PTAS for Chance-Constrained Knapsack Problem with Random Item Sizes*, Operations Research Letters **38** (2010), 161–164.

[15] A. Gupta, R. Krishnaswamy, M. Molinaro, and R. Ravi, *Approximation Algorithms for Correlated Knapsacks and Non-Martingale Bandits*, Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, IEEE, 2011, pp. 827–836.

[16] M. Henig, *Risk criteria in a stochastic knapsack problem*, Operations Research **38** (1990), 820–825.

[17] T. Ilhan, S.M.R. Iravani, and M.S. Daskin, *The Adaptive Knapsack Problem with Stochastic Rewards*, Operations Research **59** (2011), 242–248.

[18] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer-Verlag, Berlin, 2004.

[19] J. Kleinberg, Y. Rabani, and É. Tardos, *Allocating Bandwidth for Bursty Connections*, Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, 1997, pp. 664–673.

[20] ———, *Allocating Bandwidth for Bursty Connections*, SIAM Journal on Computing **30** (2000), 191–217.

[21] A. Kleywegt and J.D. Papastavrou, *The Dynamic and Stochastic Knapsack Problem*, Operations Research **46** (1998), 17–35.

[22] ———, *The Dynamic and Stochastic Knapsack Problem with Random Sized Items*, Operations Research **49** (2001), 26–41.

[23] W. Ma, *Improvements and Generalizations of Stochastic Knapsack and Multi-Armed Bandit Algorithms*, Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2014, pp. 1154–1163.

[24] ———, *Improvements and Generalizations of Stochastic Knapsack and Markovian Bandits Approximation Algorithms*, Mathematics of Operations Research (2018), Forthcoming.

[25] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Ltd., Chichester, England, 1990.

[26] Y. Merzifonluoğlu, J. Geunes, and H.E. Romeijn, *The static stochastic knapsack problem with normally distributed item sizes*, Mathematical Programming **134** (2012), 459–489.

[27] H. Morita, H. Ishii, and T. Nishida, *Stochastic linear knapsack programming problem and its applications to a portfolio selection problem*, European Journal of Operational Research **40** (1989), 329–336.

[28] J.D. Papastavrou, S. Rajagopalan, and A. Kleywegt, *The Dynamic and Stochastic Knapsack Problem with Deadlines*, Management Science **42** (1996), 1706–1718.

[29] P.J. Schweitzer and A. Seidmann, *Generalized Polynomial Approximations in Markovian Decision Processes*, Journal of Mathematical Analysis and Applications **110** (1985), 568–582.

[30] W. Shi, L. Zhang, C. Wu, Z. Li, and F.C.M. Lau, *An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing*, IEEE/ACM Transactions on Networking **24** (2016), 2060 – 2073.

[31] M. Sniedovich, *Preference order stochastic knapsack problems: methodological issues*, Journal of the Operational Research Society **31** (1980), 1025–1032.

[32] E. Steinberg and M.S. Parks, *A preference order dynamic program for a knapsack problem with stochastic rewards*, Journal of the Operational Research Society **30** (1979), 141–147.

[33] M.A. Trick and S.E. Zin, *Spline Approximations to Value Functions: A Linear Programming Approach*, Macroeconomic Dynamics **1** (1997), 255–277.

Appendix

Proof of Proposition 3.3

Proof. For simplicity in notation, let us denote $v_N^*(0)$ as v_N and $w_N^*(0)$ as w_N. We proceed by induction on the size of N, with the trivial base case that $w_{\{i\}} = q_i c_i$. We now observe

$$\begin{aligned} w_N &= v_{\{1,\dots,n\}} - \sum_{\substack{U \subsetneq N \\ n \notin U}} w_U - \sum_{\substack{U \subsetneq N \\ n \in U}} w_U = v_{\{1,\dots,n\}} - v_{\{1,\dots,n-1\}} - \sum_{\substack{U \subsetneq N \\ n \in U}} w_U \\ &= \Big(v_{\{1,\dots,n-1\}} + q_n c_n \prod_{i \le n-1} q_i \Big) - v_{\{1,\dots,n-1\}} - \sum_{\substack{U \subsetneq N \\ n \in U}} w_U \\ &= q_n c_n r_{N\setminus n} - \sum_{\substack{U \subsetneq N \\ n \in U}} w_U = q_n c_n r_{N\setminus n} - \sum_{\substack{U \subsetneq N \\ n \in U}} \Big[(-1)^{|U|+1} q_n c_n \prod_{i \in U \setminus n} p_i \Big] \\ &= q_n c_n r_{N\setminus n} - q_n c_n \Big(\sum_{U \subsetneq N\setminus n} \Big[(-1)^{|U|+2} \prod_{i \in U} p_i \Big] \Big), \end{aligned} \tag{16}$$

where $r_U := \prod_{i \in U} q_i$, and the second to last equality follows from the induction hypothesis. Considering the following identity,

$$\prod_{i \in N} p_i = \sum_{U \subseteq N} (-1)^{|U|} r_U,$$

we can thus simplify (16) above to yield

$$\begin{aligned} w_N &= q_n c_n r_{N\setminus n} - q_n c_n \sum_{U \subsetneq N\setminus n} (-1)^{|U|} \sum_{V \subseteq U} (-1)^{|V|} r_V = q_n c_n \Big[r_{N\setminus n} - \sum_{U \subsetneq N\setminus n} \sum_{V \subseteq U} (-1)^{|U|+|V|} r_V \Big] \\ &= q_n c_n \Big[r_{N\setminus n} + \sum_{U \subset N\setminus n} \sum_{V \subseteq U} (-1)^{|U|+|V|+1} r_V \Big]. \end{aligned} \tag{17}$$

Examining the double sum in (17), for a set X, r_X appears once for each (strict) subset of $(N\setminus n)\setminus X$. Thus we have

$$\sum_{U \subsetneq N\setminus n} \sum_{V \subseteq U} (-1)^{|U|+|V|+1} r_V = \sum_{X \subsetneq N\setminus n} r_X \sum_{Y \subsetneq (N\setminus n)\setminus X} (-1)^{|Y|+1}, \tag{18}$$

where the exponent for -1 is taken from the substitution of $V = X$ and $U = X \cup Y$.

On the other hand, the right hand side of (9) can be rewritten as

$$\begin{aligned} q_n c_n \, (-1)^{|N|+1} \prod_{i<n} p_i &= q_n c_n \, (-1)^{|N|+1} \sum_{X \subseteq N\setminus n} (-1)^{|X|} \, r_X = q_n c_n \sum_{X \subseteq N\setminus n} (-1)^{|N|+|X|+1} \, r_X \\ &= q_n c_n \Big[r_{N\setminus n} + \sum_{X \subset N\setminus n} (-1)^{|N|+|X|+1} \, r_X \Big] \end{aligned} \tag{19}$$

Comparing the double sum in (17) with the sum in (19), identity (18) implies that it thus suffices to show

$$S_{N,X} := \sum_{Y \subset (N\setminus n)\setminus X} (-1)^{|Y|+1} = (-1)^{|N+|X|+1}.$$

But viewing the sum $S_{N,X}$ combinatorially, we can rewrite

$$S_{N,X} = \sum_{i=0}^{|N|-|X|-2} \binom{|N|-|X|-1}{i} (-1)^{i+1} = \sum_{i=0}^{|N|-|X|-1} \binom{|N|-|X|-1}{i} (-1)^{i+1} - (-1)^{|N|-|X|}$$

$$= -\sum_{i=0}^{k} \binom{k}{i} (-1)^i + (-1)^k = (-1)^k,$$

where the third equality substitutes $k = |N| - |X| - 1$, and the last equality follows from the identity that the alternating sum of binomial coefficients is 0. Hence,

$$S_{N,X} = \begin{cases} -1 = (-1)^{|N|+|X|+1} & \text{if } |N| \text{ and } |X| \text{ have the same parity} \\ 1 = (-1)^{|N|+|X|+1} & \text{if } |N| \text{ and } |X| \text{ have different parity.} \end{cases}$$

The entire argument above is identical for any subset $M \subseteq N$. □

Computational Experiments - 10 item Instances Discussion

Table 3 provides summary results for the computational experiments described in Section 5. The success rate is defined as the percentage of instances that reached the optimality gap within the time limit, while run time is the average run time in hours for the successful instances. The incomplete remaining gap refers to the geometric mean of the remaining gap of any instance that did not reach the target optimality gap within the time limit. We observe from Table 3 that distributions D6 and D7, the two distributions that do not have support for 0, have the highest success rate. Other distributions seem to have a lower success rate as the variance of the distribution decreases; these results suggest that both including 0 in the support and smoothing the distribution can make closing the final gap of less than 0.5% difficult. We also observed that uncorrelated instances tend to take less time to complete than correlated instances, which makes sense as correlated item sizes and profits tend to make more difficult deterministic knapsack problems.

Additionally, recalling that the 10-item instances actually solve to (numerical) optimality, Figures 6 and 7 provide insight into which set cardinalities are more prevalent in the optimal solution. In particular, we observe a noticeable difference in the set size distributions between the Bernoulli instances (D1, D3, D4, D5) and non-Bernoulli instances (D2, D6, D7). The plot for Bernoulli instances is more skewed to the right and has a mode of two items, which further explains why the Quadratic relaxation (4) (which includes all singleton and pair item sets) was the best-performing approximation in earlier experiments for these instance types [6]. However, the plot for non-Bernoulli instances seems to become more normally distributed as the instance's variance decreases: D2 is centered around set sizes of 3 and 4, D6 centered around set sizes 4 and 5, and D7 centered around set size 5. This suggests that, at optimality, less extreme instances favor variables that correspond to larger item set sizes, because individual items have a relatively smaller impact, as opposed to in the more extreme Bernoulli instances.

Table 3: 10-Item Instances Summary

Dist	Initial Gap	Final Gap	RRG	Success Rate	Run Time (hr)	Incomplete Rem.Gap
D1	13.81%	0.49%	3.67%	40%	2.3	0.76%
D2	8.07%	0.21%	2.58%	40%	2.2	0.31%
D3	12.73%	0.48%	3.14%	60%	2.2	1.12%
D4	14.84%	0.36%	1.82%	70%	4.7	1.06%
D5	16.42%	0.30%	1.54%	80%	3.5	1.32%
D6	6.12%	0.02%	0.48%	100%	2.5	-
D7	5.05%	0.07%	1.69%	90%	2.5	0.60%

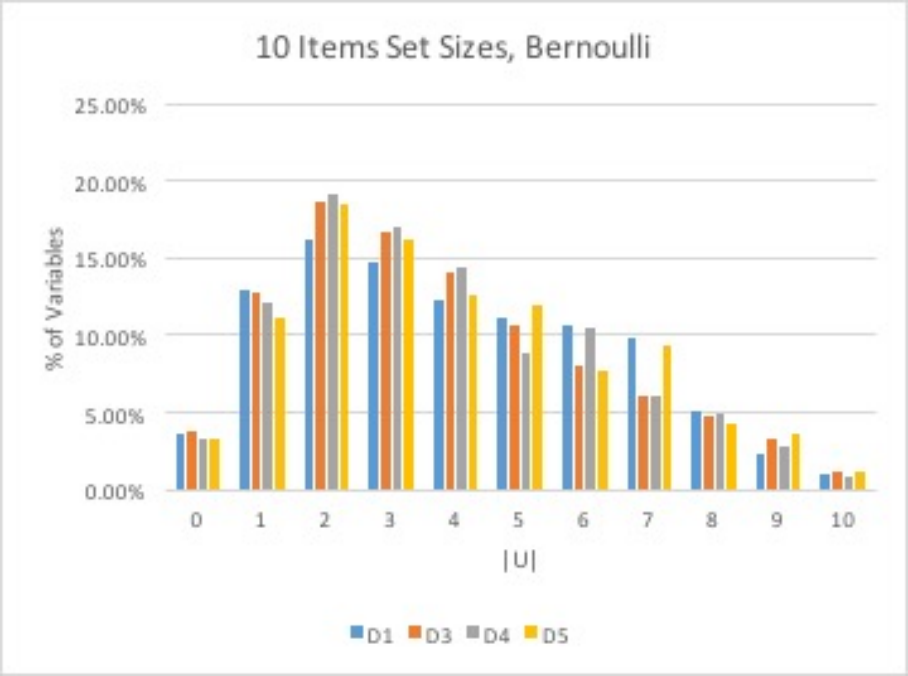

Figure 6: 10 Items: $w_U(\sigma)$ Set Size Frequencies, Bernoulli

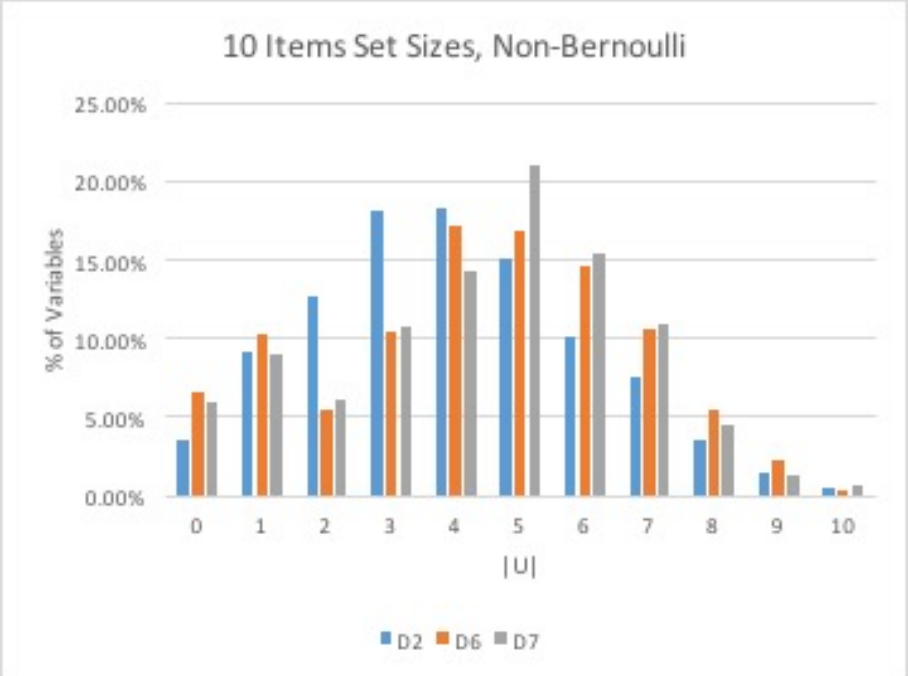

Figure 7: 10 Items: $w_U(\sigma)$ Set Size Frequencies, Non-Bernoulli

Additional Plots - 20 and 30 Item Instances

Figures 8 - 10 display various parameters of the 30 item instances against the relative remaining gap, while Figures 11 - 16 display various parameters of the 20 and 30 item instances against the relative gap closed per loop (RGPL), an alternative metric for the general algorithm's progress.

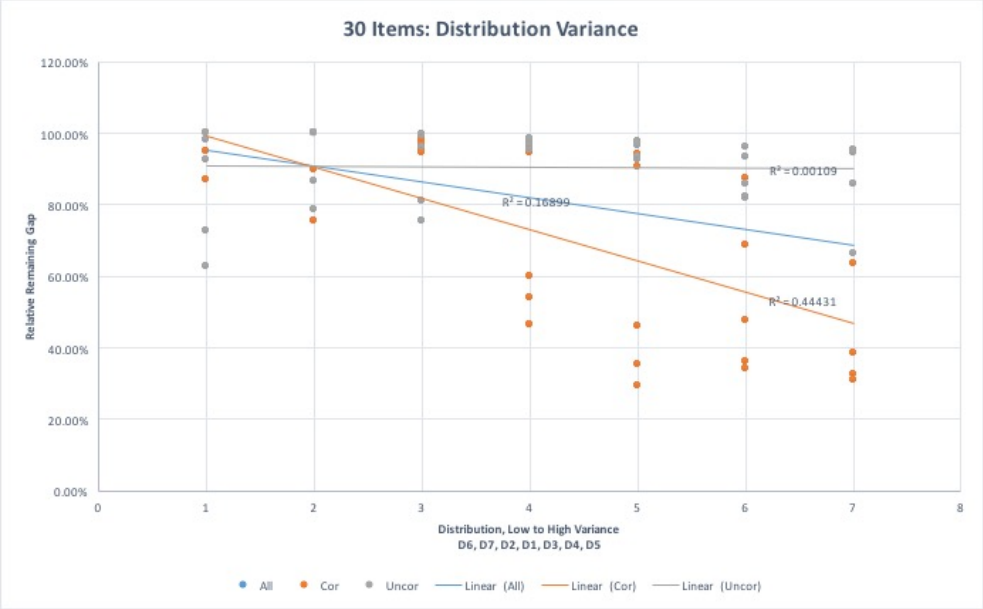

Figure 8: 30 Items - Distribution Variance vs. Relative Remaining Gap

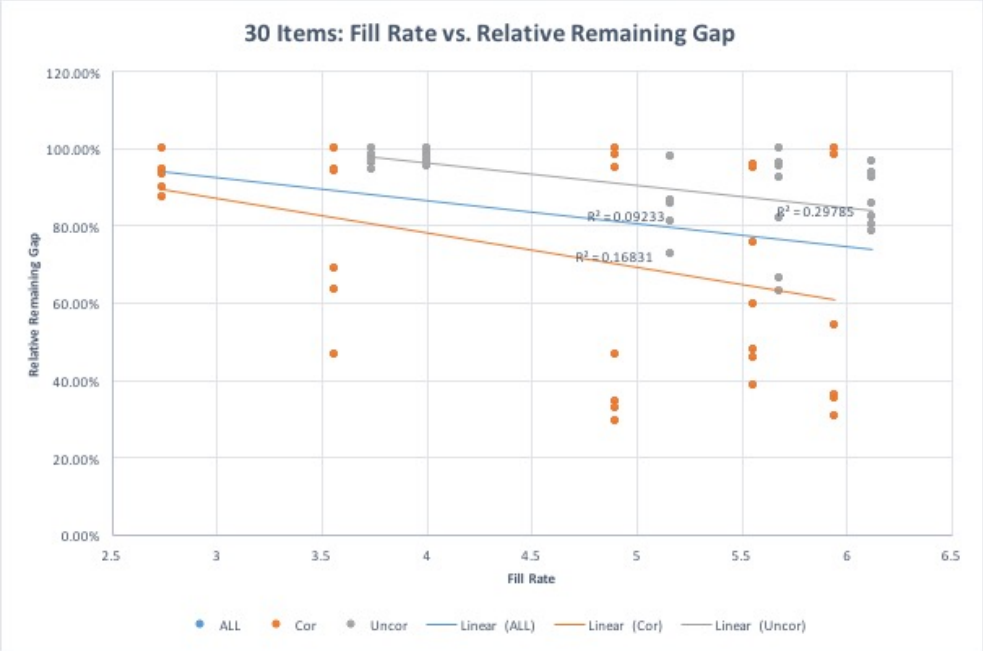

Figure 9: 30 Items - Fill Rate vs. Relative Remaining Gap

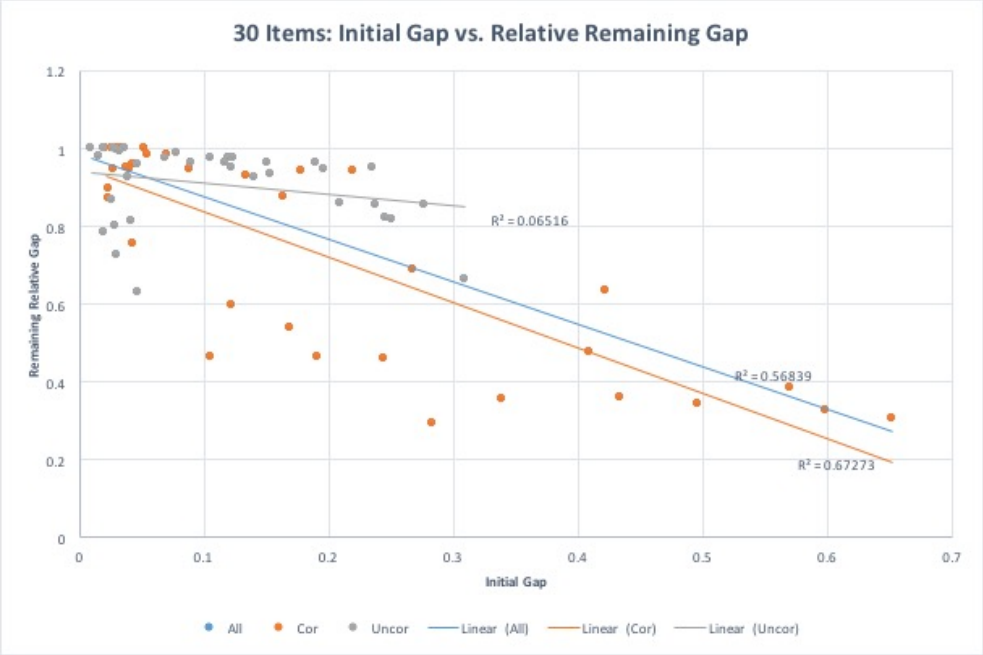

Figure 10: 30 Items - Initial Gap vs. Relative Remaining Gap

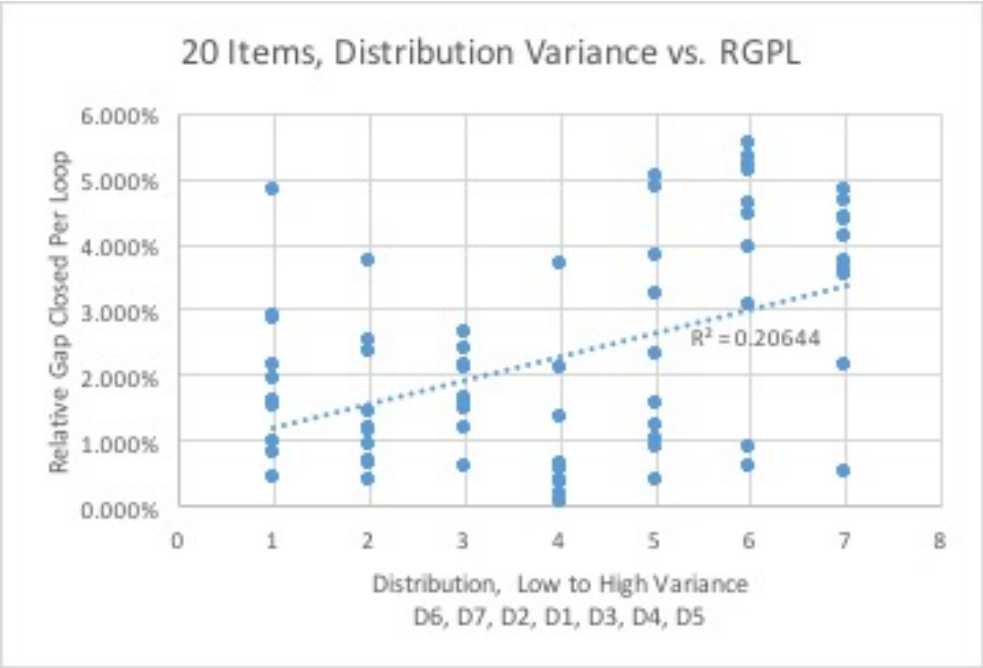

Figure 11: 20 Items - Distribution Variance vs. Relative Gap Closed Per Loop

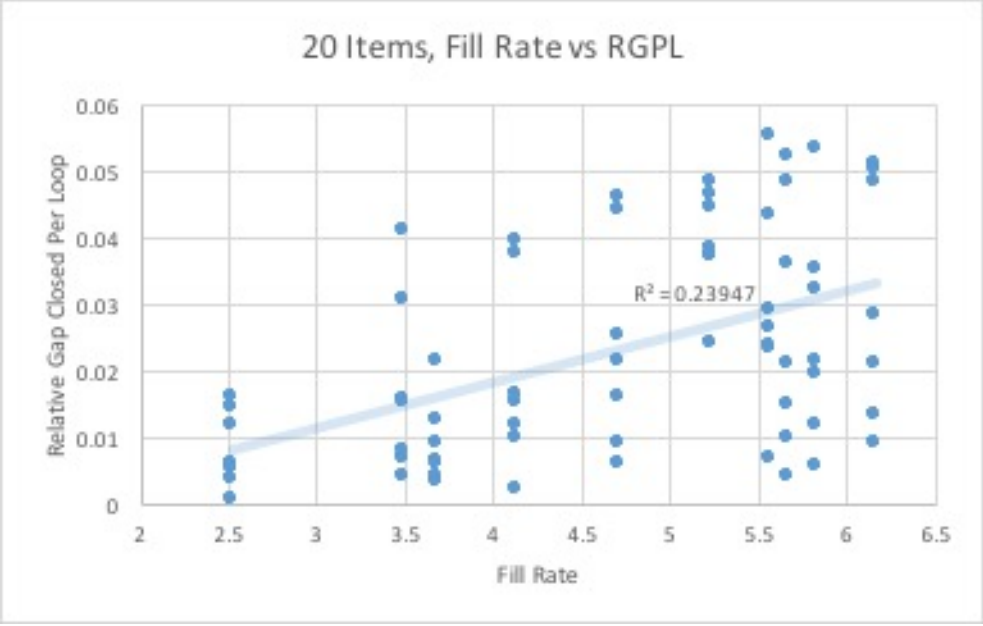

Figure 12: 20 Items - Fill Rate vs. Relative Gap Closed Per Loop

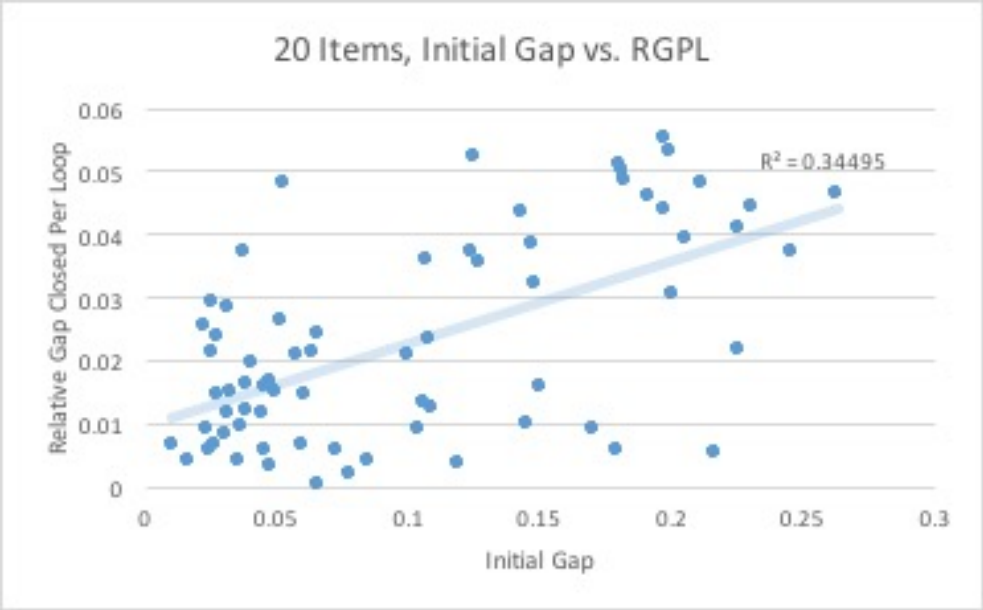

Figure 13: 20 Items - Initial Gap vs. Relative Gap Closed Per Loop

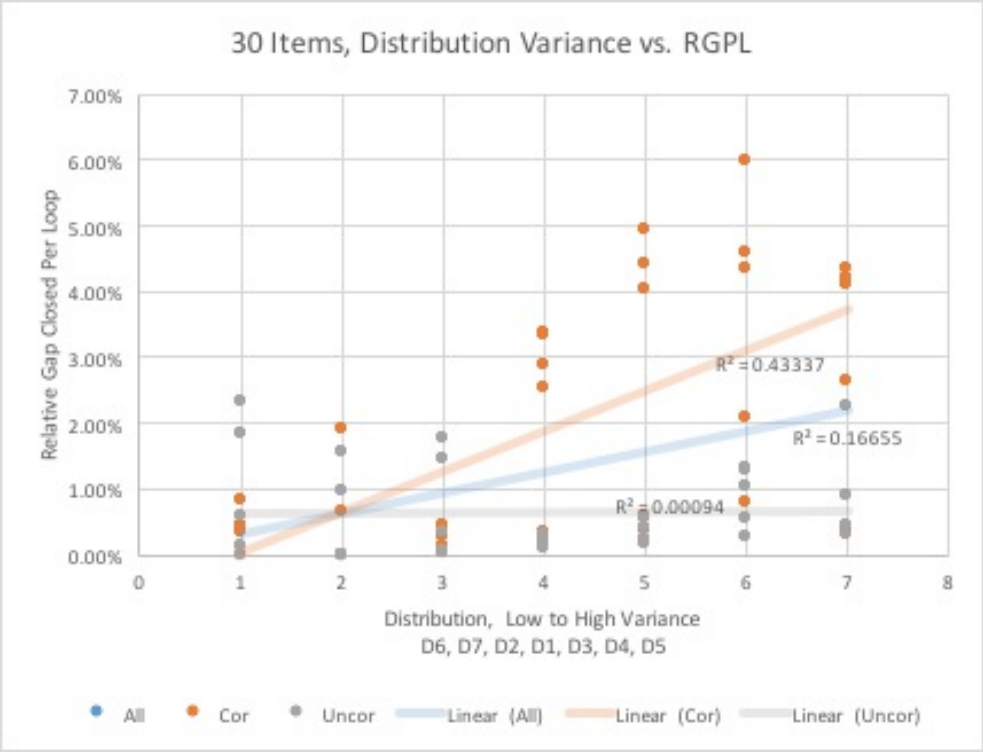

Figure 14: 30 Items - Distribution Variance vs. Relative Gap Closed Per Loop

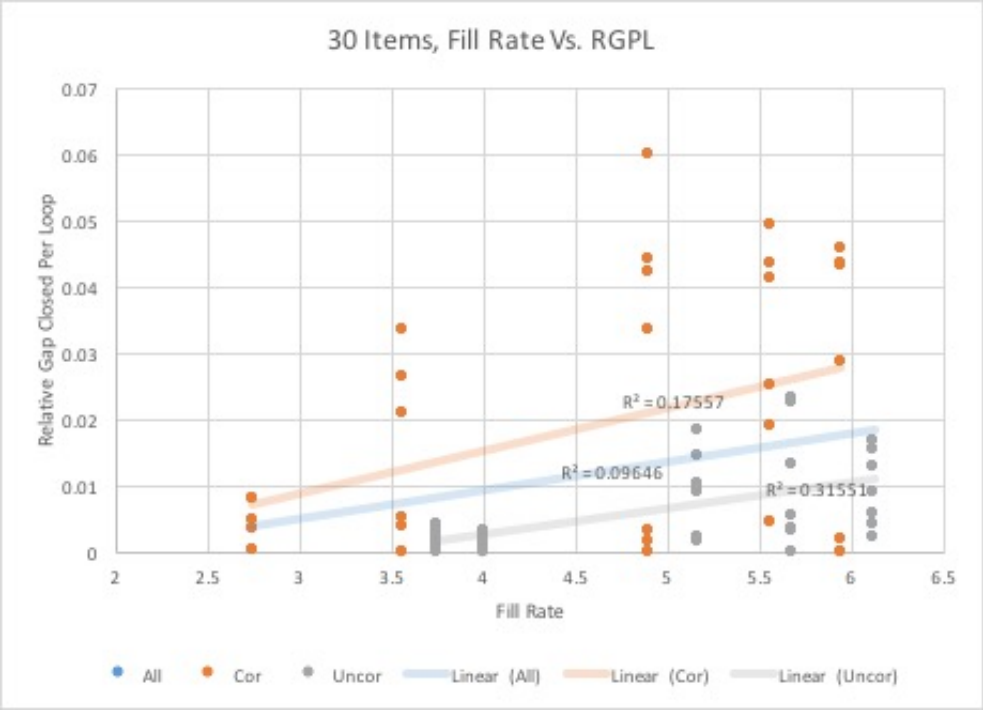

Figure 15: 30 Items - Fill Rate vs. Relative Gap Closed Per Loop

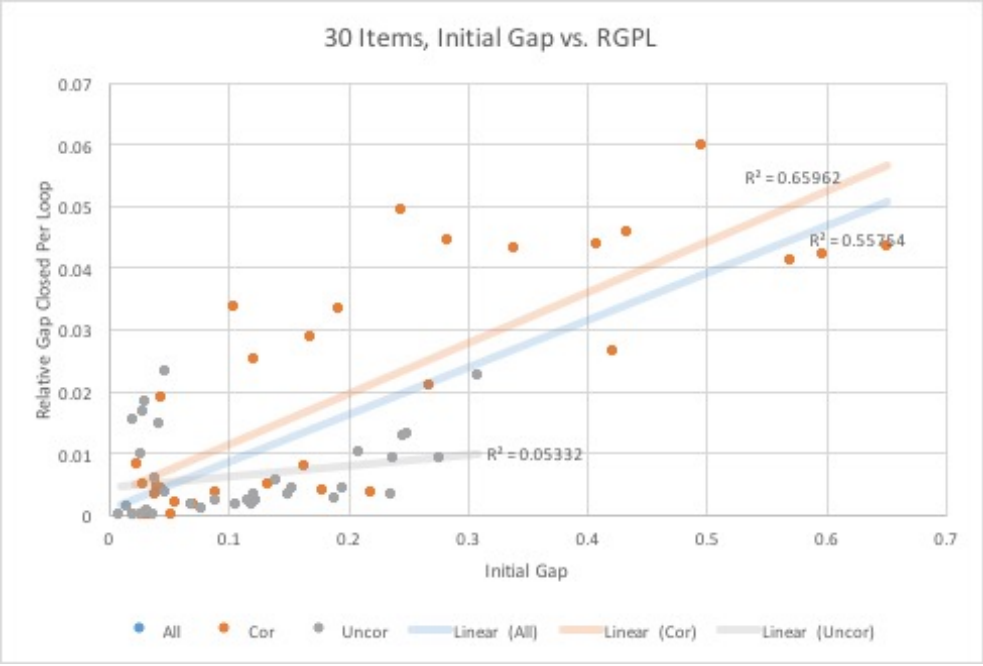

Figure 16: 30 Items - Initial Gap vs. Relative Gap Closed Per Loop

Raw Data

Tables 4 - 6 present the raw data used to calculate the summaries in Tables 1 - 3 of the general algorithm's performance in Section 5. They are separated by the number of items in each instance, recording results for 10, 20, and 30 items. For fairness in comparisons, the Avg. Inner Loops Metric for 20 and 30 item instances is only recorded for instances that completed 16 algorithm loops.

Table 4: General Algorithm Performance Data, 10 Items

Instance	Dist.	Initial Solution	Optimal Solution	Final Solution	Outer Loop Number	Time (hr)
cor2	D1	3723.00	3239.51	3319.10	14	-
	D2	3363.83	3102.76	3111.94	22	-
	D3	3938.67	3403.05	3422.53	13	-
	D4	4487.25	3615.94	3686.11	14	-
	D5	4637.60	3872.09	3953.59	15	-
	D6	3160.84	2980.63	2982.39	15	7.66
	D7	3180.66	3006.91	3009.57	15	4.90
cor4	D1	7082.50	6065.80	6076.35	18	-
	D2	5237.13	4754.38	4763.28	14	-
	D3	7311.11	6637.18	6641.55	16	2.19
	D4	9545.81	8278.62	8283.10	23	6.79
	D5	10432.00	8793.15	8800.82	19	2.62
	D6	4300.21	4082.59	4082.59	8	1.08
	D7	4328.47	4097.25	4097.25	10	2.96
cor8	D1	4780.38	4013.14	4038.08	15	-
	D2	3490.98	3227.07	3237.77	16	-
	D3	5458.00	4569.94	4599.94	16	-
	D4	5874.46	5143.37	5180.49	18	-
	D5	6454.19	5444.45	5445.61	23	4.02
	D6	2949.64	2848.44	2850.20	10	2.06
	D7	2994.84	2881.66	2899.09	12	-
cor11	D1	51.00	43.23	43.30	17	-
	D2	36.73	33.64	33.70	14	-
	D3	62.05	54.88	55.88	14	-
	D4	70.88	61.72	61.77	20	5.78
	D5	80.31	66.73	66.77	17	3.36
	D6	31.29	29.41	29.41	9	1.97
	D7	31.31	29.06	29.06	10	2.94
cor12	D1	55.38	54.00	54.04	16	0.62
	D2	54.42	52.31	52.32	10	4.62
	D3	92.67	86.84	86.87	22	2.19
	D4	126.55	112.56	112.61	27	9.62
	D5	156.98	132.59	133.30	24	-
	D6	43.85	40.38	40.38	4	0.15
	D7	44.45	43.50	43.50	4	0.13
uncor4	D1	11847.17	9768.71	9775.31	20	6.87
	D2	10051.80	8997.37	9002.33	8	2.02
	D3	11287.37	10075.41	10081.86	21	6.03
	D4	13272.00	11757.85	11762.92	25	6.60
	D5	14037.35	12022.72	12032.94	24	5.19
	D6	9294.49	8673.63	8673.63	7	0.87
	D7	9265.40	8751.16	8751.16	7	0.86
uncor5	D1	6324.50	5550.89	5603.74	14	-
	D2	5353.84	4945.85	4964.02	18	-
	D3	6895.33	5803.74	5885.59	15	-
	D4	7646.50	6114.44	6146.73	21	-
	D5	6835.15	5798.18	5801.64	21	8.00
	D6	4991.79	4753.80	4756.51	14	5.83
	D7	4983.91	4730.27	4732.16	14	5.61
uncor8	D1	6064.00	5271.50	5275.86	14	1.39
	D2	5506.90	5138.40	5143.21	9	1.51
	D3	5790.67	4964.20	4965.40	17	0.96
	D4	5580.70	5068.55	5073.16	18	1.09
	D5	5702.60	5222.68	5222.68	15	0.77
	D6	5286.64	5048.81	5048.81	8	1.77
	D7	5286.61	5060.61	5060.61	8	1.33
uncor11	D1	119.50	117.10	117.19	12	0.32
	D2	125.91	118.56	118.66	7	0.72
	D3	149.81	141.35	141.44	13	0.50
	D4	172.16	158.61	158.61	13	0.42
	D5	193.02	171.26	171.26	19	1.21
	D6	113.00	104.25	104.25	5	0.36
	D7	106.96	103.00	103.01	5	0.36
uncor12	D1	102.50	88.20	88.42	21	-
	D2	87.43	81.24	81.64	13	-
	D3	91.00	82.99	83.04	15	1.21
	D4	101.94	91.00	91.08	29	2.74
	D5	110.97	98.64	98.72	27	3.12
	D6	82.48	77.75	77.79	10	3.64
	D7	82.30	77.78	77.78	11	2.95

Table 5: General Algorithm Performance Data, 20 Items

Instance	Dist.	Initial Solution	Optimal Solution	Final Solution	Outer Loop Number	Avg. Inner Loops
cor2	D1	8215.50	7704.78	8211.96	16	5.19
	D2	7657.20	7329.29	7595.42	16	29.19
	D3	8768.33	7835.44	8722.72	13	-
	D4	9412.00	7980.20	9311.62	12	-
	D5	10061.76	8268.22	9977.12	12	-
	D6	7418.50	7145.35	7359.76	13	-
	D7	7418.50	7216.75	7380.73	13	-
cor4	D1	11434.58	10785.39	11366.11	16	21.19
	D2	9244.13	8790.56	9053.40	16	19.75
	D3	14036.33	12668.45	13526.53	16	25.625
	D4	16348.50	13647.61	13968.59	16	20.06
	D5	17548.40	15349.62	16018.84	16	14.06
	D6	8247.44	8040.47	8156.91	15	-
	D7	8267.12	8044.54	8188.33	15	-
cor8	D1	7970.17	7604.99	7950.86	16	6.25
	D2	6626.10	6465.55	6613.77	13	-
	D3	9467.67	8537.54	9282.55	16	32.88
	D4	11085.75	9468.87	10911.40	12	-
	D5	12538.87	10233.05	11744.07	16	29.56
	D6	6151.53	6052.42	6146.51	12	-
	D7	6149.83	6087.90	6144.60	13	-
cor11	D1	85.58	81.82	85.23	16	14.44
	D2	69.66	66.62	68.88	16	25.38
	D3	105.22	95.34	103.99	14	-
	D4	122.75	102.98	108.24	16	24.63
	D5	138.80	115.87	122.69	16	20.25
	D6	61.96	60.41	61.43	16	-
	D7	62.08	60.73	61.65	13	-
cor12	D1	142.17	132.53	141.31	16	23.94
	D2	113.41	106.60	111.08	16	23.63
	D3	179.50	156.37	167.61	16	22.5
	D4	209.00	174.25	179.37	16	20.44
	D5	212.00	188.03	198.43	16	12.31
	D6	99.67	95.79	98.45	16	22.87
	D7	99.67	95.99	98.96	16	22.44
uncor4	D1	17963.50	15985.06	16790.60	16	24.56
	D2	16218.73	15209.27	15831.79	16	17.06
	D3	19616.17	17090.38	18552.01	11	-
	D4	22192.52	18034.99	19241.36	16	18.56
	D5	23960.34	18967.29	20016.30	17	14.71
	D6	15498.38	14725.92	14901.89	16	14.44
	D7	15540.18	14978.42	15205.11	16	29.06
uncor5	D1	10269.29	9523.34	10245.69	16	32.25
	D2	9263.30	8841.85	9152.33	16	23.44
	D3	11529.64	10066.58	11339.29	13	-
	D4	12793.16	10609.44	11329.72	17	16.76
	D5	13933.00	11187.66	13933.00	17	20.47
	D6	8785.79	8508.18	8732.77	12	-
	D7	8785.79	8518.29	8739.33	15	-
uncor8	D1	11863.00	10938.15	11803.01	16	14.63
	D2	10965.69	10449.73	10841.14	16	23.31
	D3	12784.80	11110.73	12337.19	17	26.82
	D4	13741.64	10785.14	12617.26	16	31.44
	D5	14302.31	11663.51	12575.21	16	24.13
	D6	10548.00	10236.26	10512.67	14	-
	D7	10548.00	10277.66	10524.56	13	-
uncor11	D1	234.10	212.81	226.99	16	29.50
	D2	208.31	196.32	205.50	16	21.13
	D3	264.71	223.76	232.97	16	23.88
	D4	255.50	226.97	233.21	15	-
	D5	265.80	240.16	251.09	16	13.44
	D6	195.60	188.69	194.60	15	-
	D7	195.60	188.91	195.17	16	20.50
uncor12	D1	140.88	127.39	137.98	16	25.94
	D2	126.42	119.53	124.12	16	22.69
	D3	157.67	133.50	140.62	14	-
	D4	162.75	137.88	142.47	16	17.75
	D5	178.80	147.61	154.71	16	23.13
	D6	120.63	116.96	118.96	16	17.25
	D7	120.63	117.80	120.29	13	-

Table 6: General Algorithm Performance Data, 30 Items

Instance	Dist.	Initial Solution	Initial Policy	Final Solution	Final Policy	Outer Loop Number	Avg. Inner Loops
cor2	D1	11548.00	10608.19	11496.75	10608.19	16	5.44
	D2	10781.65	10460.06	10780.70	10460.06	11	-
	D3	12271.00	10821.49	12168.77	10821.49	15	-
	D4	13030.32	11194.17	12798.68	11194.17	16	9.87
	D5	13889.80	11387.92	13747.20	11387.92	16	9.06
	D6	10561.00	10320.98	10530.01	10320.98	16	17.40
	D7	10561.00	10317.01	10535.75	10317.01	13	22.75
cor4	D1	15557.00	13867.66	15546.03	14536.94	16	6.75
	D2	13301.63	12750.54	13278.03	12750.54	10	-
	D3	18364.25	14760.25	18251.80	16601.37	11	-
	D4	21442.94	15223.04	21242.73	18273.65	12	-
	D5	24914.60	15873.04	24549.34	21081.60	15	-
	D6	12304.55	11818.96	12279.25	11818.96	12	-
	D7	12332.46	11823.52	12330.79	11946.38	13	-
cor8	D1	11161.75	10101.19	11142.07	10649.72	16	7.07
	D2	9887.71	9618.24	9873.14	9618.24	11	-
	D3	12590.24	10681.37	12545.94	10750.26	16	11.50
	D4	14315.82	11291.80	14270.31	12190.66	15	-
	D5	16322.45	11475.97	16263.20	13196.30	14	-
	D6	9377.13	9186.32	9377.13	9186.32	16	12.93
	D7	9409.03	9163.83	9409.03	9163.83	15	-
cor11	D1	118.00	99.03	117.56	108.73	16	7.43
	D2	100.31	93.74	100.20	93.74	10	-
	D3	138.67	108.04	137.81	128.87	16	8.73
	D4	161.42	107.92	160.49	142.19	11	-
	D5	186.40	116.65	184.81	162.08	16	11.92
	D6	92.03	88.62	91.86	88.62	16	11.75
	D7	92.08	88.98	92.08	88.98	13	-
cor12	D1	192.17	164.43	191.88	176.89	16	4.13
	D2	162.31	153.84	162.16	153.84	10	-
	D3	229.56	171.46	228.11	207.56	15	-
	D4	271.06	189.09	268.53	239.00	14	-
	D5	315.87	191.34	312.04	273.81	16	24.31
	D6	148.72	141.28	148.72	141.28	16	-
	D7	148.72	144.28	148.72	144.28	16	9.69
uncor4	D1	26996.00	24412.35	26934.06	24412.35	16	11.69
	D2	25079.52	24063.74	24887.53	24063.74	13	-
	D3	28936.00	25764.59	28859.75	25764.59	11	-
	D4	31111.95	25727.13	30344.25	25727.13	14	-
	D5	33258.40	26871.57	32336.04	26871.57	16	41.75
	D6	24222.00	23496.94	24023.36	23496.94	15	-
	D7	24222.00	23591.97	24137.12	23591.97	14	-
uncor5	D1	15052.50	13961.66	15037.04	13961.66	16	8.25
	D2	14011.55	13569.32	14008.65	13569.32	11	-
	D3	16214.33	14478.62	16168.78	14478.62	16	8.56
	D4	17598.14	14791.72	17487.93	14791.72	16	7.56
	D5	19227.40	15559.64	19050.29	15559.64	16	7.69
	D6	13569.00	13299.03	13569.00	13299.03	16	10.08
	D7	13569.00	13210.18	13569.00	13210.18	16	8.47
uncor8	D1	16980.86	15874.49	16954.75	15874.49	16	7.00
	D2	16087.92	15618.98	16086.59	15618.98	11	-
	D3	17813.95	15948.73	17746.30	15948.73	16	8.19
	D4	18870.25	16404.04	18773.62	16404.04	12	-
	D5	20107.70	16816.02	19925.61	16816.02	13	-
	D6	15665.86	15418.97	15660.83	15418.97	16	9.56
	D7	15665.86	15526.89	15665.86	15526.89	16	10.79
uncor11	D1	312.82	278.75	311.66	279.25	16	8.50
	D2	285.95	273.25	285.79	273.60	12	-
	D3	341.88	299.86	338.65	299.86	14	-
	D4	375.50	300.32	361.71	300.32	14	-
	D5	407.40	311.34	375.07	311.34	15	-
	D6	273.91	261.64	273.91	266.20	16	12.07
	D7	273.91	264.26	273.91	264.26	16	9.80
uncor12	D1	205.25	188.36	204.63	188.36	16	20.67
	D2	189.85	184.47	188.77	184.47	12	-
	D3	221.58	192.00	219.65	192.00	16	20.13
	D4	241.25	193.68	232.82	193.68	14	-
	D5	263.40	206.38	255.17	206.38	16	27.00
	D6	183.10	176.18	182.58	176.18	13	-
	D7	183.10	179.43	182.99	180.11	14	-