Connecting Efficient Knapsacks – Experiments with the Equally-Weighted Bi-Criteria Knapsack Problem

Cameron Walker, Michael O’Sullivan
Department of Engineering Science
University of Auckland, New Zealand

Abstract

There are many applications of the classical knapsack problem in which the weight of the items being considered for the knapsack are identical, e.g., selecting successful applicants for grants, awarding scholarships to students, etc. Often there are multiple criteria for selecting items to be placed in the knapsack. This paper presents two new algorithms for finding efficient bi-criteria knapsacks when selecting from equally-weighted items.

The success of these two algorithms depend on unproven characteristics of the set of efficient knapsacks. If there is a path of adjacent swaps between any two efficient knapsack that remains within the set of efficient knapsacks, then the first algorithm will find the complete set of efficient knapsacks. The second algorithm requires an even stronger condition to guarantee finding the complete set of efficient knapsacks, namely that the path of adjacent swaps not only remains within the set of efficient knapsacks, but that the swaps always increase one objective while decreasing the other.

We compare the two algorithms with a bi-criteria shortest-path approach (that is known to find the set of efficient knapsacks) and show that the two algorithms produce the same solutions for a large test set of randomised problems. Our results suggest an interesting structure for the set of efficient knapsacks that may be exploited to find faster solution methods.
1 Introduction

The knapsack problem has been widely investigated for a single objective, and is considered to be one of the fundamental combinatorial optimization problems. In the multi-objective case the literature is relatively scarce, but various approaches to the problem do exist. These include branch and bound procedures (UT97; VTPU98), dynamic programming (EC96; KW00), conversion to shortest path (CCF+03), and meta-heuristics (BACK99; Han97; SKM99). For a more detailed review of the existing literature on the multiple criteria knapsack problem, and on multi-objective combinatorial optimization in general, we direct the interested reader to the survey paper by Ehrgott and Gandibleux (EG00).

In this paper we adopt a local search approach to the bi-criteria equal-weighted knapsack problem, which is a special case of the multiple criteria knapsack problem. This approach assumes the connectedness (in the graph-theoretical sense) of the set of efficient knapsacks. We present two algorithms that iteratively determine new potential efficient knapsacks by investigating knapsacks that are adjacent to some potential efficient knapsack in the currently identified set. Although this local search methodology has been successfully implemented in multiple criteria linear programming simplex-algorithms (Ise77), in the case of multiple criteria combinatorial optimization it is not always true that the set of efficient knapsacks is connected. For example, the shortest path, minimum spanning tree, and various matroid optimization problems are known to have a set of efficient knapsacks that is not connected in general (EK97). Despite this, algorithms (for the minimum spanning tree problem at least) implemented based on the connectedness assumption have yielded very good results, due to the apparent scarcity of disconnected efficient solution spaces (see (HR94; Lin95) for examples). It is not known whether the set of efficient knapsacks for the bi-criteria equal-weighted knapsack problem is connected in general, but every randomly generated example we have investigated thus far has yielded a connected efficient knapsack set.

This paper is organized as follows. In section 2 we give some background for the problem. We outline the Multiple Sweep and Single Sweep Algorithms for determining the complete set of efficient knapsacks to the bi-criteria equal-weighted knapsack problem in section 3. In section 4 we compare the solution sets of our algorithms and an efficient shortest-path algorithm (CCF+03), and give some performance results. We comment on our results in section 5 and discuss possible future work.

2 Background

The knapsack problem involves the selection from \( N \) items to put into a knapsack, with the goal of maximizing the total value of the knapsack’s contents. The knapsack has capacity \( W \) and the items’ weights are given in a vector \( w \). The “worth” of each item is given in a corresponding vector \( c \).
The knapsack problem may be formulated mathematically in the following way:

\[
\begin{align*}
\min c^\top x \\
w^\top x & \leq W \\
x & \text{ binary}
\end{align*}
\]

The decision variables \(x\) will be 1 for items included in the knapsack and 0 for items left out. This problem can be solved using known optimization techniques.

When dealing with more than one criterion, rather than looking for a single optimal knapsack, we instead consider all possible efficient knapsacks. A knapsack is said to be efficient if there exists no other knapsack that is better in terms of one criterion and at least as good in all the other criteria. The multiple criteria knapsack problem has the following formulation:

\[
\begin{align*}
\min c_1^\top x, \ldots, c_r^\top x \\
w^\top x & \leq W \\
x & \text{ binary}
\end{align*}
\]

where each item’s “worth” under criterion \(i\) is given in vector \(c_i\).

It is possible to introduce a definition of adjacency for knapsacks. We say that two knapsacks are adjacent if one knapsack can be obtained from the other by replacing one object inside the knapsack with one object not inside the knapsack. Restricting our attention to efficient knapsacks, if the induced graph is connected we can use local search techniques to identify all efficient knapsacks.

Of course if two knapsacks contain a different number of items, they cannot be adjacent. Thus, if we wish to use local search techniques to identify efficient knapsacks, we must restrict our class of problems to those where every knapsack contains the same number of items. If we restrict our attention further, to the bi-criteria case, this leads to the following question:

“With the definition of adjacency given previously, does the problem

\[
\begin{align*}
\min c_1^\top x, c_r^\top x \\
w^\top x & \leq W \\
x^\top e = K \\
x & \text{ binary}
\end{align*}
\]

have a connected set of efficient knapsacks in general?”

Clearly the answer is “no”. For example, if \(K = 2\) and \(W = 6\) we can consider four objects with rewards \((2, 0)\), \((2, 0)\), \((0, 1)\) and \((3, 2)\) respectively, and corresponding object weights \(3, 3, 2, \) and \(4\). The problem has only two efficient knapsacks: \((1, 1, 0, 0)\) and \((0, 0, 1, 1)\), which are not adjacent.
If we simplify the formulation to

\[
\begin{align*}
\min c_1^\top x, c_r^\top x \\
x^\top e &= K \\
x &\text{ binary}
\end{align*}
\]

then the question of connectedness for the set of efficient knapsacks becomes less clear. In this formulation each object has the same weight and the knapsack can hold any \( K \) objects at one time.

The Multiple Sweep Algorithm is guaranteed to find the set of efficient knapsacks if that set is connected. This guarantee will only hold for the Single Sweep Algorithm subject to a further condition. There must exist a stepwise path from an initial efficient knapsack to all other efficient knapsacks. Furthermore, this paths only traverses the set of efficient knapsacks. All our experiments thus far with these new algorithms have yielded the entire set of efficient knapsacks (confirmed by a shortest-path algorithm that does not require connectedness), suggesting that the set of efficient knapsacks is indeed connected.

3 The Sweep Algorithms

In this section we present our algorithms for identifying the complete set of efficient knapsacks to the equal-weighted bi-criteria knapsack problem – the Multiple Sweep Algorithm and the Single Sweep Algorithm.

3.1 The Multiple Sweep Algorithm

To begin the Multiple Sweep Algorithm we order the objects lexicographically, by objective 1 and then by objective 2. The first \( K \) items form an initial efficient knapsack (shown in figure 1).

This initial efficient knapsack has the maximum value in terms of objective 1. It also has the lowest value in terms of objective 2 of all the efficient knapsacks. This can be seen by noting that any knapsack with a lower objective 2 value is dominated by this initial knapsack.

We now construct a set of potential efficient knapsacks by “sweeping up” from our initial efficient knapsack. At each step of our sweep we look for the “nearest” (in terms of objective 1) adjacent knapsack to our current potential efficient knapsack, with the additional condition that our new knapsack has a lower objective 1 reward and a higher objective 2 reward.

Each such step is determined by considering stepwise object swaps, i.e., replacing an object inside the knapsack with an object outside the knapsack of lower objective
1 reward and higher objective 2 reward. The sweep is terminated when no suitable swaps remain. This gives an initial set of potential efficient knapsacks, ordered in terms of the objective 1 reward in one direction, and also in terms of the objective 2 reward but in the opposite direction. An illustration of this initial sweep is given in figure 2.

Starting from the knapsack in our list with the greatest objective 2 reward, we then proceed to try to identify the nearest (in terms of objective 2) adjacent knapsack that is not dominated by any of, and is distinct from all of, the knapsack in the current set. If we find such a knapsack we repeat the sweep procedure in the opposite direction to our initial sweep. Otherwise we move to the next knapsack in our original list and again search for a new adjacent potential efficient knapsack.

In this way we generate a second list of knapsacks that we merge with the first list, disregarding any dominated knapsacks and maintaining the lexicographical ordering.
By repeating this process we are able to identify new potential efficient knapsacks and discard any dominated knapsacks. If in two consecutive sweeps we generate no new potential efficient knapsacks we stop and our current list of knapsacks is the complete list of efficient knapsacks.

![Sweeping Down](image)

Figure 3: Sweeping Down

In fact, this algorithm does not depend on the connectedness of the set of efficient knapsacks, as only the potential efficient knapsacks that remain when the algorithm terminates are efficient. Furthermore, the set of efficient knapsacks produced by this algorithm are not guaranteed to be connected, as the merge step removes any dominated potential efficient knapsacks, which could disconnect the set. However, if the set of efficient knapsacks is connected this algorithm will find all efficient knapsacks.

### 3.2 The Single Sweep Algorithm

As with the Multiple Sweep Algorithm, we first order the items lexicographically by objective 1 and then objective 2. The first $K$ items form an efficient knapsack that is used to begin the algorithm. We construct the complete set of efficient knapsacks by sweeping from the initial efficient knapsack (see figure 1). At each step we look for the nearest efficient knapsack to the most recent efficient knapsack (in terms of objective 1). In the Single Sweep Algorithm each new efficient knapsack can be adjacent to any of the knapsacks on the current list. This new knapsack must have a higher objective 2 reward (and lower objective 1 reward) than the efficient knapsack determined in the previous iteration.

An illustration of how this algorithm works is given in figure 4. Note that the fourth efficient knapsack from the left in figure 4 is in fact adjacent to the second efficient knapsack rather than the third.

This single sweep continues until no new knapsack can be found. Not only does this algorithm require connectedness, but furthermore for each efficient knapsack $E$ there
must be a path in the efficient knapsack space from the initial efficient knapsack to \( E \) that travels from the right to the left only. Thus there can exist no knapsack spaces such as is shown in figure 5, as not all the efficient knapsacks will be identified for such problems.

The performance of this algorithm is improved by storing two additional pieces of information with each efficient knapsack. The first piece of information is the nearest “candidate” knapsack for each efficient knapsack. This is the closest potential efficient knapsack that is adjacent to the efficient knapsack in question. To be a candidate it is necessary to not be dominated by any of the efficient knapsacks in the current list. At each iteration of the algorithm the candidates are compared and whichever is nearest to the most recent efficient knapsack in the current list is deemed to be efficient. We then calculate its candidate and recalculate any candidates that are dominated by this new efficient knapsack. This process continues until there are no candidates remaining.
The second piece of information that we store is the “gap” used in the object swap for each candidate. Recall that the objects have been ordered lexicographically and that each candidate is obtained by performing a single object swap on the efficient knapsack. If we keep track of the difference in index between the object removed and its replacement (the gap) then, if a candidate becomes dominated, we can speed up the process of computing the new candidate by only considering object swaps that give a gap at least as big as the previous gap.

4 Experimental Results

In order to check the correctness of the Multiple Sweep Algorithm and the Single Sweep Algorithm we randomly generated twenty-five instances of nine different scenarios (small, medium, and large) and determined the complete set of efficient knapsacks using these two algorithms as well as a very efficient shortest path algorithm (CCF+03). We note here that the shortest path algorithm can solve for any set of positive weights, whereas our algorithms are specific to the equal-weighted bicriteria knapsack problem.

We coded all three algorithms in Matlab 6 and ran them on a computer with a 1.4GHz AMD Athlon chip. The nine scenarios involve 20 objects with knapsack sizes of 5, 7 and 10; 40 objects with knapsack sizes of 5, 12 and 17; and 60 objects with knapsack sizes of 5, 15 and 25. There is no point in allowing the knapsack to hold more than half the objects as we can transform such a problem into a knapsack problem where the decision is which objects to leave out.

For all of these tests the Multiple Sweep Algorithm and the Single Sweep Algorithm identified the same set of efficient knapsack as the shortest-path algorithm. This indicates that the structure of the set of efficient knapsacks (at least for our randomised problems) is such that our local search algorithms work correctly.

We also compared the performance of the Multiple Sweep Algorithm and the Single Sweep Algorithm with the shortest-path algorithm. The sample sizes are large enough to invoke the Central Limit Theorem so we present our results in the form of 95% confidence intervals that result from paired t-tests. The times given are in seconds and the confidence intervals represent the difference between the mean time for the first algorithm to solve the corresponding knapsack problems and the mean time for the second algorithm to solve the same knapsack problems. If a confidence interval contains 0 then there is no significant difference. If the interval is greater than 0 then the second algorithm has a lower mean solution time, if the interval is less than 0 then the first algorithm has a lower mean solution time. The results are given in table 1.

In comparing the three algorithms there is no overall winner. The shortest path algorithm is the fastest for the cases \((K, N) = (5,60)\) and \((25, 60)\). The Single Sweep Algorithm is the fastest for the cases \((5,20)\) and \((15,60)\). For the other five scenarios no algorithm was significantly better than both of the other algorithms.
<table>
<thead>
<tr>
<th>$K$</th>
<th>$N$</th>
<th>Paths vs Multiple</th>
<th>Path vs Single</th>
<th>Multiple vs Single</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>[0.11  0.48]</td>
<td>[0.40  0.66]</td>
<td>[0.15  0.32]</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>[-0.31  0.47]</td>
<td>[-0.55  0.31]</td>
<td>[-0.36 -0.04]</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>[-0.16  0.89]</td>
<td>[-1.96 -0.24]</td>
<td>[-1.88 -1.06]</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>[-1.82  0.05]</td>
<td>[-8.71 -4.27]</td>
<td>[-6.99 -4.24]</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>[-20.2 -0.12]</td>
<td>[-1.68  5.88]</td>
<td>[5.67 18.8]</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>[-41.2 -11.6]</td>
<td>[-5.83  6.13]</td>
<td>[17.1  36.0]</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>[-17.5 -9.5]</td>
<td>[-100 -63.6]</td>
<td>[-83  -53.8]</td>
</tr>
<tr>
<td>12</td>
<td>60</td>
<td>[-178 -51.8]</td>
<td>[1.7  17.3]</td>
<td>[66.0 169.5]</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>[-620 -281]</td>
<td>[-71.8 -5.23]</td>
<td>[272  552]</td>
</tr>
</tbody>
</table>

Table 1: Confidence intervals for algorithm comparisons

The mean times (in seconds) for each algorithm to solve each scenario are given in table 2.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$N$</th>
<th>Shortest Path</th>
<th>Multiple Sweep</th>
<th>Single Sweep</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>1.05032</td>
<td>0.75184</td>
<td>0.51544</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>1.67036</td>
<td>1.5918</td>
<td>1.79284</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2.71064</td>
<td>2.34244</td>
<td>3.81284</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>4.19372</td>
<td>5.07512</td>
<td>10.68568</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>17.70364</td>
<td>27.8512</td>
<td>15.60688</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>29.97984</td>
<td>56.3662</td>
<td>29.82836</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>9.48032</td>
<td>22.96588</td>
<td>91.36608</td>
</tr>
<tr>
<td>15</td>
<td>60</td>
<td>63.16136</td>
<td>178.1360</td>
<td>60.42704</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>143.2748</td>
<td>593.3659</td>
<td>181.8108</td>
</tr>
</tbody>
</table>

Table 2: Mean solution times for the algorithms

5 Summary and Future Work

The Multiple Sweep and Single Sweep Algorithms perform well on the test set of random examples. They always match the set of efficient knapsacks found by the shortest-path algorithm developed by Captivo et al (CCF+03). It seems probable, therefore, that the set of efficient knapsacks for the equally-weighted bi-criteria knapsack problem is always connected and contains stepwise paths from the initial efficient knapsack to all other efficient knapsacks, although this still needs to be proved theoretically.

Neither of our algorithms showed an improvement on the shortest-path algorithm. However, they do indicate structure in the set of efficient knapsacks that may allow for the development of faster methods. The Multiple Sweep Algorithm exploits the bi-criteria objective, so it may prove difficult to extend this approach to more criteria. However, the Single Sweep Algorithm does not use the objective dimension,
so we may look to extend this approach in the future to solve the equally-weighted multi-criteria knapsack problem.

References


