Project Title: KS-Solve: Local Search for the Knapsack Problem

Members: Janani Sriram

1. Problem Definition

The Knapsack Problem is an extensively researched combinatorial optimization problem. Suppose we have a knapsack that can carry a maximum of W kgs. We also have *n* items *<x_1,… , x_n>,* such that each item is associated with a value v_i, a class c_i and a weight w_i, *i*ϵ*<1,… n>.* We want to fit a subset of the n items into the knapsack without exceeding the maximum weight while maximizing total value and choosing at least one item from each class. Mathematically this is formulated as,

Maximise ${}^{n}\Sigma v_i\ x_i$ while satisfying constraints,

1. ${}^{n}\Sigma w_i\ x_i <= W$

2. There is one item from each Class c_j j ϵ<1,m>

Where x_i ϵ *<0,1>* and m is the number of classes.

This formulation restricts the number of copies of the item added to the knapsack to be 0 (Item not added) or 1(Item added) and only whole items can be added. Every constraint imposed on addition of items is a knapsack constraint. An m-dimensional Knapsack Problem has m constraints. For the purpose of this project we consider the class of Knapsack problems known as Multiple Choice 0-1 Knapsack problem.
The goal of this project is to implement the Multiple Choice variant of the optimal branch and bound algorithm (1) and heuristic local search using Beam Search and Genetic Algorithm.

2. Approach

1. Branch and Bound

This algorithm is based on branching recursively by adding items and pruning the search tree based on an upper bound which is the maximum potential value in the case of the knapsack problem. This algorithm is concisely is defined in (1) as,

> “The key idea of the BB algorithm is: if the lower bound for some tree node (set of candidates) A is greater than the upper bound for some other node B, then A may be safely discarded from the search. This step is called pruning, and is usually implemented by maintaining a global variable m (shared among all nodes of the tree) that records the minimum upper bound seen among all subregions examined so far. Any node whose lower bound is greater than m can be discarded.”

It has been shown that the branch and bound algorithm can be bounded by the objective function derived from the relaxed LP algorithm as described in (2) and greedily finds solutions by adding whole items to current values and weights or fractions thereof.

1. The branch-and-bound solver starts with an initial empty assignment and greedily searches the BFS tree for the next assignment.

2. Perform a breadth-first traversal on the remaining nodes from the root node.

3. At each node update the current weights and values. Calculate the bound. Update the global optimum or prune if necessary.

<u>State</u>: A state is a bit vector denoting a partial placement of items in knapsack. At each state the total value V, total weight W and the bound B for that subtree of added items should be maintained.

<u>Successor Function</u>: The successors of states are generated using a breadth first search strategy.

<u>Objective Function</u>:

For a state s at depth d of the search tree the objective function is,

f(s)= V+Value of items that can be fully placed + Fractional Value of item that cannot be fully placed.

$f(s)= V_d+_{(i=d+1)}\Sigma^j V_i+((C-W_j)/W_{j+1})*V_{j+1}$

where the successor nodes from levels d+1 to j denote items can be fully fit into the knapsack and the node at j+1 is fit into the knapsack whose capacity is C with a fractional value.

<u>Solution</u>: Sequence of items added up to the leaf node with optimal value when all nodes have been generated.

The general branch and bound algorithm for the single knapsack problem will be extended for the multiple choice case for this project as follows -

1. Use BFS to visit nodes greedily to maximize value within allowable capacity. Allow pruning only when the global optimum seen so far satisfies constraint 2 and is greater than the current heuristic value.
2. Enumerate optimal solutions for (1) to check whether there is an item from each class. If not penalize the actual value of the state by setting it to 0 (do not update global bound). This step reduces to a brute force search over constraint-1 optimized solutions to solve for constraint 2.

The complexity of the branch-and-bound algorithm is $O(2^n)$ even for the 1-D Knapsack problem and is very slow for large datasets. The rest of this project will investigate and compare 2 related heuristic local search techniques that are likely to converge faster for this problem.

Penalty Function

For each state modify value to include a penalty,

i) If no more items can be added and the number of classes remaining to be assigned is non zero then set value of state to be zero.

ii) If more items can be added
$f'(n)=f(n)-\alpha$(Total number of classes-Number of classes assigned), where α is a tuning parameter that balances between maximizing value and choosing values over different classes. With a large α the solution quickly finds assignments satisfying constraint and with smaller α the solution slowly picks items from different classes. When all classes are assigned penalty is zero.

2. Local Beam Search

The Beam Search algorithm starts with an initial set of k states and generates successors of all the states. From these successors the best k states are chosen for the next iteration. In this manner beam search proceeds by converging towards dominating regions of the search tree. (3). One of the extensively studied problems with beam search is its tendency to converge towards and get stuck in a part of the state space and many variants have been suggested. However in many cases the knapsack problem exhibits certain partial assignments that are intuitively desired to appear in the final solution. So with a good assignment of initial states we should be able to converge to an optimal solution. For the purpose of this project an initial 'good' set of partial assignments is generated by sorting all states using the value to weight ratio in Dantzig's greedy approximation algorithm (2) for the relaxed version of the knapsack problem. Then the solver picks the next item from each class with the maximum value to weight ratio to generate each one of k states. Finally in order to ensure completion of search the solver will perform a random restart of all states if it gets stuck.

3. Genetic Algorithms

GAs are a class of local search algorithms that are a variant of stochastic beam search that generate successor states by combining two parent states rather than one. The set of states that form the parents at each generation are collectively known as the population. The process of generating a new population from an initial population involves generating successor states for each parent state and mating them. As in the local beam search algorithm the initial set of states will be generated using the value to weight ratios and the fitness function used to evaluate a population is the penalized evaluation function f'(s) with a uniform crossover point. Such evolutionary metaheuristic algorithms have been reported to perform well in a family of related problems such as combinatorial auctions and more generalized knapsack problems and is expected to perform better than a local beam search due to reproduction of good patterns of assignment. Assign a small probability of mutation if there is significant change in population from one generation to another and high probability if most individuals are retained (random restart).

3. Data and Test Cases

A Knapsack Problem over n items is defined by the following n-tuples

1) Metadata – Knapsack Capacity C, Number of classes M
2) Item codes – Alphanumeric Values that uniquely name an item
3) Weights – Positive floating point values
4) Values – Positive integers
5) Class – One of M Class labels

Additional data structures are -

6) Node – Item to be added, Current Value, Current Weight, Items included in partial assignment, Bound/Objective function Value.
7) Assignment – Bit Vector whose i^{th} is set if item i is added to the knapsack

The following reports will be generated -

1. Test the branch and bound algorithm with small datasets of sizes 10-40 and analyse the performance for different values for maximum value among all items and number of classes. Compare with performance of the 2 local search algorithms and report their deviation from the brute force optimized total value.
2. Test the beam search solver with different values of k for randomly instantiated and greedily instantiated initial states. Repeat for datasets of sizes 50-1000.
3. Test the GA solver with different values of initial population size for randomly instantiated and greedily instantiated initial states. Repeat for datasets of sizes 50-1000.
4. Report the performance of GA solver for crossover points of 5 and 15 and 50% of the number of items (length of chromosome).
5. Extract the following features for different runs of the local search solvers – average number of states visited, median value of states and number of restarts.

4. Evaluation

The correctness and optimality of the described implementation of local search algorithms for the 2-dimensional Knapsack Problem will be investigated by this project. The results will provide an empirical case study with an objective function loosely based on an optimal objective function for a relaxed problem (5; 1). Correctness will be reported by comparison against brute force results on small datasets. Optimality will be reported by weighing the returned optimum against the optimum reported in a standard Multi Knapsack Problem dataset (6) and select instances of benchmark data described in (4). This performance is likely to be poor since the described implementation of the local search algorithms cannot be compared with a well refined benchmarked solution for an extensively researched problem such as the Knapsack problem. Hence desired properties such as rejection of infeasible solutions, dominating partial assignments etc. for the solvers will be reported by visualizing intermediate results for tailor-made test cases (4).

5. Timeline

March 2

- Input Parser
- Abstract Data Types
- Branch and Bound

March 5

- Local Beam Search

March 11

- Genetic Algorithm
- Problem Generator
- Feature Extraction from Solvers

March 13

- Analysis
- Graphs
- Lessons Learnt
- Report

References

1. wikipedia, Branch and Bound. [Online] http://en.wikipedia.org/wiki/Branch_and_bound.

2. *Discrete Variable Extremum Problems.* **Dantzig, G.B.** 1975.

3. **Russell, Stuart and Norvig, Peter.** *Artificial Intelligence: A Modern Approach.*

4. *David Pisinger;.* **problems?, Where are the hard knapsack.** s.l. : Computers & Operations Research, 2005, Vol. 32.

5. *The Accuracy of Search Heuristics: An Empirical Study on Knapsack Problems.* **Leventha, Daniel H.; Sellmann, Meinolf.** s.l. : Springer, LNCS, 2008.

6. Multiple Knapsack Problems. *CMU Artificial Intelligence Repository.* [Online] http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/test/sac/readme.txt.

7. *The multiple-choice knapsack problem.* **Sinha, P and Zoltners, AA.** No. 3, 1979, Vol. Vol. 27.