Login Name	**:**	*jxk19u*
Family Name	**:**	*Kremser*
Other Names	**:**	*Jiri*

G52AIM (Artificial Intelligence Methods)

Lecturers: Graham Kendall & Rong Qu

Academic Year 2009-2010

(Coursework I – The 0-1 Multidimensional Knapsack Problem)

A. A brief description of the 0-1 MKP

The classical knapsack problem is a mathematical abstraction to many real world problems, however often there is not only one constraint for knapsack. It was obvious to extend this model to more a general variant, in which many constraints on the knapsack can be set. Acronym 0-1 MKP stands for multidimensional (or multiconstraint) knapsack problem, where the number of copies of an item in the knapsack can be at most one.

Informally, an instance of this problem is some set of items with different prices and constraints and the goal is to maximize the profit by selecting some subset of these items while all the constraints are satisfied. When solving the *m*-dimensional 0-1 KP, the *m* constraint vectors and capacities must be provided with the problem instance. Each dimension represents a different constraint, for instance sum of all weights of items in the knapsack can not be greater than maximum weight the knapsack can carry or another constraint can be made on the volume of knapsack. The constraints can be also thought as resources with capacities, the constraint vector then represents a consumption of a particular resource.

The KP is well-studied problem, since it can model many useful problems such as general resource allocation problems, investment of a budget in particular or cargo loading. By using MKP instead of 1 dimensional KP, more factors from real world can be included in the model and then the result will be more confident for decision support in a company. Another applications are in cryptography, when solving Subset sum problem[1], which can be considered as the multiple choice KP.

Computational complexity of this combinatorial problem is in class NP-Complete, therefore many heuristics approaches are applied to optimize a solution, with large instances of this problem (number of items and dimension), however, we have to settle for sub-optimal solution. More about this problematic will be described in the section C.

B. A mathematical description of the 0-1 MKP

Formally 0-1 MKP can be defined as the following linear program.

(0-1 MKP)

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \qquad (1)$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i=1,\ldots,m \quad j=1,\ldots,n, \quad x_j \in \{0,1\} \qquad (2)$$

Objective function *(1)* denotes the sum of prices of all items placed in the knapsack, since p_j denotes the *j-th* price and x_j is one or zero[2] denoting if the *j-th* item is in knapsack or is not. The n is the number of all possible items. Linear inequality constraints *(2)* mean the capacity c_i of knapsack in the *i-th* dimension (*i-th* constraint) cannot be lower than sum of all items' weight placed (x_j) in knapsack. Weights are denoted as w_{ij}

1 When prices are equal to weights.

2 Differ from more general bounded and unbounded KP.

and can be considered as numeral abstraction of *i-th* constraint on *j-th* item. Since there are *m* (sometimes denoted as *d*) dimensions, there are *m* such constraints and inequality *(2)* is only schema of for all *m* knapsack constraints. It is allowed to that $w_{ij} = 0$ for some *i* and *j*, as long as $\forall\, j \in \{0,\ldots,n\}.\ \sum_{i=1}^{m} w_{ij} \geq 1$ holds. To avoid trivial problem instances and to guarantee that an item can be packed at all we assume [KPS04] $\forall i \in \{1,\ldots,n\} \forall\, j \in \{1,\ldots,m\}.\ w_{ij} \leq c_i \ \wedge\ \sum_{j=1}^{n} w_{ij} \geq c_i$. Knapsack constraints create a matrix *W* of the size $n \times m$. 0-1 MKP can be reformulated as a decision problem by setting a threshold *t* and then asking whether the $\sum_{j=1}^{n} p_j x_j > t$ while satisfying all knapsack constraints *(2)*. Nevertheless, the computational complexity remains in the same class. This is similar to Graph Coloring and Graph k-Coloring.

C. Optimisation methods for the 0-1 MKP

Algorithms for solving KP in general can be divided into two classes. Exact algorithms and approximative algorithms. The computational complexity of all non-trivial generalizations of KP are NP-Complete, in other words the large instances of this problem are practically unsolvable[1]. The scope of this paper could not cover all this problematic about P = NP problem, but in short. If P = NP holds, which is commonly believed to not be true, there will be an algorithm for solving KP in a polynomial time, since NP-Complete problems can be reduced to one each other in a polynomial time. It may be good challenge for quantum computation to implement algorithm[2] for solving KP in future. Exact algorithms guarantee the optimal solution, although they can be used only for small instances or in case the prices (values) and weights are not correlated [FR99]. This class contains for example stupid Generate-and-Test algorithm, Branch and Bound method for solving KP or algorithms using dynamic programming method. In general it is very difficult to predict the hardness of a particular NP-Complete problem in advance without solving it. Providing we do not insist on the exact solution or the instance of problem is very large and we need a solution in a polynomial time, we have no other option than to use some of the approximative algorithms and settle for some sub-optimal solution.

Approximative algorithms often use some heuristics which depend on particular problem and can be divided to local search algorithms and population based algorithms. I will describe some of approximative algorithms, which are widely used[3].

a) local search algorithms (with meta-heuristics)

Greedy Algorithm

Unbounded KP can be solved by simple greedy algorithm. By ordering items by efficiency (*p/w* ratio sometimes called as *pseudo-utility ratio*) and then adding as many items with best efficiency as possible. Once the adding next item would exceed the capacity of the sack, try the item with the second best ratio and so on. Although this

1 In reasonable amount of time.

2 Grover's algorithm would be a good candidate, but unfortunately NP $\subset$ BQP does not hold, nevertheless there is quadratic speed up in comparison to classical computation [FR99].

3 LP-relaxation and and Core Concepts (Problems) will be omitted.

approach can not be used for solving 01-MKP it should solve the Unbounded-MKP and also it is an inspiration for the other methods [KSV93]. The solution's worst quality (cost) is then bounded from below by O/2, where O stands for quality of the optimal solution.

VNS

This meta-heuristic was described in the lectures. For solving 0-1MKP Relaxation Guided VNS is used, which is based on VND. In short, the initial solution is made by greedy first-fit algorithm (using LP-relaxation). Neighbourhoods (functions) are not a-priori ordered, but the actual neighbourhood is dynamically chosen during the computation, since choosing the right neighbourhood before another can significantly improve the performance of the algorithm. J. Puchinger and G. R. Raidl proposed [PuRa] following neighbourhoods.

- *IRF(x^f, k)* – removes k items from feasible solution and then add any combination of items back as long as the solution remains feasible.
- *IAR(x^f, k)* – add k items to knapsack. If any constraint is violated, then some items must be removed.

Both this neighbourhood operator must repair the solution if the feasibility is violated. This reparation can be done by greedy algorithm. Another neighbourhood can be defined as a simple swap of a pair of items, from which one is already in knapsack and one is not.

Tabu search

We can use the some neighbourhood operator defined above and perform local search with tabu list with fixed size[1]. F. Dammeyer and S. Voss proposed the reverse elimination method [DV93].

b) population based algorithms

GA

Representation of solution is an arbitrary[2] bit string of size *n* (*n* is number of items). If there is one on *j-th* position, the *j-th* item is placed in sack.

Parent selection: Tournament or roulette wheel.

Crossover: Uniform.

Mutation: Negation of randomly selected bit with very small probability.

Repair: More information in [CB98]. I will go further in my final report since I want use a GA.

ACO

More information in [ACG00].

BCA

More information in [NVT08].

1 Represents the forgetting.

2 Hence, the solution may be infeasible and it should be repaired or we should prevent this situation by having safe crossover and mutation operators. Or we simply allow infeasible solutions and we will penalize them.

Resources

[KPD04] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems,* Springer, 2004.

[FR99] L. Fortnow, J. Rogers *Complexity limitations on Quantum computation*, Comput. Syst. Sci. (Boston, MA: Academic Press), 1999.

[Pis95] David Pisinger, *Algorithms for Knapsack Problems*, PhD thesis, 1995.

[KSV93] AHG Rinnooy Kan, L. Stougie, C. Vercellis, *A class of generalized greedy algorithms for the multi-knapsack problem*, Discrete applied mathematics, 1993.

[PuRa] Jakob Puchinger, Gűnther R. Raidl, *Relaxation Guided Variable Neighborhood Search* , Vienna University of Technology.

[DV93] F. Dammeyer, S. Voss, *Dynamic tabu list management using the reverse elimination method*, 1993.

[CB98] P. C. Chu, J. E. Beasley, *A Genetic Algorithm for the Multidimensional Knapsack Problem* , Imperial College, 1998.

[ACG00] I. Alaya, Ch. Solnon, K. Ghédira , *Ant Algorithm for the Multidimensional Knapsack Problem*, 2000.

[NVT08] P. N. Nhicolaievna, Le Van Thanh, *Bee Colony Algorithm for the Multidimensional Knapsack Problem,* International MultiConference of Engineers and Computer Scientists, 2008.