République Algerienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignment Supérieur et la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
Univesité Kasdi Merbah - Ouargla -
جامعة قاصدي مرباح ـ ورقلة ـ

Option: Informatique Fundamental

Knapsack Problem
multi-dimension and multi-choice

Prepared By:
Abderrahman RIAG

Supervisor:
Dr. Meriem KHELIFA

Promotion : 2021/2022

Contents

CHAPTER 1

General Introduction:

CHAPTER 2

OPTIMIZATION PROBLEM

1 Knapsack Problem:

The knapsack problem (KP) can be formally defined as follows: Given an instance of the knapsack problem with item set N, consisting of n items, $N := \{1, ..., n\}$ j with profit p_j and weight w_j, and the capacity value c, (Usually, all those values are taken from the positives integer numbers.) Then the objective is to select a subset of N such that the total profit of the selected items is maximized and the weights does not exceed c.
Alternatively, a knapsack can be formulated as a solution of the following linear integer programming formulation:

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

$$\text{subject} \sum_{j=1}^{n} w_j x_j \leq c, \tag{1–2}$$

$$x_j \in \{0, 1\}, j = 1, ..., n. \tag{1–3}$$

The two functions (1.1) and (1.2) define the objective (maximize the profit) and the constraint (the total weight does not exceed c) respectively.
Such:
• x_j is a binary value, which means $x_j = 1$ if we put the item j into the knapsack and $x_j = 0$ if we don't.
• j refers to the item, $j \leq n$.

2 Multi-Dimensional Knapsack Problem MDKP:

We given a knapsack with m-dimensions, let b_i be the capacity of the ith dimension, $i = 1, 2, ..., m$. There are n items, and let u_j be the number of copies of item $j, j = 1, 2, ..., n$, the jth item requires a_{ij} units of the ith dimension of the knapsack. The reward of including a single copy of item j in the knapsack is c_j. The problem can formulated as follows:

$$maxZ = \sum_{j=1}^{n} c_j x_j \tag{2–1}$$

$$\text{subject to} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, 2, ..., m, \tag{2–2}$$

3 Multi-dimensional Multi-choice Knapsack Problem (MMKP):

We have n groups of items. Group i has l_i items. Each item of the group has a particular value and it requires m resources. The objective of (MMKP) is pick exactly one item from each group for maximum total value of the collected items in mathematical notation, let v_{ij} and $\overrightarrow{r_{ij}} = (r_{ij1}, r_{ij2}, ..., r_{ijm})$ be the value (profit) and required resource vector of the object o_{ij}, i.e., jth of the ith group. Also assume that $\overrightarrow{R} = (R_1, R_2, ..., R_m)$ be the resource bound of the knapsack [?].
Now, the problem is to

$$(KP) \qquad \text{maximize} \sum_{j=1}^{n} p_j x_j \tag{3–1}$$

$$\text{subject} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{3–2}$$

$$x_j \in \{0,1\}, j = 1, ..., n. \tag{3–3}$$

where
$k = 1, 2, ..., m, x_{ij}$
in $\{0,1\}$
are the picking variables, and for all $i \in 1$ to n, $\sum_{j=1}^{l_i} x_{ij} = 1$.
Figure 1 illustrate a MMKP, We have to pick exactly one item from each group. Each item has two resources, r_1 and r_2. Clearly we must satisfy $\sum(r_1$ of picked items$) \leq 17$ and $\sum(r_2$ of picked items$) \leq 15$ and maximize the total value of picked items.

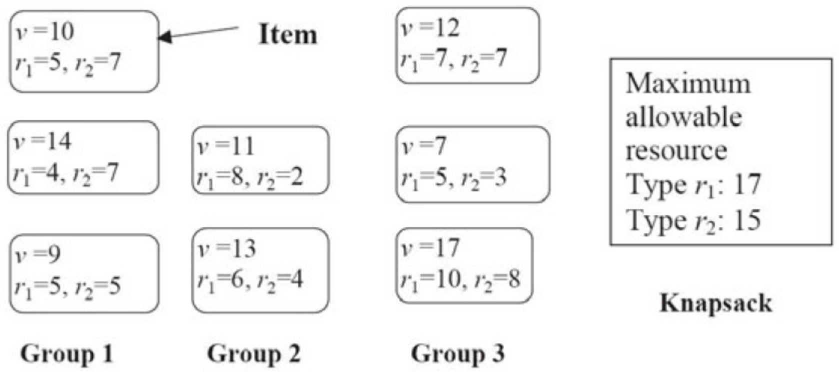


Figure 2.1: Multi-dimension Multi-choice Knapsack Problem

CHAPTER 3

THE OPTIMIZATION METHODS

Introduction:

Sometimes, there are more then one way to solve a problem. We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem. When analyzing an algorithm, we mostly consider a computational complexity which is divided into: Time complexity and Space complexity. Heuristic and metaheuristic techniques are used for solving computational hard optimization problem. Local search is a heuristic technique while Ant Colony Optimization (ACO), inspired by the ants' foraging behavior, is one of the most recent metaheuristic technique. These technique are used for solving optimization problems.

1 Optimization Methods:

Following the complexity of the problem, it may be solved by an exact method or an approximate method (Figure2). Exact methods obtain optimal solution and guarantee ether optimality. For NP-complete problems, exact algorithms are non polynomialtime algorithms (unless P=NP). Approximate (or heuristic) methods generate high quality solutions in reasonable time for practical use, but there is not guarantee of finding a global optimum solution.

2 Classification of optimization methods:

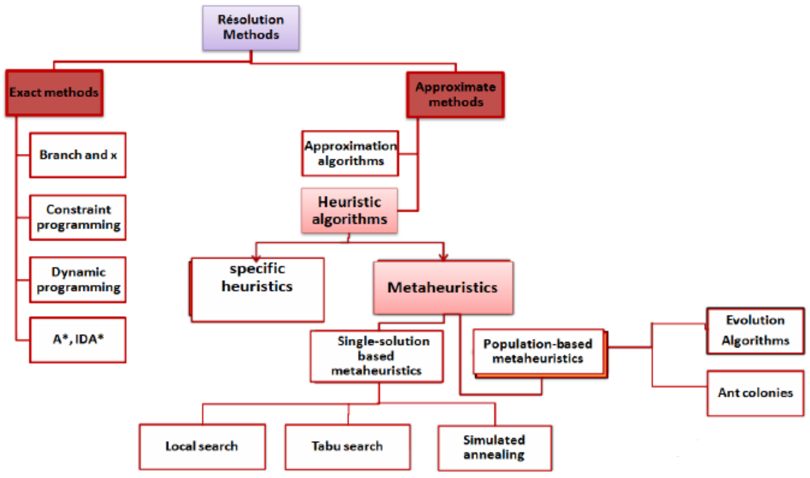


Figure 3.1: Classical Optimization Methods

2.1 Exact Methods:

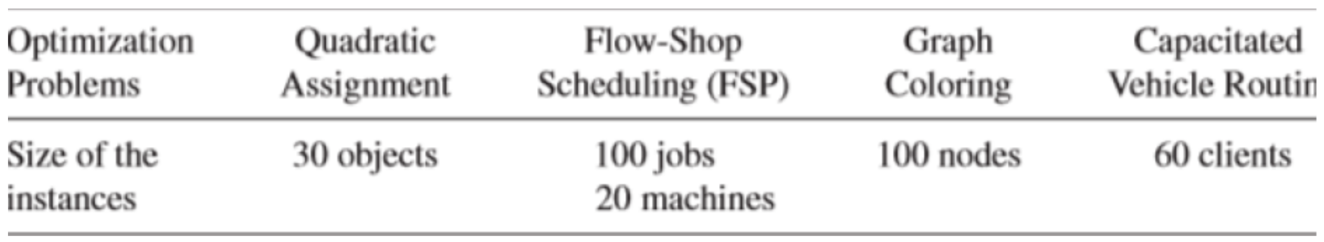
Exact methods seek to find the optimal solution with certainly by explicitly or implicitly examining the entire search space. They have advantage of guaranteeing the optimal solution, however, the computational time necessary for reaching this solution become very excessive depending on the size of the problem (combinatorial explosion)and the number of objectives to optimize. What limits the use of this type of methods for small size problems. These generic methods are: Branch & bound, Branch & cut and Branch & price, other methods are less general, such as: linear programming in integer, the algorithm of A *. Other methods are specific to a given problem like Johnson's algorithm for scheduling.
In class of exact methods we can find the following classical algorithms:
Dynamic programming, branch and X family of algorithms (branch and bound, branch and cut, branch and price) developed in operations research community, constraint algorithms, and A^*, IDA^* iterative deepening algorithms) developed in artificial intelligence community.
Those enumerative methods may be viewed as tree research algorithms.The search is carried out over the whole interesting search space, and the problem is solved by subdividing it into simple problems.
subsubsectionDynamic Programming: is based recursive division of a problem into simpler subproblems. This procedure is based into Bellman's principle that says that " The subpolicy of an optimal policy is itself optimal".
This stagewise optimization method is a result of a sequence of partial decisions. The procedure avoids the total enumeration of the search space by pruning partial decisions sequence that cannot lead to the optimal solution.

The branch and bound algorithms and A^*:

are based on an implicit enumeration of all solutions of the considered optimization problem. The search space is explored by dynamically building a tree whose root node represents the problem being solved and its whole associated search space.
The leaf nodes are the potential solutions and the internal nodes are subproblems of the total solution space. The pruning of the search tree is based on a bounding function that prunes subtrees that do not contain any optimal solution.

Constraint Programming:

is a language built around concepts of tree search and logical implications. Optimization problems in constraint programming are modeled by means of a set of variables linked by a set of constraints. The variables take their values on a finite domain of integers. The constraints may have mathematical or symbolic forms.
Exact methods can be applied to small instances of difficult problems. Table 2 shows for some popular NP hard optimization problems the order of magnitude of the maximal size of instances that state-of-the-art exact methods can solve to optimality.



Optimization Problems	Quadratic Assignment	Flow-Shop Scheduling (FSP)	Graph Coloring	Capacitated Vehicle Routin
Size of the instances	30 objects	100 jobs 20 machines	100 nodes	60 clients

Figure 3.2: TABLE 2 Order of Magnitude of the Maximal Size of Instances that State-of-the-Art Exact Methods can Solve to Optimality

Some of the exact algorithms used are implemented on large networks of workstations (grid computing platforms) composed of more than 2000 processors with more than 2 months of computing time [546]! The size of the instance is not the unique indicator that describes the difficulty of a problem, but also its structure. For a given problem, some small instances cannot be solved by an exact algorithm while some large instances may be solved exactly by the same algorithm.

2.2 Approximate methods:

In the class of approximate methods, two subclasses of algorithms may be distinguished: approximation algorithms and heuristic algorithms.Unlike heuristics ,which usually find reasonably "good" solutions in a reasonable time, approximation algorithms provide provable solution quality and provable run-time bounds. Heuristics find “good” solutions on large-size problem instances. They allow to obtain acceptable performance at acceptable costs in a wide range of problems. In general, heuristics do not have an approximation guarantee on the obtained solutions. They may be classified into two families: specific heuristics and metaheuristics. Specific heuristics are tailored and designed to solve a specific problem and/or instance. Metaheuristics are general purpose algorithms that can be applied to solve almost any optimization problem. They maybe viewed as upper level general methodologies that can be used as a guiding strategy in designing underlying heuristics to solve specific optimization problems.

Definition ϱ-Approximation algorithms:

An algorithm has an approximation factor ϱif its time complexity is polynomial and for any input instance it produces a solution a such that

$$a \leq \epsilon * \text{s if } \epsilon > 1 \tag{1–1}$$

$$\epsilon * s \leq \text{a if } \epsilon < 1 \tag{1–2}$$