A fast dynamic programming multi-objective knapsack problem

Marcos Daniel Valadão Baroni[*] Flávio Miguel Varejão

June 14, 2017

Abstract

This work addresses... The Multidi Objective knapsack programming. The dynamic programming method... The data structure...

1 Introduction

2 The Multiobjective Knapsack Problem

A general multiobjective optimization problem can be described as a vector function f that maps a tuple of n parameters (decision variables) to a tuple of k objectives. Formally:

$$\begin{aligned} &\text{min/max } \boldsymbol{y} = f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})\big) \\ &\text{subject to } \boldsymbol{x} = (x_1, x_2, \dots, x_n) \in X \end{aligned}$$

where $\boldsymbol{x}$ is called the *decision vector* or *solution*, X denotes the set of feasible solutions, and $\boldsymbol{y}$ is the *objective vector* or *criterion vector* where each objective has to be minimized (or maximized).

Considering two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$, a is said to *dominate* b if, and only if:

$$\begin{aligned} &\forall i \in \{1, 2, \dots, k\} : f_i(\boldsymbol{a}) \geq f_i(\boldsymbol{b}) \\ &\exists j \in \{1, 2, \dots, k\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{aligned}$$

A solution $\boldsymbol{a} \in X$ is called *efficient* or *non-dominated* if there is not other feasible solution $\boldsymbol{b} \in X$ such that $\boldsymbol{b}$ dominates $\boldsymbol{a}$. The set of solutions of a multiobjective optimization problem consists of all efficient solutions. This set is known as *Pareto optimal*.

The instance of a multiobjective knapsack problem with k objectives consists of an integer capacity $W > 0$ and n items. Each item i has a positive weight

[*]Research supported by Fundação de Amparo à Pesquisa do Espírito Santo.

w^i and k non negative integer profits $p_i^1, \dots, p_i^k$. A solution is represented by a vector $\boldsymbol{x} = (x_1, \dots, x_n)$ of binary decision variables x_i, such that $x_i = 1$ if item i is included in the solution and 0 otherwise, satisfing the capacity of the knapsack. For any instance of the problem, we aim at determining the set of efficient solutions.

Formally the definition of the problem is:

$$\begin{aligned} \max\ f(\boldsymbol{x}) &= \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})\big) \\ \text{subject to } w(\boldsymbol{x}) &< W \\ x_i &\in \{0,1\} \quad i = 1, \dots, n \end{aligned}$$

where

$$\begin{aligned} f_j(\boldsymbol{x}) &= \sum_{i=1}^{n} v_i^j x_i \quad j = 1, \dots, k \\ w(\boldsymbol{x}) &= \sum_{i=1}^{n} w_i x_i \end{aligned}$$

The MOKP is considered a $\mathcal{NP}$-Hard problem, since it is a generalization of the well-known $0-1$ knapsack problem and it is quite difficult to determine the Pareto optimal set for the MOKP, especially for high dimension instances, in which the Pareto set it self tends to grow exponentially. For this reason, the development of methods efficiently deal with high-dimension instances is...

3 The Dynamic Programing Algorithm

[1]

4 The use of data structure

The k-d tree is a type of binary search tree for indexing multidimensional data with simple construction and low space usage. Despite its simplicity it efficiently supports operations like nearest neighbour search and range search [2]. For those reasons k-d tree is widely used on spacial geometry algorithms [7, 3], clustering [5, 4] and graphic rendering algorithms [6].

Like a standard binary search tree, the k-d tree subdivides data at each recursive level of the tree. Unlike a standard binary tree, that users only one key for all levels of the tree, the k-d tree uses k keys and cycles through these keys for successive levels of the tree.

Concerning it's efficiency, it is important to consider the number of dimensions k-d tree is indexing. As a general rule, a k-d tree is suitable for efficiently indexing of n elements if n is much greater than 2^k. Otherwise, when k-d tree are used with high-dimensional data, most of the elements in the tree will be evaluated and the efficiency is no better than exhaustive search [8].

Indexing the solutions and range operations.

Tends to increase the feasibility on problems with higher dimensions.

5 Computational experiments

- Base de dados utilizaca
- Parametros dos algoritmos
- Anlise dos resultados (comparao)

6 Conclusions and future remarks

- Concluses dos resultados
- Trabalhos futuros

References

[1] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[4] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[5] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[6] John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

[7] Franco P Preparata and Michael Shamos. *Computational geometry: an introduction.* Springer Science & Business Media, 2012.

[8] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry.* CRC press, 2004.