A fast dynamic programming multi-objective knapsack problem

Marcos Daniel Valadão Baroni* Flávio Miguel Varejão

June 27, 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:

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

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

Considering two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$, $\boldsymbol{a}$ is said to *dominate* $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective. Formally:

$$dominates(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, k\} : f_i(\boldsymbol{a}) \geq f_i(\boldsymbol{b}) \text{and} \\ \exists j \in \{1, 2, \ldots, k\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

This relation is based on the dominance relation proposed by Weingartner and Ness for the multidimensional knapsack problem [10].

A feasible solution $\boldsymbol{a} \in X$ is called *efficient* if its not dominated by any other feasible solution. The set of all efficient solutions of a multiobjective optimization problem is known as *Pareto optimal*. Solving a multiobjective problem consists in giving its Pareto optimal set.

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

An instance of a multiobjective knapsack problem (MOKP) with k objectives consists of an integer capacity $W > 0$ and n items. Each item i has a positive weight w_i and non negative integer profits $p_i^1, p_i^2, \ldots, p_i^k$. Each profit p_i^k represents the contribution of the i-th item for k-th objective. A solution is represented by a set $\boldsymbol{s} \subseteq \{1, \ldots, n\}$ containing the indexes of the itens included in the solution. A solution is feasible if the total weight included in the knapsack does not exceed its capacity. Formally the definition of the problem is:

$$\begin{aligned} \max\ & f(\boldsymbol{s}) = \big(f_1(\boldsymbol{s}), f_2(\boldsymbol{s}), \ldots, f_k(\boldsymbol{s})\big) \\ \text{subject to}\ & w(\boldsymbol{s}) < W \\ & \boldsymbol{s} \subseteq \{1, \ldots, n\} \end{aligned}$$

where

$$f_j(\boldsymbol{s}) = \sum_{i \in \boldsymbol{s}} p_i^j$$

$$w(\boldsymbol{s}) = \sum_{i \in \boldsymbol{s}} w_i$$

The MOKP is considered a $\mathcal{NP}$-Hard problem since it is a generalization of the well-known $0-1$ knapsack problem, in which $k = 1$. It is quite difficult to determine the Pareto optimal set for the MOKP, especially for high dimension instances, in which the solution it self (Pareto optimal set) tends to grow exponentially. Even for the bi-objective case, small problems may prove intractable. For this reason we are interested in developing efficient methods for handling large solution sets, which may bring tractability to previously intractable instances.

3 The Dynamic Programing Algorithm

Paragrafo de introducao da secao, justificando toda a explicacao que segue..

We will introduce the dynamic programming algorithm for the MOKP proposed in [1]. This algorithm is based on the classical dynamic programming algorithm proposed in [6], although with some optimizations.

3.1 The basic dynamic programming algorithm

The sequential process used in the algorithm consists of n stages. At the i stage the algorithm generates a set S^k solutions that correspond to subsets containing exclusively the first i itens, i.e. , $\forall \boldsymbol{x} \in S^k, \boldsymbol{x} \subseteq \{1, \ldots, k\}$.

3.2 Avoiding deficient solutions

...those solutions

Algorithm 1 Basic dynamic programming algorithm for MOKP

1: **function** DP($\boldsymbol{p}, \boldsymbol{w}, W$)
2: $S^0 = \{\emptyset\}$
3: **for** $i \leftarrow 1, n$ **do**
4: $S_1^i = S^{i-1} \cup \{\boldsymbol{x} \cup i \mid \boldsymbol{x} \in S^{i-1}\}$
5: $S^i = \{\boldsymbol{x} \mid \nexists \boldsymbol{a} \in S_1^i : dominates(\boldsymbol{a}, \boldsymbol{x})\}$
6: **end for**
7: **end function**

3.3 Removing unpromissing solutions

3.4 Item order

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 [8, 3], clustering [5, 4] and graphic rendering algorithms [7].

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 [9].

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] George L Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[7] 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.

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

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

[10] H Martin Weingartner and David N Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.