Operations Research Letters 37 (2009) 23–26

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

On k-max-optimization

Jochen Gorski[a], Stefan Ruzika[b,∗]

[a] University of Wuppertal, Faculty of Mathematics and Natural Sciences, Department of Mathematics and Informatics, Gaußstr. 20, 41097 Wuppertal, Germany

[b] University of Kaiserslautern, Department of Mathematics, P.O.Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany

ARTICLE INFO

Article history:
Received 18 February 2008
Accepted 16 September 2008
Available online 17 October 2008

Keywords:
Combinatorial optimization
Bottleneck objective
Min–max
Bottleneck problem

ABSTRACT

We generalize bottleneck objectives in combinatorial optimization by minimizing the kth largest cost coefficient in a feasible solution. A bisection algorithm is presented which is based on iteratively solving an associated sum objective problem with binary cost coefficients. This algorithm is applicable to general combinatorial optimization problems.

1. Introduction

We consider combinatorial optimization problems where the kth largest cost coefficient of a feasible solution should be minimized. More precisely, let $E = \{e_1, \ldots, e_n\}$ denote a *ground set of elements* and let $F \subset 2^E$ denote a subset of the power set of E which we refer to as the *feasible set*. We call any $S \in F$ a *feasible solution*. Our considerations are restricted to problems where $|S| \geq m$ for some $m \in \mathbb{N}$ and all $S \in F$. To simplify notation we may assume without loss of generality that $|S| = m$ for all $S \in F$. Let $c: E \to \mathbb{Z}$ be a cost function on the elements of the ground set. We may assume that the elements of E are renumbered such that $c(e_1) \leq \cdots \leq c(e_n)$.

Typically, the objective of a combinatorial optimization problem is to minimize either the sum of costs $\sum_{e\in S} c(e)$ or the largest cost coefficient $\max_{e\in S} c(e)$ over all feasible solutions S. We refer to these problems as *combinatorial sum* (CSP) and *bottleneck* (CBP) *optimization problems*, respectively.

In this paper we introduce a new type of objective function for combinatorial problems which can be seen as a generalization of the bottleneck objective. The task is to find a feasible solution $S \in F$ minimizing the kth largest cost coefficient in S.

Let $S = \{e_{i_1}, \ldots, e_{i_m}\}$ be a feasible solution with $1 \leq i_1 < \cdots < i_m \leq n$. We define an operator k-max which yields the kth largest among the elements of S, i.e.,

$$k\text{-max}(S) = \underset{e\in S}{k\text{-max}}\, c(e) = c(e_{i_{m-k+1}}).$$

The problem of minimizing the kth largest cost coefficient can now be concisely formulated as

$$(k\text{MAX}) \quad \min_{S\in F}\ \underset{e\in S}{k\text{-max}}\, c(e).$$

The special case $k = 1$ is the well-known bottleneck or min-max problem.

To the best of our knowledge, problems of type (kMAX) have not been studied in the literature. However, the notation of the kth largest element of a feasible set S can be found in [1]. There, the minimization of the maximum deviation of the cost coefficients of a feasible solution $S \in F$ to its kth largest coefficient is discussed for general combinatorial optimization problems.

Although (kMAX) has not been considered, there exist several other interesting generalizations of CBP. Some of them are mentioned in the following since they slightly resemble (kMAX). In [2] the ground set E is partitioned into p non-empty disjoint subsets. For feasible $S \in F$ the objective is to minimize the sum of costs of those elements which have maximum cost in the subsets $S \cap E_k$ for $k = 1, \ldots, p$. The considered problem generalizes CBP and CSP simultaneously, since for $p = 1$ the problem simplifies to a CBP, while a CSP has to be solved for $p = |E|$. A generalized CSP is discussed in [3]. Instead of minimizing the complete sum of all cost coefficients of a feasible solution, only the sum of the k largest cost coefficients is considered. Note that this approach also contains the bottleneck case since for $k = 1$ the problem simplifies to a CBP, while for $k \geq \max\{|S| : S \in F\}$ an ordinary CSP has to be solved. In lexicographic CBP (see [4,5]), the largest cost coefficient has to be minimized in first place. Among all optimal solutions to this CBP, the second largest cost coefficient has to be optimized, then the third largest and so on. Moreover, there exist several studies on CBPs with fixed cardinality (for a survey, we refer to [6]), i.e. $|S| = m$ for all $S \in F$ is required.

∗ Corresponding author.
E-mail address: ruzika@mathematik.uni-kl.de (S. Ruzika).

Despite these existing generalizations of CBPs, (kMAX) may enable the modeling of many real world problems which could not have been easily formulated so far. We want to demonstrate briefly its potential in image registration. In general registration problems two given data sets have to be rendered in a joint coordinate system such that corresponding features of these data sets are aligned. Such data sets normally correspond to 2- or 3-dimensional images as it is the case for example in medical image registration. For a given model set A of n distinct points in the plane (which may correspond for example to characteristic points of a given reference image) and an image set B of the same cardinality (one may think of characteristic points in a template image), it is assumed that the points in B correspond to points in A but are afflicted with some data errors. The goal is to find the best possible assignment between the points of the two sets such that some distance measure between each pair of aligned points is as small as possible. Neither minimizing the average deviation of the points (which would be modeled by a CSP) nor the worst case assignment (this corresponds to a CBP) adequately deals with the noise-induced errors in set B. Instead, it seems more suitable to disregard those $k-1$ pairs of points which are furthest apart from each other. These $k-1$ assignments are considered outliers due to noise and the task in the optimization problem is to minimize the kth largest distance in the assignment of set B to set A.

2. Algorithm

Our algorithm for solving Problem (kMAX) is applicable to general combinatorial optimization problems since we only require the solution of a sequence of CSPs. We utilize bisection search for the kth largest cost coefficient in an optimal solution: In each iteration, we decide whether there exists a solution whose kth largest cost coefficient is smaller than a given cost coefficient $c(e_i)$, for a fixed $i \in \{1, \dots, n\}$. This decision is based on the solution of a sum problem having the same feasible set as Problem (kMAX).

Given $j \in \mathbb{N}$ with $1 \le j \le n$, we assign auxiliary costs to each element in the ground set by

$$d_j(e_i) := \begin{cases} 0, & \text{if } i \le j \\ 1, & \text{if } i > j. \end{cases}$$

The sum problem which is iteratively solved during the algorithm is given by

$$(SP)\ \min_{S \in F} d_j(S) := \sum_{e \in S} d_j(e).$$

Since the costs are binary, this sum problem may be easier to solve than general CSPs over the same feasible set. Indeed there exist combinatorial optimization problems where solving the sum objective problem is NP-hard in general, while for only two distinct cost coefficients on E there exists a polynomial time algorithm to solve problems of type (SP) (see [7] and Section 3).

First, we consider finiteness and correctness of Algorithm 1.

Theorem 1. *Algorithm 1 terminates in finitely many steps. The solution S it returns is optimal for (kMAX).*

Proof. In each iteration the value of LB or UB is increased or decreased by at least one unit, respectively. Therefore, termination is guaranteed. We prove the validity of the following loop-invariant which implies the correctness of the algorithm.

In each iteration of the **while**-loop, the index of the kth largest element of an optimal solution to (kMAX) is contained in the set $\{n \in \mathbb{N} :\ \text{LB} \le n \le \text{UB}\}$.

Initialization: Lines 2 and 3 ensure that the loop invariant is valid at the start of the first iteration.

Algorithm 1 Bisection Algorithm for k-max Problems

```
GENERALIZED BOTTLENECK
Input: A ground set $E$, a set of feasible solutions $F \subset 2^E$, and $k \in \mathbb{N}$.
Output: A feasible solution $S \in F$ being optimal to Problem ($k$MAX).
 1: Sort elements $e \in E$ such that $c(e_i) \le c(e_{i+1})$.
 2: LB $\leftarrow 1$
 3: UB $\leftarrow n$
 4: $j \leftarrow \lceil \frac{n}{2} \rceil$
 5: while $|\text{UB} - \text{LB}| > 0$ do
 6:    Assign costs $d_j$ to each $e \in E$.
 7:    Solve Problem (SP). Denote $S$ the optimal solution and $d(S)$
       the optimal value.
 8:    if $d_j(S) < k$ then
 9:       UB $\leftarrow j$
10:    else
11:       LB $\leftarrow j + 1$
12:    end if
13:    $j \leftarrow \text{LB} + \lfloor \frac{\text{UB}-\text{LB}}{2} \rfloor$
14: end while
15: Solve Problem (SP). Denote $S$ the optimal solution and $d(S)$ the
    optimal value.
16: return $S$.
```

Maintenance: In each iteration, a cost value of zero is assigned to all elements having an index smaller or equal to j. All other elements are assigned a cost value of one. Note that (kMAX) and (SP) have the same feasible set. We denote an optimal solution of (kMAX) by S^{opt}. The objective function value $d_j(S)$ of (SP) counts the minimum number of elements in S which have an index larger than j.

Case 1: $d_j(S) < k$.

Then there exists a feasible solution whose kth largest element of S is at most $c(e_j)$. Therefore, the index of the kth largest element of S^{opt} is contained in $\{LB, \dots, j\}$.

Case 2: $d_j(S) \ge k$.

By contradiction, suppose the cost of the kth largest element of S^{opt} is less or equal to $c(e_j)$. Consider the value $d_j(S^{\text{opt}})$. It is $d_j(S^{\text{opt}}) < k$ which is a contradiction to S being optimal for (SP). Thus, the index of the kth largest element of S^{opt} is contained in the set $\{j+1, \dots, \text{UB}\}$.

Termination: We repeat the **while**-loop until UB $=$ LB. In this case, $j = \text{UB} = \text{LB}$ and $c(e_j)$ is the optimal objective function value.

Due to the validity of the loop invariant, line 15 ensures that an optimal solution S of (kMAX) with $k\text{-max}(S) = c(e_j)$is returned. □

Note that it cannot be guaranteed that the optimal solution S for (SP) generated in the last iteration of the **while**-loop is also optimal for (kMAX) (cf. the example in Section 3). Therefore Problem (SP) has to be solved once more in line 15 after the optimal value for j has been found.

Lemma 2. *In the **while**-loop of Algorithm 1 at least one optimal solution S to (kMAX) is computed when solving Problems (SP) in line 7. More precisely, S is the optimal solution to the Problem (SP) for which the upper bound UB was updated the last time before the stopping criterion of the **while**-loop was satisfied.*

Proof. Let S denote the optimal solution to the Problem (SP), for which the upper bound was updated the last time before the stopping criterion of the **while**-loop was satisfied, and let

Table 1
In the first two rows the assigned values to c and p are listed, while in the subsequent rows the feasible solutions of (3) are shown.

$c(e)$	1	2	3	4	5	6	7	8	9	10	11	$\sum_{e\in S} p(e)$
$p(e)$	45	54	21	10	14	58	15	56	17	43	27	
S_1	1	1	0	0	0	$\boxed{1}$	0	1	0	1	1	283
S_2	1	1	$\boxed{1}$	0	0	1	0	1	0	1	0	277
S_3	1	1	0	0	0	$\boxed{1}$	0	1	1	1	0	273
S_4	1	1	0	0	0	$\boxed{1}$	1	1	0	1	0	271
S_5	1	1	0	0	$\boxed{1}$	1	0	1	0	1	0	270
S_6	1	1	0	$\boxed{1}$	0	1	0	1	0	1	0	266
S_7	1	1	$\boxed{1}$	0	0	1	0	1	0	0	1	261

$c(e_i) := k\text{-max}(S)$. By contradiction, suppose that S is not optimal for (kMAX), i.e., there exists another feasible solution $S^* \in F$ such that $c(e_{i^*}) := k\text{-max}(S^*) < c(e_i), i^* < i$. Without loss of generality we assume that $\mathrm{UB} = i$ (otherwise we have to perform at least one more subsequent iteration where the upper bound will be updated). According to the loop-invariant, $i^* \in \{\mathrm{LB}, \dots, \mathrm{UB} - 1\}$, and we have to perform at least one more iteration. Due to the definition of S all subsequent iterations will never lead to an update of the upper bound UB until the stopping criterion is met. So, $i^* \notin \{\mathrm{LB}, \dots, j\}$ for all $\mathrm{LB} \le j \le \mathrm{UB} - 1$ and after the final iteration $i^* = \mathrm{LB} = \mathrm{UB} = i$ holds true. This contradicts our assumption. □

Due to Lemma 2, line 15 in Algorithm 1 can be omitted when storing the solution S which has led to an update of the upper bound UB at the end of the **while**-loop. At termination this solution is then returned by the algorithm.

Note that in the course of iterations, an optimal solution to Problem (kMAX) is not always necessarily contained in the current set of optimal solutions to Problem (SP). Especially in earlier iterations only a coarse estimation of the position of the optimal cost coefficient to Problem (kMAX) is available. Thus, there may exist many feasible solutions having a better value with respect to the current objective d_j of Problem (SP) than optimal solutions of (kMAX).

Finally we analyze the running time of Algorithm 1.

Theorem 3. *The running time of Algorithm 1 is in $\mathcal{O}(T \log n)$ where T denotes the time needed for solving (SP).*

Proof. First, we note that n elements can be sorted in $\mathcal{O}(n \log n)$ which can be assumed to be in $\mathcal{O}(T \log n)$. Line 13 guarantees that the **while**-loop is bisected in every iteration. Consequently, there are $\log n$ many iterations. Line 6 is in $\mathcal{O}(n)$, line 7 is in $\mathcal{O}(T)$, all other operations in $\mathcal{O}(1)$ and, thus, the running time follows. □

3. Example

To demonstrate the properties of Algorithm 1 introduced in Section 2 we consider the (kMAX) cardinality constrained knapsack problem

$$\min_{S\in F}\ k\text{-}\max_{e\in S} c(e) \tag{1}$$

with ground set $E = \{e_1, \dots, e_n\}$ and feasible set

$$F = \left\{ S \in 2^E : |S| = m \wedge \sum_{e\in S} p(e) \ge \text{const.} \right\}$$

for arbitrary but fixed $0 \le m \le n$ and $0 \le k \le m$. The vectors $c \in \mathbb{R}^n$ and $p \in \mathbb{R}^n$ are called *cost vector* and *profit vector*, respectively. It is well known that the cardinality constrained knapsack problem with sum objective is NP-hard in general (see [8]). However, if the cost vector c is binary, i.e. $c \in \{0, 1\}^n$, as it is the case for Problem (SP), the sum-version of the problem can be solved in polynomial time which we will show by an algorithm. As a consequence, Problem (1) can also be solved in polynomial time although the same problem with sum objective is NP-hard in general. Next, we outline a brief description of our polynomial time algorithm for the cardinality constrained knapsack problem with binary costs:

(1) Partition the ground set $E = E_0 \cup E_1$ into two disjoint subsets $E_i := \{e \in E : c(e) = i\}, i = 0, 1$.
(2) Sort the elements of E_0 and E_1 in non-increasing order according to their profit $p(e)$.
(3) Construct a first solution candidate S which contains the first m most profitable elements of E_0, if $|E_0| \ge m$, or all elements of E_0 and the $m - |E_0|$ most profitable elements of E_1, if $|E_0| < m$, respectively.
(4) If the solution S satisfies the constraint
$$\sum_{e\in S} p(e) \ge \text{const.},$$
it must be optimal by construction.
(5) Otherwise iteratively replace the most unprofitable element $e_{\text{out}} := \operatorname{argmin}\{p(e) : e \in (S \cap E_0)\}$ of $S \cap E_0$ by the most profitable element $e_{\text{in}} := \operatorname{argmax}\{p(e) : e \in (E_1 \setminus S)\}$ of $E_1 \setminus S$ until the sum constraint (4) is met for the first time. If this is never the case the given problem is infeasible.

In the following we consider the 4-max cardinality constrained knapsack problem with the ground set $E = \{e_1, \dots, e_{11}\}$ and the feasible set

$$F := \{S \in 2^E : |S| = 6 \wedge \sum_{e\in S} p(e) \ge 261\},$$

where the profits p and the costs c are specified in the first two rows of Table 1. This problem has seven feasible solutions $S_1, \dots, S_7$. They are also listed in Table 1. Note that $c(e_i) = i$ for all $i \in \{1, \dots, 11\}$. Obviously, the solutions S_2 and S_7 are both optimal for the given 4-max-problem with $4\text{-max}(S_2) = 4\text{-max}(S_7) = c(e_3) = 3$.

The values assigned to the variables LB, UB, and j during the execution of the algorithm as well as the set of optimal solutions for each subproblem (SP) and the optimal function values with respect to d_j can be found in Table 2. The algorithm stops after three iterations. In the first two iterations, the optimal function value of d_j is less than $k = 4$. In these cases the upper bound UB

Table 2
The rows inform about the values of the algorithm before and after each iteration.

Iteration	S^{opt}	$d_j(S^{\text{opt}})$	LB	UB	j	$\lvert\text{UB}-\text{LB}\rvert$
Prior to 1st	–	–	1	11	6	10
1st	$\{S_2, S_5, S_6, S_7\}$	2	1	6	3	5
2nd	$\{S_2, S_7\}$	3	1	3	2	2
3rd	$\{S_1, \dots, S_7\}$	4	3	3	3	0

is updated using the current value of j. In the last iteration, the optimal function value of d_j equals 4, and the lower bound LB is updated to $j+1$. This leads to $\text{UB} = \text{LB} = j = 3$ and the stopping criterion of the while-loop is satisfied.

In the first iteration the set of optimal solutions for the subproblem (SP) consists of the four feasible solutions S_2, S_5, S_6, and S_7, while in the second iteration S_2 and S_7 are optimal for (kMAX). In the third iteration, all feasible solutions of (kMAX) are also optimal to (SP). According to the loop-invariant used in the proof of Theorem 1, $j = 3$ corresponds to the index of the optimal function value of (kMAX). Due to Lemma 2, S_2 and S_7 are both optimal for (kMAX), since they are optimal for (SP) in the iteration in which the upper bound is updated the last time before the stopping criterion of the **while**-loop is met. Note that the optimal solution S obtained when solving (SP) the last time during the **while**-loop does not have to correspond to an optimal solution to (kMAX) since e.g. $S = S_1$ is possible.

4. Future research

We propose a solution algorithm for the minimization of the kth largest cost coefficient in a feasible solution of a combinatorial optimization problem. The algorithm is based on a successive reformulation of the considered problem as a simple sum problem with binary costs over the same feasible set. Further research can focus on multiple objective optimization problems with more than one k-max objective function and the question whether the reformulation technique presented in this article can also be applied to these kind of problems. Moreover, the development of efficient algorithms for solving special classes of combinatorial optimization problems with a binary sum objective can be of interest.

Acknowledgements

The research of Stefan Ruzika has been partially supported by Deutsche Forschungsgemeinschaft (DFG) Grant HA 1737/7 and by the Center of Mathematical and Computational Modelling of the University of Kaiserslautern.

References

[1] A.P. Punnen, Y.P. Aneja, Lexicographic balanced optimization problems, Operations Research Letters 32 (2004) 27–30.
[2] A.P. Punnen, K.P.K. Nair, Y.P. Aneja, Generalized bottleneck problems, Optimization 35 (2) (1995) 159–169.
[3] A.P. Punnen, Y.P. Aneja, On k-sum optimization, Operations Research Letters 18 (1996) 233–236.
[4] F. Della Croce, V.T. Paschos, A. Tsoukias, An improved general procedure for lexicographic bottleneck problems, Operations Research Letters 24 (4) (1999) 187–194.
[5] P.T. Sokkalingam, Y.P. Aneja, Lexicographic bottleneck combinatorial problems, Operations Research Letters 23 (1) (1998) 27–33.
[6] M. Ehrgott, H.W. Hamacher, F. Maffioli, Fixed cardinality combinatorial optimization problems, Tech. Rep., University of Kaiserslautern, 1999.
[7] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrantz, S.S. Ravi, Spanning trees — short or small, SIAM Journal of Discrete Mathematics 9 (2) (1996) 178–200.
[8] J.B. Mazzola, R.H. Schantz, Single-facility resource allocation under capacity-based economies and diseconomies of scope, Management Science 41 (4) (1995) 669–689.