Bit Allocation in Sub-linear Time and the Multiple-Choice Knapsack Problem

Alexander E. Mohr*
Sieg 114, Box 352350
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195–2350
amohr@cs.washington.edu

Abstract

We show that the problem of optimal bit allocation among a set of independent discrete quantizers given a budget constraint is equivalent to the multiple choice knapsack problem (MCKP). This result has three implications: first, it provides a trivial proof that the problem of optimal bit allocation is NP-hard and that its related decision problem is NP-complete; second, it unifies research into solving these problems that has to-date been done independently in the data compression community and the operations research community; third, many practical algorithms for approximating the optimal solution to MCKP can be used for bit allocation. We implement the GBFOS, Partition-Search, and Dudzinski-Walukiewicz algorithms and compare their running times for a variety of problem sizes.

1 Introduction

The problem of optimal bit allocation among a set of independent discrete quantizers given a budget constraint has been widely studied by the data compression community [1, 2, 3, 4, 5]. Likewise, the multiple-choice knapsack problem (MCKP) has been extensively studied in the operations research community [6, 7, 8, 9, 10]. In this paper, we show that these two problems are equivalent, *i.e.*, an algorithm that solves one of these problems can be convered into an algorithm that solves the other within the same time and memory bounds.

This equivalence unifies twenty years of research into these problems and has a number of implications. First, it provides a trivial proof that finding an optimal solution to the bit allocation problem is NP-hard. Second, efficient approximation algorithms that have been

*Supported in part by National Science Foundation Grant CCR-0104800. The author may also be contacted via dcc2002@alexmohr.com. This paper appeared at the 2002 Data Compression Conference.

developed for MCKP can now be applied to bit allocation. Some of these algorithms run in linear time. Others run in sub-linear time when the convex hulls of the distortion-rate curves are known, as commonly occurs in bit allocation. We introduce the two problems and argue their equivalence; provide overviews of both linear time and sub-linear time algorithms; and present experimental results comparing implementations of these two algorithms to the GBFOS [3] algorithm.

2 The Two Problems

In this section, we first explain and then formulate the bit allocation problem and the multiple-choice knapsack problem.

2.1 The Bit Allocation Problem

The bit allocation problem can be formulated as follows. We are given a set of m input signals $s_1, \ldots, s_i$ such that each input signal s_i has a set of n_i possible reproductions. Quantizer q_{ij} produces the jth reproduction of source s_i with distortion d_{ij} and rate r_{ij}. If we use quantizer q_{ij} on s_i, we set indicator variable $x_{ij} = 1$ and otherwise set $x_{ij} = 0$. Given a rate budget R, our goal is to minimize the overall distortion D. Formalizing these relationships yields the following four equations:

$$\text{minimize} \quad D = \sum_{i=1}^{m} \sum_{j=1}^{n_i} d_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{j=1}^{n_i} r_{ij} x_{ij} \leq R, \tag{2}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \qquad i = 1, \ldots, m, \tag{3}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j. \tag{4}$$

The values of the x_{ij}'s encode a solution to the bit allocation problem: we select the reproduction produced by quantizer q_{ij} if and only if $x_{ij} = 1$.

2.2 The Multiple-Choice Knapsack Problem

The multiple-choice knapsack problem has a similar formulation. We are given a set of m "tables" $T_1, \ldots, T_m$ such that table T_i contains n_i "items." The jth item on table T_i has profit p_{ij} and weight w_{ij}. Given a knapsack with weight capacity C, our goal is to fill the knapsack by selecting one and only one item from each table such that our overall profit P is maximized without exceeding the knapsack's capacity, as illustrated in Figure 1. The "multiple-choice" term in the problem designation refers to the requirement of selecting exactly one item from each table, much as students are required to fill in exactly one oval for each problem on their multiple-choice tests. If we use indicator variable x_{ij} to indicate

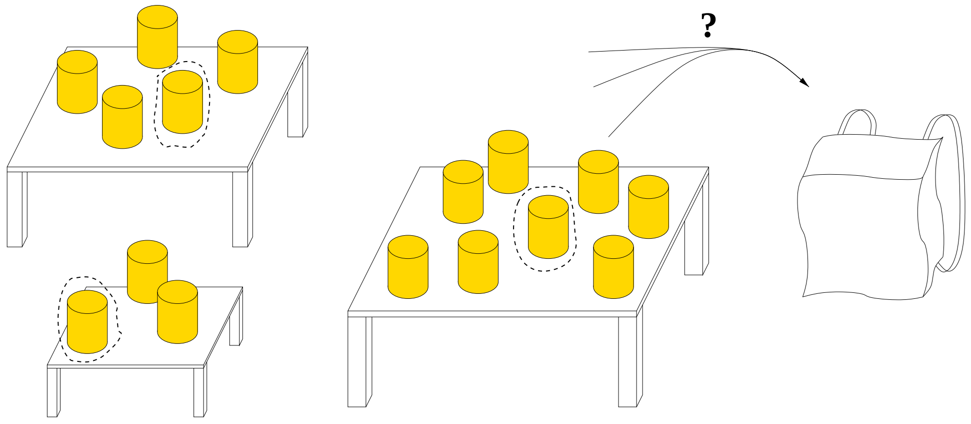


Figure 1: The multiple-choice knapsack problem. Each cylinder has a profit and weight associated with it.

whether the jth item from table T_i was selected, we can formalize the problem with these four equations:

$$\text{maximize} \quad P = \sum_{i=1}^{m} \sum_{j=1}^{n_i} p_{ij} x_{ij} \tag{5}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ij} x_{ij} \leq C, \tag{6}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \qquad i = 1, \ldots, m, \tag{7}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j. \tag{8}$$

Clearly, if we set $p_{ij} = -d_{ij}$, $r_{ij} = w_{ij}$, $P = -D$, and $C = R$, Equations 1–4 will be identical to Equations 5–8. The same set of indicator variables $\{x_{ij} | i = 1, \ldots, m$ and $j = 1, \ldots, n_i\}$ will be an optimal solution to both problems and an algorithm that finds such a set for one problem can be used to find such a set for the other. Therein lies a trivial proof of NP-hardness for bit allocation: because these two problems are equivalent and MCKP is well-known to be NP-hard (*e.g.*, [9]), optimal bit allocation is also NP-hard.

3 Approximation Techniques

It is possible to exactly solve the above problems using branch-and-bound algorithms, but because the worst-case running time of these algorithms is exponential in both the number of tables and the number of items on each table, branch-and-bound algorithms are often too slow to be useful. An alternative approach is to relax some number of constraints, solve the relaxed problem exactly, and use the resulting solution to the relaxed problem to construct a heuristic solution for the original problem.

3.1 Lagrangian Relaxation

An approach that has become popular for the bit allocation problem is *Lagrangian relaxation*, in which Equations 1 and 2 are combined into the following:

$$\text{minimize} \quad D = \sum_{i=1}^{m} \sum_{j=1}^{n_i} (d_{ij} + \lambda r_{ij}) x_{ij}. \tag{9}$$

3.2 Linear Relaxation

For the MCKP, a more common approach is *linear programming relaxation* in which the integrality constraint on x_{ij} is removed by replacing Equation 8 with:

$$0 \leq x_{ij} \leq 1, \qquad \forall i, j. \tag{10}$$

This relaxed problem is often called the *linear MCKP* or *LMCKP* and it is relatively easy to solve. An interesting property of LMCKP is that the solution that maximizes P represents an upper bound on the MCKP from which it derives. This upper bound is also at least as tight as an upper bound resulting from Lagrangian relaxation and is potentially better.

4 Approximation Algorithms

The known approximation algorithms for MCKP can be broadly divided into three classes: those that work on arbirary inputs, those that require the items in each set to be sorted by either weight or profit, and those that require the weights and profits of each set to form a convex hull.

4.1 Arbitrary Inputs

When the convex hulls are not known, the problem can be solved by using algorithms due to Dyer [7] and Zemel [8] that are linear in the total number of items, or $O(\sum_{i=1}^{m} n_i)$. If each table contains the same number of items, n, then these algorithms run in time $O(mn)$. It is also possible to use a Lagrangian relaxation which has time $O(mn)$ for each value of λ that is evaluated. Theoretically, Shoham and Gersho [2] can be extended to run in strictly linear expected time, but that approach has not yet been implemented to our knowledge.

Alternatively, one could sort the items in each set by profit or weight, or find the convex hull in $O(n\log n)$ time, where $n'_i \leq n_i$ is the number of points on the convex hull of table T_i [11]. Once the convex hull is known, a more specialized algorithm can be applied.

4.2 Sorted Inputs

A second option is available when the items on each table are sorted by either their profit or weight: Graham's scan [12] or the incremental method [13] will find the convex hull for table T_i in $O(n_i)$ time, or total time that is linear in the total number of reproductions. Again, once the convex hull is known, any of the algorithms for that more restricted condition can be applied.

4.3 Convex Hulls Known

In this paper, we focus on the situation when the convex hulls are known and present three algorithms: GBFOS [3], an algorithm we call Partition-Search, and the algorithm of Dudzinski and Walukiewicz [6]. In the following analyses of running times, we assume that each table contains the same number of items n; *i.e.*, $n_i = n, \forall i$.

4.3.1 GBFOS

The GBFOS algorithm [3] works by repeatedly merging pairs of sorted sets. In $O(mn\log m)$ time, it finds the complete convex hull of all possible rates and distortions. Once the complete convex hull is known, a simple search locates a point on that hull that has rate as large as possible, yet satisfies the bit budget.

4.3.2 Partition-Search

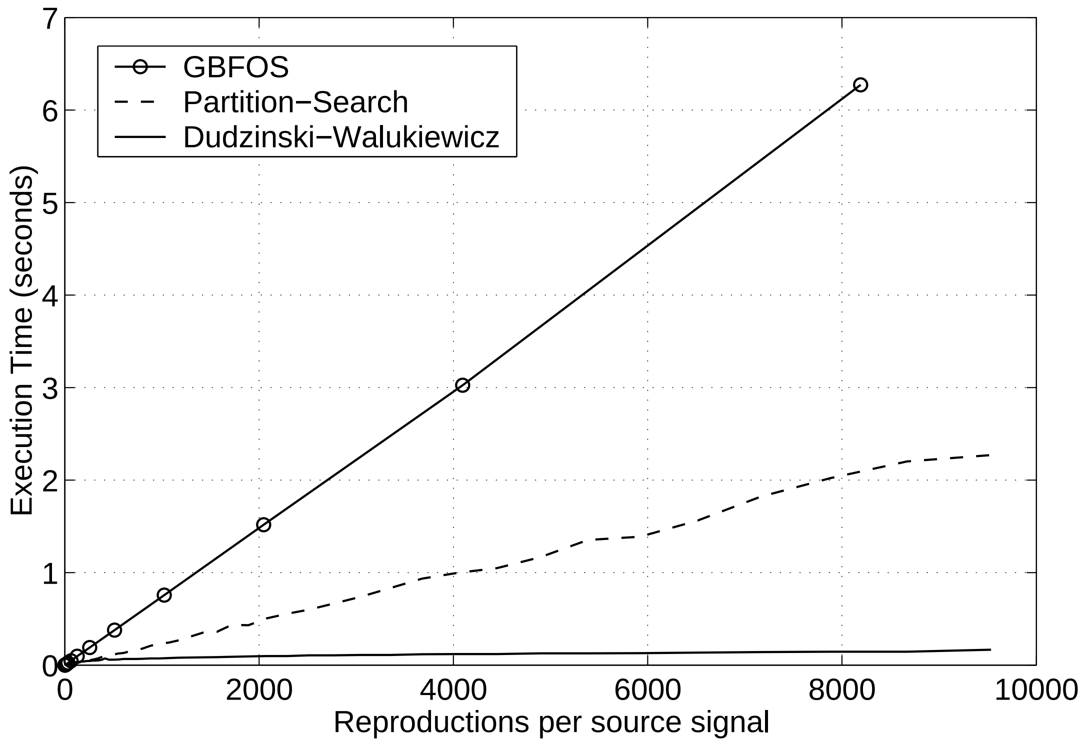
Partition-Search can be used to find a solution in $O(mn)$ time. Partition-Search first finds the median over all slopes between successive points on each convex hull. Any linear time median algorithm [13] will accomplish this task in $O(mn)$ time. By partitioning each convex hull around an item such that the slope from its predecessor on the hull is greater than the median and the slope from it to its successor on the hull is less than the median, we can test whether setting $x_{ij} = 1$ for these partitioning items will exceed our capacity C. If it does, then we only search among slopes greater than the median in the future and can discard all points with smaller slopes from further consideration. Likewise, if we have excess capacity remaining, then we need search only among slopes less than the median and can discard all points with higher slopes from further consideration. By repeating this process with the remaining points, we remove at least half of the points in each iteration. Though there will be $O(\log mn)$ iterations, each successive iteration is half as expensive and the overall time remains $O(mn)$.

4.3.3 Dudzinski-Walukiewicz

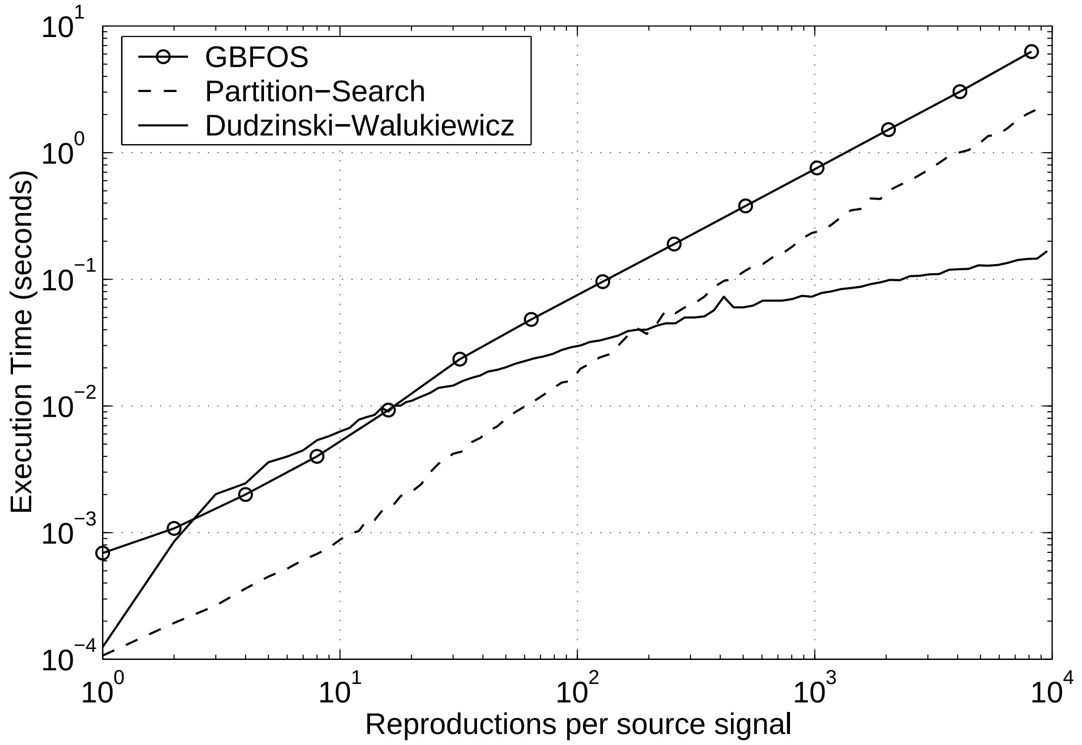
The algorithm of Dudzinski and Walukiewicz [6] runs in $O(m\log^2 n)$ time, sub-linear in the number of reproductions. They achieve this sub-linear time by exploiting the selection algorithm of Frederickson and Johnson [14], which can find the median of an $n \times m$ matrix with sorted columns in $O(m\log n)$ time. With $O(\log n)$ iterations of the $O(m\log n)$ algorithm, the resulting complexity is $O(m\log^2 n)$. These algorithms and their accompanying proofs of correctness are reasonably complex, so we refer the interested reader to the cited articles and move now to explore their real-world performance.

5 Evaluation of the Algorithms

We implemented GBFOS [3], Partition-Search, and Dudzinski-Walukiewicz [6] to determine how useful these algorithms are in practice and also to compare their performance



(a)



(b)

Figure 2: Execution times for GBFOS, Partition-Search and Dudzinski-Walukiewicz for $m = 1,000$ sources against the number of available reproductions per source. (a) Normal scale axes. (b) Log-Log scale axes.

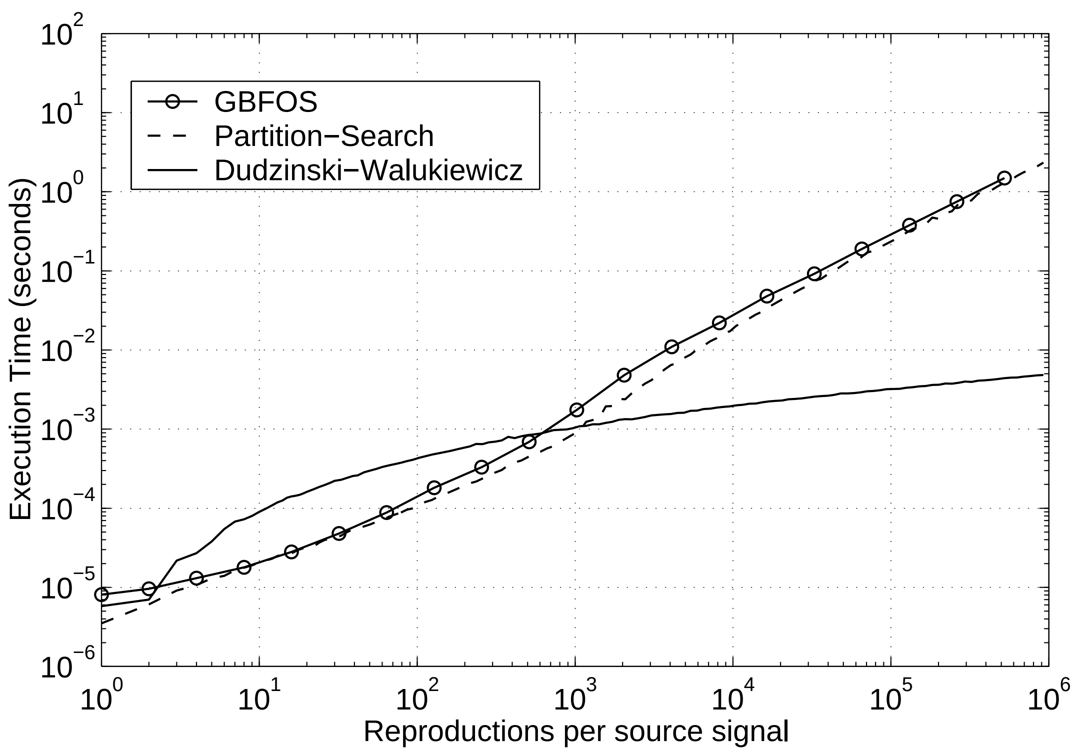


Figure 3: Execution times for GBFOS, Partition-Search, and Dudzinski-Walukiewicz for $m = 10$ reproductions against the number of reproductions per source.

for inputs of different sizes. Note that these implementations are relatively straightforward prototypes of the underlying algorithms: extensive optimization was not attempted.

Each reported trial represents the average execution time required to solve ten randomly generated instances of the given problem size. Because of timer granularity issues on the hardware (Intel Pentium III, 933 MHz running Linux 2.4.7) and a randomized partition algorithm that was the basis for linear time selection subroutines, reported execution times varied by a few percent from execution to execution. The total number of quantizers was limited to about 10 million to ensure that all experiments ran in main memory. The solutions found by all three algorithms were identical, so here we report only execution time.

In Figures 2a and 2b, we graph execution time versus increasing numbers of reproductions for each source (n) but fix the number of sources at $m = 1,000$. GBFOS is always at least six times slower than Partition-Search. With few reproductions for each source signal, the $O(m\log^2 n)$ Dudzinski-Walukiewicz algorithm is eight times slower than the $O(mn)$ Partition-Search algorithm; however, when the number of reproductions exceeds 200, the asymptotic advantage of Dudzinski-Walukiewicz overcomes its larger constant factors hidden by the big-O notation. When the number of reproductions rises to $9,500$, Dudzinski-Walukiewicz is an order of magnitude faster at 167ms compared to the 2.27s of Partition-Search.

In Figure 3, we again plot execution time versus the number of reproductions per source, but this time fix the number of sources at a small value, $m = 10$. As compared to the previous graph with $m = 1,000$, the crossover point at which Dudzinski-Walukiewicz

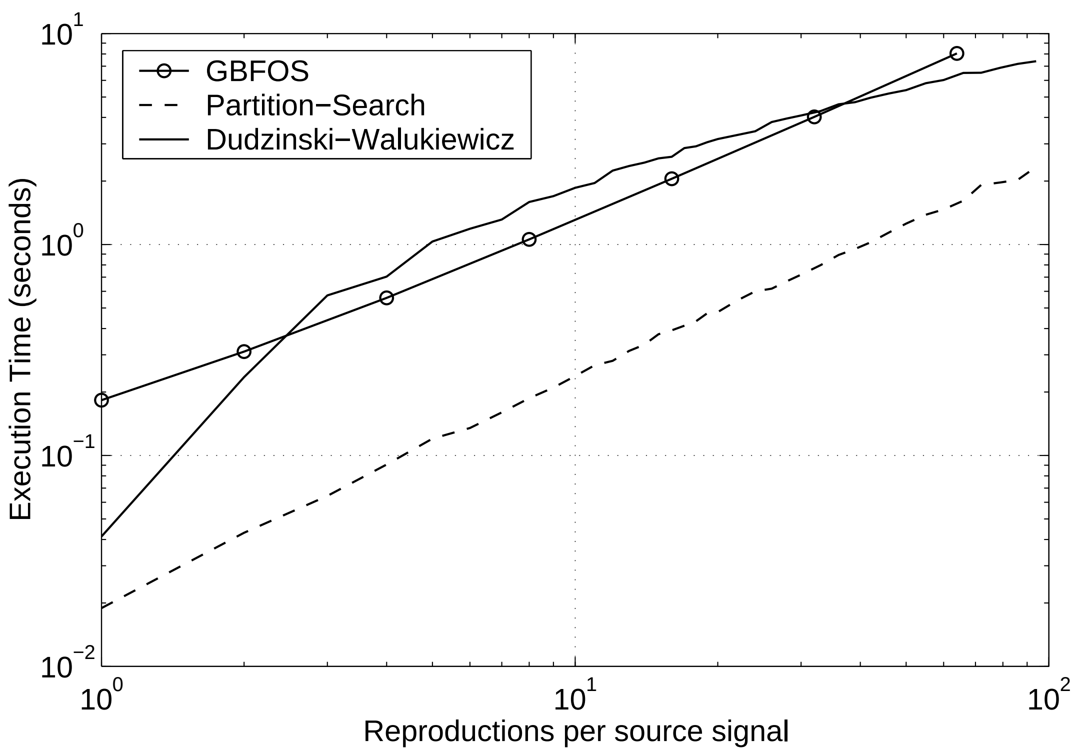


Figure 4: Execution times for GBFOS, Partition-Search, and Dudzinski-Walukiewicz for $m = 100,000$ sources against the number of reproductions per source.

outperforms Partition-Search is $n = 1,000$ rather than $n = 200$. Although Partition-Search always outperforms GBFOS, the improvement is relatively minimal as the additional log$m$ factor of GBFOS is insignificant at small m. Dudzinski-Walukiewicz performs at most a factor of five worse than Partition-Search when the number of reproductions is small, yet both algorithms use microseconds of CPU time on such problems so the factor of five is unlikely to be noticeable. At the other extreme, when $n = 1$ million, Dudzinski and Walukiewicz's algorithm [6] is three orders of magnitude faster than Partition-Search, though it seems unlikely that problem instances with one millon reproductions will often be encountered in practice.

In the final graph, Figure 4, we show results for $m = 100,000$ reproductions. With the constraint of 10 million total reproductions, n was at most 100, so Dudzinski-Walukiewicz was never faster than Partition-Search. If the number of reproductions for each source was allowed to grow larger than 100, however, Dudzinski-Walukiewicz would eventually be more efficient than either of the other two algorithms, as we saw in Figure 2.

6 Discussion

In all of these experiments, Partition-Search outperforms GBFOS in execution time, though it should be noted that Partition-Search finds an allocation for a single rate whereas GB-FOS generates the convex hull of all rates. How likely is Dudzinski-Walukiewicz to provide better performance than GBFOS or Partition-Search in practical situations? It usually outperforms the linear-time algorithm when the number of reproductions grows from $n = 100$ and $n = 1,000$. For an application like JPEG 2000 configured to use a small number of subbands as sources and the corresponding bitplanes as the available reproductions, Partition-Search is likely to perform best. On the other hand, a full-search entropy-constrained vector quantizer with at least 1000 elements in its codebook will almost certainly see performance gains by using the Dudzinski and Walukiewicz algorithm.

Improvements to the implementations are certainly possible, as is creating a hybrid that adapts to parameters and calls the algorithm most suited to a given situation. We also point out that execution time can be reduced by approximating the n' points on a convex hull by n'' points, $n'' < n'$, which will trade off bit allocation accuracy for decreased execution time.

7 Related Work

Batra [15] used MCKP as a model for bit allocation as part of work on object-based scalability. He treated bit allocation as a special case of MCKP and included extensive coverage of Dyer's [7] and Zemel's [8] $O(mn)$ algorithms for solving MCKP when the convex hulls are not known. He also added inter-dependencies and non-linearities to the formulation to allow dependent quantization, for example. He included performance results for IBM's optimization solutions software package on a Pentium 233 MHz PC for 37 sources, each with 13 reproductions, and reported execution times of 1.2s for LMCKP and 1.5–30.9s for exact MCKP branch and bound computations. We focus instead on approximation algorithms for bit allocation when the convex hulls are known and demonstrate that they can viably solve problems with millions of quantizers in fractions of a second.

8 Conclusion

We have shown that optimal bit allocation among a set of independent discrete quantizers with an overall budget constraint is equivalent to MCKP. We use this equivalence to provide a trivial proof of the NP-hardness of bit allocation and to apply a sub-linear time approximation algorithm to the problem of bit allocation when the convex hulls are known. We implemented three bit allocation algorithms and evaluated their performance on a variety of problem sizes.

In the future, we will apply these algorithms to bit allocation for ECVQ, JPEG 2000, and unequal loss protection. We are also investigating the selection algorithm of Frederickson and Johnson [14] and attempting to extend it to solve the weighted selection problem in less than $O(m\log^2 n)$ time. If we succeed, we expect to improve MCKP performance over Dudzinski and Walukiewicz [6].

Acknowledgments.
I wish to thank Richard Ladner and Eve Riskin for helpful discussions about this work, as well as Sarah Schwarm and the reviewers for their suggestions.

References

[1] A. V. Trushkin, “Optimal bit allocation algorithm for quantizing a random vector,” *Probl. Inf. Transmission*, vol. 17, pp. 156–161, July-Sept. 1981. Translated from Russian.

[2] Y. Shoham and A. Gersho, “Efficient bit allocation for an arbitrary set of quantizers,” *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 36, pp. 1445 – 1453, Sept. 1988.

[3] E. A. Riskin, “Optimum bit allocation via the generalized BFOS algorithm,” *IEEE Transactions on Information Theory*, vol. 37, pp. 400–402, Mar. 1991.

[4] X. Wu, “Globally optimal bit allocation,” in *Proceedings Data Compression Conference*, (Snowbird, Utah), pp. 22–31, Apr. 1993.

[5] K. Ramchandran and M. Vetterli, “Best wavelet packet in a rate-distortion sense,” *IEEE Trans. Image Processing*, vol. 3, pp. 533–545, 1994.

[6] K. Dudzinski and S. Walukiewicz, “A fast algorithm for the linear multiple-choice knapsack problem,” *Operations Research Letters*, vol. 3, pp. 205–209, 1984.

[7] M. E. Dyer, “An $O(n)$ algorithm for the multiple-choice knapsack linear program,” *Mathematical Programming*, vol. 29, pp. 57–63, May 1984.

[8] E. Zemel, “An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems,” *Information Processing Letters*, vol. 18, pp. 123–128, 1984.

[9] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, NY: J. Wiley and Sons, 1990.

[10] D. Pisinger, *Algorithms for Knapsack Problems*. Copenhagen, Denmark: University of Copenhagen, 1995.

[11] D. G. Kirkpatrick and R. Seidel, “The ultimate planar convex hull algorithm?,” *SIAM Journal on Computing*, vol. 15, no. 2, pp. 287–299, 1986.

[12] R. L. Graham, “An efficient algorithm for determining the convex hull of a finite planar set,” *Information Processing Letters*, vol. 1, pp. 132–133, 1972.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, Massachusetts, U.S.A.: M.I.T. Press, 1990.

[14] G. N. Frederickson and D. B. Johnson, “The complexity of selection and ranking in X + Y and matrices with sorted columns,” *Journal of Computer and System Sciences*, vol. 24, pp. 197–208, 1982.

[15] P. Batra, “Modeling and efficient optimization for object-based scalability and some related problems,” *IEEE Transactions on Image Processing*, vol. 9, pp. 1677–1692, Oct. 2000.