Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 1259–1273
computers & operations research
www.elsevier.com/locate/cor

Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls

Md Mostofa Akbar[a, b, *], M. Sohel Rahman[b], M. Kaykobad[b], E.G. Manning[a], G.C. Shoja[a]

[a] *Department of CSC, PANDA Lab, University of Victoria, Victoria, BC, Canada, USA*
[b] *Department of CSE, BUET, Dhaka, Bangladesh*

Available online 5 November 2004

Abstract

This paper presents a heuristic to solve the Multidimensional Multiple-choice Knapsack Problem (MMKP), a variant of the classical 0–1 Knapsack Problem. We apply a transformation technique to map the multidimensional resource consumption to single dimension. Convex hulls are constructed to reduce the search space to find the near-optimal solution of the MMKP. We present the computational complexity of solving the MMKP using this approach. A comparative analysis of different heuristics for solving the MMKP has been presented based on the experimental results.

Keywords: Algorithms; Convex hull; Heuristics; MMKP; Multimedia systems

1. Introduction and definition of MMKP

The classical 0–1 Knapsack Problem (KP) is to pick up items for a knapsack for maximum total value, so that the total resource required does not exceed the resource constraint R of the knapsack. The classical 0–1 KP and its variants are used in many resource management applications such as cargo loading, industrial production, menu planning, and resource allocation in multimedia servers [1–3]. Let there be *n items* with *values* $v_1, v_2, \ldots, v_n$ and let the corresponding resources required to pick the items be $r_1, r_2, \ldots, r_n$, respectively. The items can represent *services* and their associated values can be values of

* Corresponding author. Department of CSE, BUET, Dhaka, Bangladesh.
E-mail address: mostofa@cse.buet.ac.bd (Md Mostofa Akbar).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.09.016

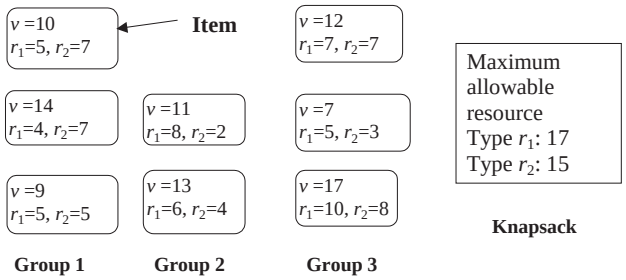

Fig. 1. Multidimensional Multiple-choice Knapsack Problem.

revenue earned from that service. In mathematical notation, the 0–1 KP is to find $V = \text{maximize} \sum_{i=1}^{n} x_i v_i$, subject to the constraint $\sum_{i=1}^{n} x_i r_i \leqslant R$ and $x_i \in \{0, 1\}$. The 0–1 KP is an NP-hard problem [4]. There is a pseudo-polynomial algorithm running in $O(nR)$ time using the concept of dynamic programming.

The Multidimensional Multiple-choice Knapsack Problem (MMKP) is a variant of the classical 0–1 KP. Let there be *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 the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to m resource constraints of the knapsack. In mathematical notation, let v_{ij} be the value of the jth item of the ith group, $\vec{r}_{ij} = (r_{ij1}, r_{ij2}, \ldots, r_{ijm})$ be the required resource vector for the jth item of the ith group, and $\vec{R} = (R_1, R_2, \ldots, R_m)$ be the resource bound of the knapsack. Now, the problem is to

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij} \ \text{(objective function)},$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} r_{ijk} \leqslant R_k \ \text{(resource constraints)},$$

where V is the *value of the solution*, $k = 1, 2, \ldots, m$, $x_{ij} \in \{0, 1\}$ are the picking variables, and $\sum_{j=1}^{l_i} x_{ij} = 1$.

Fig. 1 illustrates an MMKP. We have to pick exactly one item from each group. Each item has two resources, r_1 and r_2. The objective of picking items is to maximize the total value of the picked items subject to the resource constraints of the knapsack, that is $\sum(r_1 \text{ of picked items}) \leqslant 17$ and $\sum(r_2 \text{ of picked items}) \leqslant 15$. Notably, it may happen that no set of items satisfying the resource constraints exists implying that no solution will be found.

The rest of the paper is organized as follows. Related research works on MMKP and on the KPs, in general, have been described in Section 2. We present our main contribution in Section 3, where we describe the convex hull approach to solve the MMKP. The pseudo-code and corresponding computational complexity of the heuristic for solving the MMKP have been presented in this section. Section 4 shows the experimental results in a number of graphs and Section 5 describes the observation from the experimental results. Section 6 concludes the paper by summarizing the performance and applications of this heuristic.

2. Literature review

There are various algorithms for solving variants of KPs [4]. The Multidimensional Knapsack Problem (MDKP) is one kind of KP where the resources are multidimensional, i.e. there are multiple resource constraints for the knapsack. The Multiple Choice Knapsack Problem (MCKP) is another KP, where the picking criteria for items are restricted. In this variant of KP there are one or more groups of items with the constraint that exactly one item has to be picked from each group. Actually, the MMKP is the combination of the MDKP and the MCKP. It is worth mentioning here that there is another variant of KPs in the literature very similar to MMKP, where the restriction of picking exactly one item from each group is relaxed, i.e. you can either pick one item from a group or leave it. This variant of KP seems to be easier than the MMKP since we can exclude items from one or more groups if they are non-profitable.

Depending on the nature of the solution, the algorithms for MMKP can be divided into two groups, namely exact algorithms, which strive for exact solutions, and heuristic algorithms, where we are satisfied in near-optimal solutions. Finding exact solutions is NP-hard. Using the Branch and Bound with Linear Programming (BBLP) technique, Kolesar [5], Shih [6], Nauss [7] and Khan [3] presented exact algorithms for 0–1 KP, MDKP, MCKP and MMKP, respectively. It may be noted in this regard that although the search space for a solution in MMKP is smaller than the search space in other variants of KP, exact algorithms are not applicable to the various practical problems, e.g. on-line admission control problem. This is because of the existence of more restriction of picking items from a group in an MMKP instance. Experimental results in [8] present the time requirements for BBLP algorithms. Interested readers are referred to [9] for the new trends of exact algorithms.

On the other hand, there exist a number of heuristics in the literature for MMKP and for KPs in general. For example, a greedy approach has been proposed [4,10] to find near-optimal solutions of KPs. For a 0–1 KP as described in the previous section, items are picked from the top of a list sorted in descending order on v_i/r_i (value per unit resource) because these items seem to be the valuable and profitable items. To apply the greedy method to the MDKP, Toyoda proposed a measurement called *aggregate resource consumption* [11]. Khan [3] has applied the concept of aggregate resource consumption to pick a new candidate item in a group to solve the MMKP. This heuristic, named HEU, finds a solution by only upgrading the selected items of each group. Again, in [12], a modified version of HEU named M-HEU was presented, where a pre-processing step to find a feasible solution and a post-processing step to improve the total value of the solution with one upgrade followed by one or more downgrades were added. M-HEU provides solutions with total value on average equal to 96% of the optimum, with a worst-case time complexity of $O(mn^2(l-1)^2)$. Here, n is the number of groups, l the number of items in each group (assumed constant for convenience of analysis) and m the resource dimension.

Magazine and Oguz [1] proposed another heuristic based on Lagrange multipliers to solve the MDKP. Moser's [13] heuristic also used the concept of graceful degradation from the most valuable items based on Lagrange multipliers to solve the MMKP.

Various other techniques like tabu search [14], simulated annealing [15] and genetic algorithms [16] can also be applied to solve the variants of KP. The genetic algorithm has the exponential worst-case complexity—it can explore all the items. Tabu search and simulated annealing are based on looking at the neighbours. These are costlier than the greedy approach used in HEU. HEU uses a two-way interchange approach and searches candidates in the neighbourhood which yield better revenue, and changes one selection to another. But in tabu search, simulated annealing and genetic algorithm approach, current solution is moved to another solution by upgrading some and downgrading some. This upgrade and

downgrade at the same step requires more time because we have to search all neighbouring combinations of current solution. In the next section, we present our main contribution by presenting a new heuristic to solve MMKP by constructing convex hulls.

3. Solving the MMKP by constructing convex hulls

The *convex hull* of the points in a two-dimensional space is defined by the smallest convex polygon containing those points. There are two different groups of line segments in the convex hull connecting the bottommost and the topmost points. Each of these groups of line segments are called *frontier* of the convex hull. In [17], Lee has efficiently used the idea of convex hull to solve the quality of service (QoS) management problem of the Multiple Resource Multiple Dimension (MRMD) systems. The QoS controller, proposed by Lee, of the MRMD system transforms each multidimensional resource to a single dimension by multiplying a penalty vector. Now each QoS level for each session represents a point in the two-dimensional space. The offered bid price represents the y co-ordinate and the transformed resource represents the x co-ordinate. A convex hull is constructed for each session with the points represented by the QoS levels. Admission control and QoS adaptation of a session are done based on the gradient of the segments of the *efficient convex hull frontier* [18], where *Gradient* is a vector that always points in the direction of maximum change, with a magnitude equal to the slope of the tangent to the curve at the point, and the efficient convex hull frontier is the frontier, which earns more revenue in terms of resource usage.

The QoS management problem of the MRMD system can be easily mapped to an MMKP as is evident from Fig. 2. So the solution algorithm proposed by Lee can be applied to solve the MMKP. However, the MRMD system has some restrictions between required resources for a QoS level (an item in a group of the MMKP) and associated utility (value associated with the items). The QoS levels follow a special monotone feasibility order, i.e., a QoS level with higher utility must require higher resource requirements. As a result, for the MRMD system the gradients of these segments never become negative, which may not necessarily be the case for an MMKP instance. In particular, this algorithm is not applicable for those

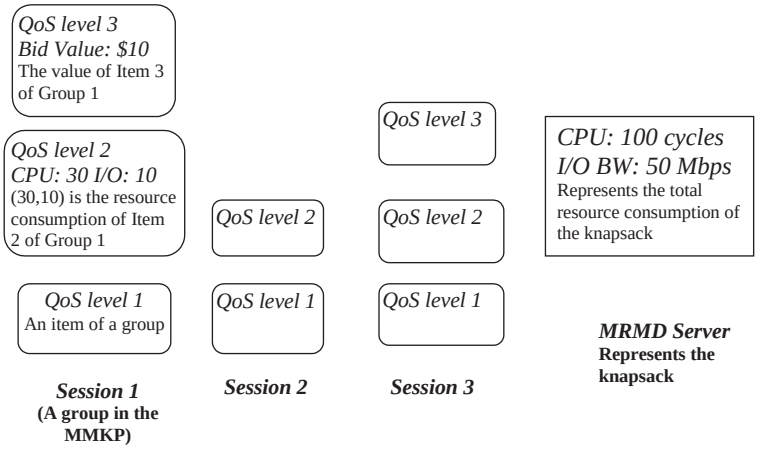

Fig. 2. Mapping of MRMD system to the MMKP.

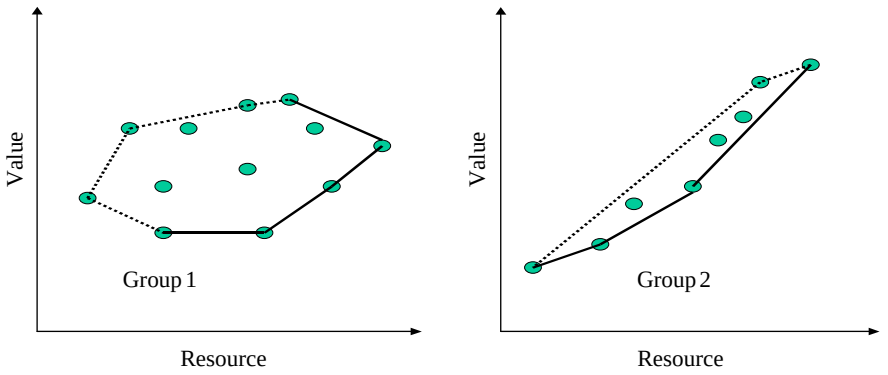

Fig. 3. Convex hulls of the items of two groups.

MMKPs where some higher-valued items require less resource consumptions than lower-valued items because they will create negative gradients and hence the corresponding segments will be at the end of the sorted list; whereas since they give higher utility using less resources, they should be considered first and hence should be at the front of the sorted list. We use a different sorting criterion to make it applicable for the MMKP as follows. We sort the segments according to the angle between the positive x-axis and the segment. Thus, the negative gradients are mapped to angles higher than 180°, which ensures their place in the beginning of the sorted (sorted in descending order) list. Therefore, these are selected for upgrading in the beginning. This makes the selection criterion reasonable because items with higher values and lower resource requirements (which causes the gradients to be negative) are always preferable for the maximization of total values. Fig. 3 shows two convex hulls. The dotted lines show the efficient convex hull frontier. The efficient convex hull frontier of Group 1 has some segments with negative gradients, which give more utility with less resource. On the other hand, the efficient convex hull frontier for Group 2 follows the monotone feasibility property. Every segment on this convex hull frontier has a positive and non-increasing gradient. The use of angles instead of gradient removes all the problems of picking items in the generalized MMKP. The next subsection presents the algorithm for solving the MMKP using the convex hulls.

3.1. *Heuristic algorithm for solving the MMKP using convex hull approach (C-HEU)*

Below, we present the heuristic algorithm for solving MMKP using the convex hull approach. The pseudo-code of the devised algorithm presented below is preceded by the definitions of some variables used in the algorithm.

current_sol, *saved_sol*: The solution vector containing the indices of the selected items.
snf: The Boolean variable (*snf* indicates “solution not found”) indicating whether *current_sol* is feasible or not.
penalty: The transformation vector to transform multidimensional resource to single dimension (see Remark 1).

Feasible( ): A function returning true if the solution under consideration does not violate any resource constraints and false otherwise.
Utility( ): A function returning the total value earned from the selected items.
initial_penalty( ): Returns the initial penalty vector (see Remark 2).
adjust_penalty( ): Returns the penalty vector based on the current resource usage (see Remark 2).
list_of_frontier_segments, ch_frontier: Lists of the frontier segments. A segment is a vector with two items representing a straight line.
p_1, p_2: Items representing points in two-dimensional space. The associated value represents the y co-ordinate and the transformed single-dimensional resource represents the x co-ordinate.
current_group: Group of a currently selected point.
current_item: Currently selected item of a group.
p_item: Item denoted by point p.
rsum: Vector summation of resource vectors.
Begin Procedure adjust_selected_item (p)
/*This procedure selects a new item of a group that contains the item corresponding to p. The procedure ignores point p if the resource consumption would become infeasible from a feasible solution by selecting this item. However, in the case when no solution is found yet, it selects the item anyway, in the hope to find a feasible solution in the future. */

1. *current_group*← the group that contains the item corresponding to p.
2. *current_item*←the currently selected item of group *current_group*.
3. *p_item*←item of group *current_group* denoted by point p.
4. $fp \leftarrow$ feasibility of the resource consumption after selecting the item denoted by p.
5. *if* $(fp = true)$ *then*
6. *snf* ← *false* //solution found
7. change the selection of group *current_group* from *current_item* to *p_item*
8. update *current_sol*
9. *else if* $(fp = false$ *and snf* $= true)$
10. change the selection of group *current_group* from *current _item* to *p_item*
11. update *current_sol*
12. //when $fp = false$, solution not found but trying to find a feasible solution.
13. *endif // otherwise leave the feasible solution as it is.*

End Procedure
Begin Procedure initial_penalty( )
/*Calculate the initial penalty vector using an appropriate formula*/

1. *rsum* ←vector summation of resource vectors of each item in each group.
2. $q \leftarrow$ apply appropriate formula (see Remark 2 below) on vector *rsum* and total resource vector.
3. *return q*.

End Procedure
Begin Procedure adjust_penalty(q)
/*Updates the penalty vector using the information about the available residual resources*/

1. $q' \leftarrow$ apply appropriate formula (see Remark 2 below) on vector q, total resource vector and available resource vector.
2. *return* q'*;*

end procedure

Begin Procedure C_HEU ( )
1. *current_sol*← The item with lowest value from each group;
2. *if feasible*(*current_sol*) =*false then*
3. *snf*←*true* //Solution not yet found
4. *endif*
5. *penalty* = *initial_penalty*( )
6. *for repeat* ← 1 to 3 *do* //only three iterations for finding solution
7. *saved_sol* ← *current_sol* //saving the current solution
8. *u* ← *Utility*(*current_sol*) //saving utility
9. *for* each group in the MMKP *do*
10. Transform each resource consumption vector of each item to single dimension using vector *penalty* (see Remark 1 below).
11. *ch_frontier* ← efficient convex hull frontier of the items of the group
12. *list_of_frontier_segments* ← *list_of_frontier_segments* + *ch_frontier*
13. *endfor*
14. Sort the segments of *list_of_frontier_segments* in descending order according to the angle of each segment
15. *for* each segment in the *list_of_frontier_segments do*
16. p_1, p_2 ← The items associated with the segment.
17. *adjust_selected_item* (p_1)
18. *adjust_selected_item* (p_2)
19. *end for*.
20. *if Utility*(*current_sol*) < *u then* // New solution is inferior than the saved one
21. *current_sol*←*saved_sol*
22. *endif*
23. *penalty*← *adjust_penalty*(*penalty*) //adjust penalty for the next iteration
24. *end for*
25. *if snf*= *true then*
26. *Solution Not found*
27. *else*
28. *current_sol* is the final solution.
29. *endif*

end Procedure.

Remark 1. The vector *penalty*, as noted above, is used as a transformation vector to transform from multidimensional resource to single dimension. An explanation of this transformation is in order. The vector *penalty* is used to give a "price" to each resource combination. Specifically, let $\vec{q} = (q_1, q_2, \ldots, q_m)$ be the penalty vector; then the penalized resource vector may have the form $\vec{R}_P = (R_1.q_1, R_2.q_2, \ldots, R_m.q_m)$. And finally the transformation to single dimension can use the following form: $R^* = \|\vec{R}_P\| = \sqrt{R_{P_1}^2 + R_{P_2}^2 + \ldots + R_{P_m}^2}$.

Remark 2. The responsibility of the procedure *initial_penalty*( ) is to calculate the initial penalty. On the other hand, the procedure *adjust_penalty*( ) updates the penalty vector using information about the residual resources from the previous iteration. Many different formulas can be used in these two procedures. In [17],

the following two formulas are used, respectively, in procedure *initial_penalty*( ) and *adjust_penalty*( ). Let R and R' denote, respectively, the total and available (residual) resource vector. Then the formula for calculating the kth component for *initial_penalty*( ) can have the following form: $q_k = (rsum_k/R_k) + 1$. On the other hand, a possible formula for *adjust_penalty*( ) would be:

$$q_k = \frac{R_k}{R'_k + R_k} * q_k + 1.$$

Recall that *rsum* and q are defined in the pseudo-code. It may be noted here that in the analysis of these two procedures, as performed in the following section, these definitions of the formulas are assumed to be adopted.

Remark 3. The sorting in line #14 requires some explanation for better implementation. The time required by line #14 as listed in the algorithm is $O(nl \lg nl)$. However, by clever implementation it can be reduced to $O(nl \lg n)$. The modification will be as follows. First, remember that the usual two-dimensional convex hull algorithms can output the frontier segments as a sorted list. So, in Line #12 we do not concatenate these lists, but we just store these lists independently. After Line #13, we have n sorted lists, each of which has at most l elements. Therefore, the task is the merging problem: "given n sorted lists with l elements, we want to merge them into a single sorted list". This can be done in $O(nl \lg n)$ with the use of a heap. For the sake of clarity, the above implementation is avoided in the algorithm listed.

Remark 4. In Procedure *C-HEU*( ) we repeat the *for* loop in Lines 6–24 only 3 times. In a few cases, more than three iterations might give us a better solution. Experimental results show that in most of the cases solution value does not improve after 3 iterations.

Remark 5. It is worth mentioning here that the algorithm *C-HEU* can be used with minor modifications to solve another variant of KP, which is almost similar to MMKP. This variant, as indicated in Section 2, differs from the MMKP in that the restriction in MMKP of picking exactly one item from each group is relaxed here. In order to solve this variant we artificially include in each group a "null" item, which consumes no resource and gives no value. It is clear that the initial solution set will be comprised of these artificial items from each group and if in the final solution one of them still remain present, it will mean that no item should be taken from that group.

3.2. Computational complexity and lower bound

The worst-case complexity of finding the solution of an MMKP using C-HEU can be obtained as follows. Assume that there are n groups each having l items (in case of different number of items per group, assume that l is the maximum number of elements in a group) and let m be the dimension of resource consumption vector. It is easy to verify that the procedure *initial_penalty* and *adjust_penalty* takes, respectively, $O(nl + m)$ and $O(m)$ operations. The time complexity of procedure *adjust_selected_item* is determined by the number of operations required by the feasibility check at Line #4 of that procedure, which can be done in $O(m)$ (the dimension of resource vector is m) as follows. We keep track of the current resource consumption and just deduct the current item's (item denoted by p) resource. Note carefully that the determination of the group in Line #1 can be done in constant time by keeping the group info into the data structure representing the segments. Now we are ready to analyze our main procedure

C-HEU. Note that there are a constant number of iterations in the *for* loop at Lines #6–#24 and hence can be ignored from the analysis. The function *Utility*( ) in Line #8 requires $O(n)$ running time. The *for* loop at Lines #9–#13 iterates n times. Since there are at most l items per group, Lines #10 and #11 take, respectively, $O(lm)$ (see Remark 1 above) and $O(l \lg l)$ (see [18] for convex hull algorithms). So the total time required by this loop is $O(nlm + nl \lg l)$. The sorting at Line #14 can be performed in $O(nl \lg n)$ as explained in Remark 3 above. The following *for* loop executes for $O(nl)$ times. Since the operations required by the procedure *adjust_selected_item* is $O(m)$ (as deduced above), the total running time of this for loop is $O(nlm)$. So the overall running time of the procedure C-HEU can be deduced as follows:

$$O(nl + m) + O(n) + O(nlm + nl \lg l) + O(nl \lg n) + O(nlm) + O(m) = O(nlm + nl \lg l + nl \lg n).$$

The lower bound of the achieved total value for the Single Resource Multiple Dimension (SRMD) System [17] using the approximation algorithm by constructing convex hulls is $V_{\text{lower}} = (U_{\text{optimal}} - p_{\max})$, where $p_{\max}$ is the change of total value for any upgrade, i.e., the maximum difference of values of any two items of any group, and U_{optimal} is the optimal total value achieved by using the exact algorithms. Please refer to [17,19,20] for the proof of this lower bound. The problem solved by this approximation algorithm is actually an MCKP, where the resource dimension is single and this is a special case of MMKP, where the resource dimension might be multiple. Our heuristic algorithm uses the same approach by constructing the convex hull and we can expect the same lower bound if it is executed for an MCKP. Although this is not the lower bound of achieved total value for an MMKP, we can easily speculate on the behaviour of the achieved optimality for an MMKP using the convex hull approach, as MMKP is the generalized variant of MCKP.

The algorithm will definitely show better results for a larger problem size, i.e., if the number of groups and total amount of resources are increased. U_{optimal} increases with a greater number of groups as we get more selected items. Since $p_{\max}$ remains the same and depends on the characteristics of the distribution of the items, the ratio $V_{\text{lower}}/U_{\text{optimal}}$ will increase.

4. Experimental results

In order to study the run-time performance of C-HEU, we implemented C-HEU along with four other algorithms

(1) $V_{\text{opt\_est}}$, an *estimate of the optimal solution* based on a branch-and-bound search using linear programming. This is actually the first iteration of BBLP and this estimate must be higher than the optimal solution. Only one iteration using linear programming determines this estimate, whereas an indefinite number of iterations determines the optimal solution. The subsequent iterations generate estimates closer to the optimal total solution value.
(2) *Moser's heuristic*, based on Lagrange relaxation.
(3) *M-HEU*, a heuristic based on the aggregate resource consumption.

(4) *Greedy approach*, based on a linear search of all the items of each group. It picks the highest-valued feasible item from each group. In this paper, we call this heuristic G-HEU.

We implemented all the algorithms using the Visual C++ programming language. For simplicity of implementation, we assumed that each group has the same number of items, i.e., $l_1 = l_2 = \cdots = l_n$. We used the Simplex Method code from [10] for linear programming. We ran the algorithms on a Pentium III IBM Think Pad 700 MHz with 192 MB of RAM running Windows 2000. Two categories of data sets were used in the performance study of C-HEU, namely, randomly generated and correlated data sets (see [21] for benchmark data sets on the MMKP). For each set of parameters n, l and m, we generated 10 MMKP instances in which the values are correlated with resource requirements and 10 random MMKP instances in which the values are not correlated with resource requirements. We ran the algorithms on all 10 instances, and plotted the averages of solution-value and execution time.

4.1. Test pattern generation

The data sets for testing the performance of different heuristics were initialized as follows:

R_c: maximum amount of a resource consumption by an item.
P_c: maximum cost per unit resource.
R_i: total amount of the ith resource—$n \times R_c \times 0.5$. Here we assume $R_c \times 0.5$ amount resource for each item.
P_i: cost of the ith resource—$Random(P_c)$—a uniform discrete random number from 0 to $(P_c - 1)$.
r_{ijk}: kth resource of the jth item of the ith group—$Random(R_c)$. As the total resource for each item is $R_c \times 0.5$, we can say approximately 50% solutions are infeasible because there is a chance that items with resource consumption between $R_c \times 0.5$ to R_c may be infeasible.
v_{ij}: value of the jth item of the ith group—$Random(m \times (R_c/2) \times (P_c/2)) \times (j+1)/l$, when the item values are not correlated with the resource requirement.
v_{ij}: $\sum r_{ijk} \times P_k + Random(m \times 3 \times (R_c/10) \times (P_c/10))$, when there is a linear correlation between the resource consumption and item values.

4.2. Test results

The graphs of Figs. 4–6 compare the optimality achieved by the different heuristics with the increase in number of groups, number of items in each group and number of resource dimensions. For the experiments done with larger data sets, the computation times for the optimal solution by BBLP are too large for practical interest, as it takes too long (days or years) on the average to do the computation. So the solution values of the heuristics have been normalized by the *estimated optimum total value*, which is calculated as follows. The branch and bound algorithm for the MMKP involves the iterative generation of a *search tree*. A *node* of the tree is expanded by selecting an item of a particular group, called *branching group*. At a node, if the items of a group are not selected then the group is called *free group*. Initially, there is only one node in the tree where all the groups are free. Applying linear programming technique on the free groups of a node, we can determine an *estimate of optimum total value* as well as the branching group of a node. The use of linear programming to determine the branching group reduces the time requirement in the average case. In each iteration the node with the highest upper bound is explored. The nodes, which do not give any solution value using linear programming, are considered as infeasible. These nodes are

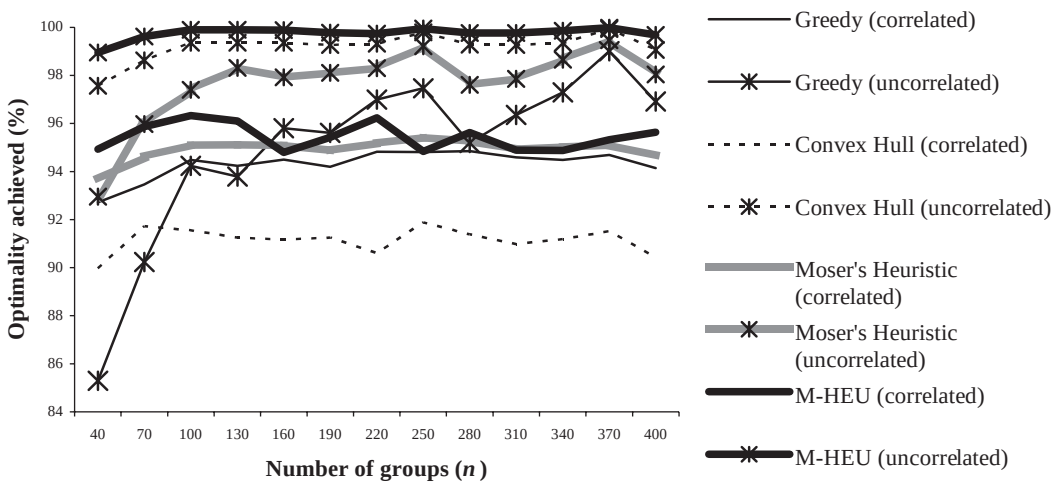

Fig. 4. Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with $l = 10$ and $m = 10$.

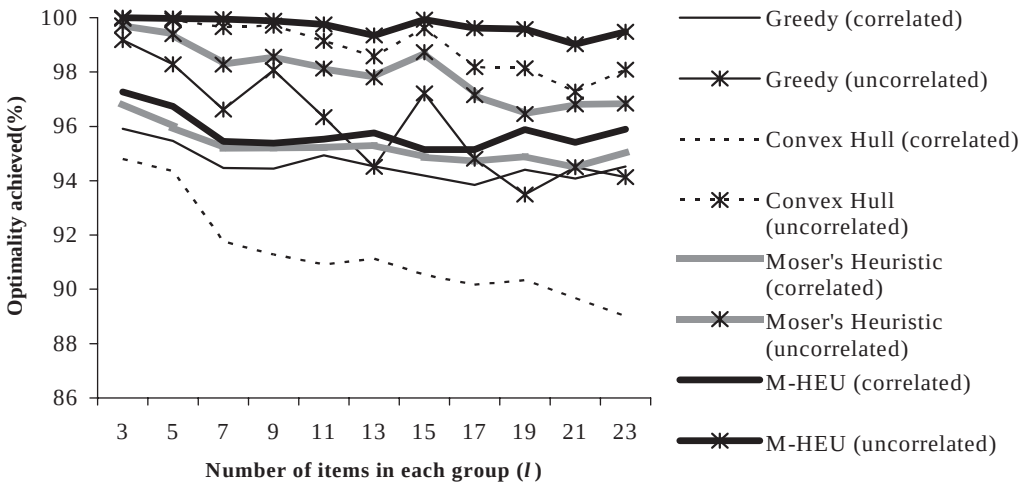

Fig. 5. Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with $n = 200$ and $m = 10$.

deleted from the tree. A solution is found when a node without any free group has the maximum estimated total value.

The graphs of Figs. 7–9 compare the time required by different heuristics. We have not plotted the time required by G-HEU as it takes a very insignificant time for this range of data sets. All the plotted data in the above-mentioned graphs are the average of 10 problem sets.

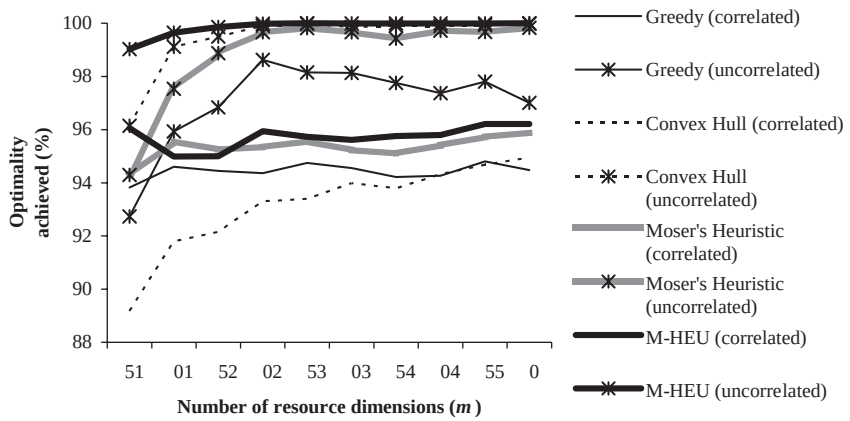

Fig. 6. Performance of different heuristics normalized with the estimated optimal total value for the MMKP data sets with $n = 200$ and $l = 10$.

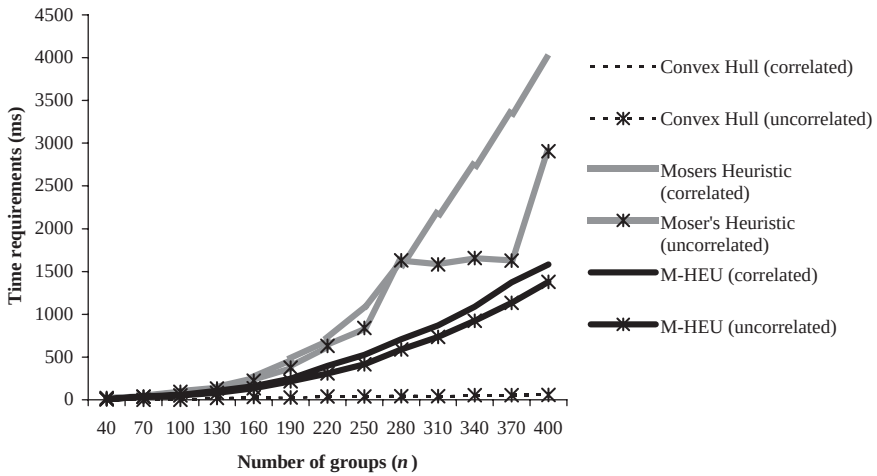

Fig. 7. Time required by different heuristics for the MMKP data sets with $m = 10$ and $l = 10$.

5. Observations

One can make the following observations from the presented tables and figures.

- The heuristics produce solutions that are close to the optimal solutions provided by the algorithm BBLP. M-HEU produces the solution nearest to the optimal solution among all the heuristics.
- We find from Figs. 4–6 that M-HEU, Moser's heuristic and C-HEU give better results for uncorrelated data sets than correlated data sets. C-HEU gives better results than Moser's heuristic for uncorrelated data sets. This is remarkable as C-HEU with $O(nlm + nl\,1g\,l + nl\,1g\,n)$ is giving better results than Moser's heuristic with $O(mn^2(l-1)^2)$.

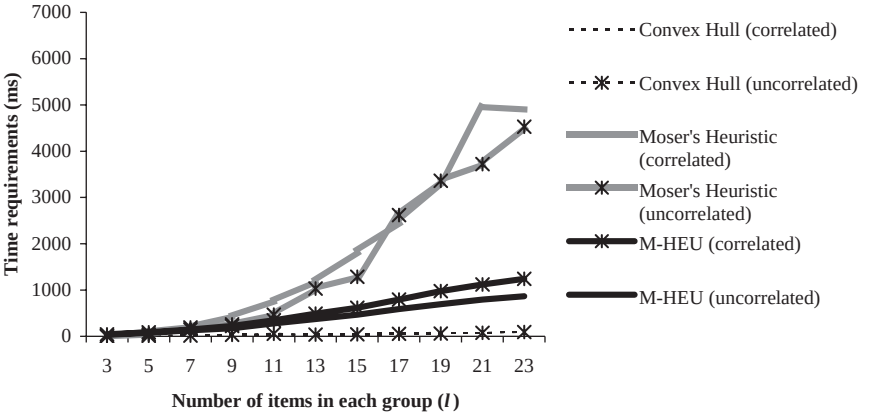

Fig. 8. Time required by different heuristics for the MMKP data sets with $n = 200$ and $m = 10$.

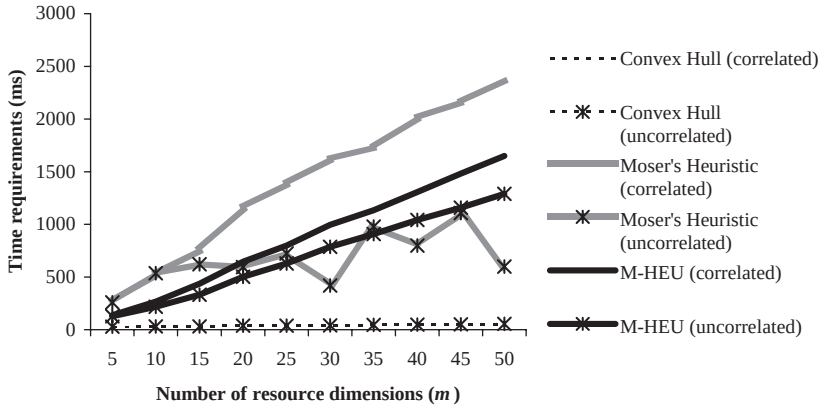

Fig. 9. Time required by different heuristics for the MMKP data sets with $n = 200$ and $l = 10$.

- We also find from Figs. 7–9 that for correlated data sets, all algorithms take more time than uncorrelated data sets. Please see [12] for a plausibility argument of the behavioural differences between correlated and uncorrelated data sets in solving the MMKP.
- In M-HEU, Moser's heuristic and C-HEU, the most profitable items with large values and a small resource requirement are given priority for picking, so we obtain the same behaviour for these heuristics.
- Fig. 4 shows that for a smaller problem set the optimality achieved by different heuristics increases with the increase in problem set size. But for larger problem sets the optimality remains almost stable. We find almost the same trend for an increase in the number of resource dimensions in Fig. 6.
- Fig. 5 shows that the achieved optimality decreases with an increase in the number of items in each group. This degradation is the worst for C-HEU. This is likely because we completely ignore some items from the search space of each group by constructing convex hulls. In other heuristics we ignore some items while picking items but not at all.

- Figs. 7 and 8 show that the time requirements of M-HEU and Moser's heuristic increase quadratically with an increase in group size (number of groups) and the number of items in each group. The time requirement for C-HEU is much less than for the heuristics of quadratic complexity such as M-HEU and Moser's heuristic, implying that it is not quadratic. Recall that the theoretical analysis shows that the worst-case running time C-HEU is $O(nlm + nl\ 1g\, l + nl\ 1g\, n)$. Fig. 9 shows how the time requirements increase almost linearly with an increase in the number resource dimensions of the MMKP.
- We find some irregularities in the data for computation time. As the computation time required by the heuristic and BBLP depends on the contents of the data set, it may happen that smaller data sets take longer than larger data sets. We find such irregularities in Fig. 7 ($n = 280$–370, uncorrelated data sets) and Fig. 9 ($m = 25$–50, uncorrelated data sets) for Moser's heuristics. Similar irregularities were observed and reported by Khan et al. in [3].

6. Conclusion

C-HEU is a heuristic with $O(nlm + nl\ 1g\, l + nl\ 1g\, n)$ complexity to solve the MMKP. The experimental data show that the optimality achieved by this heuristic lies between 88% and 98%. This heuristic is definitely more scalable than other heuristics with quadratic complexity. It also achieves better optimality than some heuristic with quadratic complexity for the problem sets where the values associated with the items are not correlated with the resource consumption. It can be applied successfully for the admission controllers for multimedia systems that require quicker response time than M-HEU or Moser's heuristic. C-HEU is especially applicable for the systems where the requested QoS levels are not proportional to the resource requirement to serve those QoS levels.

Acknowledgements

The authors would like to express their gratitude to the anonymous referees for their helpful comments and suggestions.

References

[1] Magazine M, Oguz O. A heuristic algorithm for multidimensional zero-one knapsack problem. European Journal of Operational Research 1984;16(3):319–26.
[2] Khan S, Li KF, Manning EG. Padma: an architecture for adaptive multimedia systems. IEEE Pacific rim conference on communications, computers and signal processing, August 20–22, 1997. p. 105–8.
[3] Khan S. Quality adaptation in a multi-session adaptive multimedia system: model and architecture. PhD dissertation. Department of Electrical and Computer Engineering, University of Victoria, 1998.
[4] Martello S, Toth P. Algorithms for knapsack problems. Annals of Discrete Mathematics 1987;31:70–9.
[5] Koleser P. A branch and bound algorithm for knapsack problem. Management Science 1967;13:723–35.
[6] Shih W. A branch and bound method for multiconstraint knapsack problem. Journal of the Operational Research Society 1979;30:369–78.
[7] Nauss R. The 0-1 knapsack problem with multiple choice constraints. European Journal of Operation Research 1978;2: 125–31.

[8] Akbar MM. Distributed utility model applied to optimal admission control and QoS adaptation in distributed multimedia server system and enterprise networks. PhD dissertation. Department of Computer Science, University of Victoria, 2002.
[9] Martello S, Pisinger D, Toth P. New trends in exact algorithms for the 0-1 knapsack problem. European Journal of Operational Research 2000;123:325–32.
[10] Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical recipes in C: the art of scientific computing. 2nd ed, Cambridge, UK: Cambridge University Press; 1992.
[11] Toyoda Y. A simplified algorithm for obtaining approximate solution to zero-one programming problems. Management Science 1975;21:1417–27.
[12] Khan S, Li KF, Manning EG, Akbar MM. Solving the knapsack problem for adaptive multimedia system. Studia Informatica Universalis 2002;2:161–82.
[13] Moser M, Jokanovic DP, Shiratori N. An algorithm for the multidimensional multiple-choice knapsack problem. IEICE Transactions on Fundamentals of Electronics 1997;80(3):582–9.
[14] Dammeyer F, Voss S. Dynamic tabu list management using the reverse elimination method. Annals of Operations Research 1991.
[15] Drexel A. A simulated annealing approach to the multiconstraint zero-one knapsack problem. Annals of Computing 1988;40:1–8.
[16] Khuri S, Back T, Heitkotter J. The zero/one multiple knapsack problem and genetic algorithms. ACM Symposium of applied computation; 1994.
[17] Lee C. On QoS management. PhD dissertation. School of Computer Science, Carnegie Mellon University, August 1999.
[18] Cormen TH, Leiserson CE, Rivest RL. Introduction to algorithms. Cambridge, MA/New York: MIT Press, McGraw-Hill; 1990.
[19] Lee C, Lehoczky J, Rajkumar R, Siewiorek D. On quality of service optimization with discrete qos options. Proceedings of the IEEE real-time technology and applications symposium. IEEE, June 1998.
[20] Lee C, Siewiorek D. An approach for quality of service management. Technical Report CMU-CS-98-165, Computer Science Department, Carnegie Mellon University, October 1998.
[21] ftp://panoramix.univ-paris1.fr/pub/CERMSEM/hifi/MMKP.