Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem

Md Mostofa Akbar[1, 2], Eric G. Manning[3], Gholamali C. Shoja[2], Shahadat Khan[4]

[2] Department of CS, PANDA Lab, UVic, Victoria, BC, Canada
{mostofa,gshoja}@csc.uvic.ca
[3]Department of CS and ECE,PANDA Lab, UVic, Victoria, BC, Canada
Eric.Manning@engr.UVic.ca
[4] Eyeball.com, Suite 409-100 Park Royal, West Vancouver, B.C. Canada
shahadat@eyeball.com

Abstract.

The Multiple-Choice Multi-Dimension Knapsack Problem (MMKP) is a variant of the 0-1 Knapsack Problem, an NP-Hard problem. Hence algorithms for finding the exact solution of MMKP are not suitable for application in real time decision-making applications, like quality adaptation and admission control of an interactive multimedia system. This paper presents two new heuristic algorithms, M-HEU and I-HEU for solving MMKP. Experimental results suggest that M-HEU finds 96% optimal solutions on average with much reduced computational complexity and performs favorably relative to other heuristic algorithms for MMKP. The scalability property of I-HEU makes this heuristic a strong candidate for use in real time applications.

1 Introduction

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. 0-1 classical 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.

Let there be *n items* with *values* $v_1,v_2,\ldots,v_n$ and the corresponding resources required to pick the items are $r_1,r_2,\ldots,r_n$ respectively. The items can represent *services* and their associated values can be *utility* or *revenue* earned from that service. In mathematical notation, the 0-1 knapsack problem is to find V= maximize $\sum_{i=1}^{n} x_i v_i$, where $\sum_{i=1}^{n} x_i r_i \le R$ and $x_i \in \{0,1\}$.

The MMKP is a variant of the 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

[1] Supported by grants from Canadian Commonwealth Scholarship Program.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 659–668, 2001.

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 j th item of the i th group, $\vec{r}_{ij} = (r_{ij1}, r_{ij2}, \cdots, r_{ijm})$ be the required resource vector for the j th item of the i th group and $\vec{R} = (R_1, R_2, \cdots, R_m)$, be the resource bound of the knapsack. Then the problem is to find V = maximize $\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij}$, so that, $\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} r_{ijk} \le R_k, k = 1, 2, \ldots, m$ and $\sum_{j=1}^{l_i} x_{ij} = 1$, $x_{ij} \in \{0,1\}$, the picking variables. Here V is called the *value of the solution*.

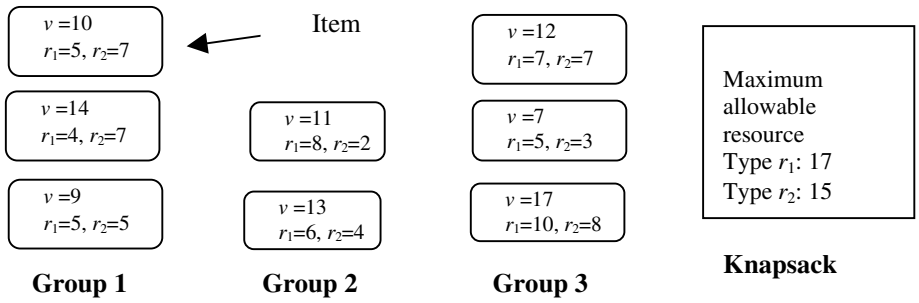

Fig. 1. Knapsack Problem.

Fig. 1 illustrates the 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 achieve maximum total value of the picked items subject to the resource constraint of the knapsack, that $\sum(r_1 \text{ of picked items}) \le 17$ and $\sum(r_2 \text{ of picked items}) \le 15$.

Many practical problems in resource management can be mapped to the MMKP. The Utility Model for adaptive multimedia systems proposed in [4, 5] is actually an MMKP. Users submitting their requests to a multimedia system can be represented by the groups of items. Each level of *QoS* (*Quality of Service*) of a user's requested session is equivalent to an item; each session is equivalent to a group of items. The values of the items are equivalent to offered prices for the session. The multimedia server is equivalent to the knapsack with limited resources, e.g. CPU cycles, I/O bandwidth and memory. The proposed exact solutions in [1, 2] are so computationally expensive that they are not feasible to apply in real time applications such as multimedia systems. Hence heuristic or approximate algorithms for solving the MMKP have an important role in solving real time problems.

In this paper, we present two heuristic algorithms for solving the MMKP, which are suitable for application in real time multimedia problems. Some related work on KP and MMKP will be described in section 2. In section 3 the heuristic algorithms M-HEU and I-HEU will be presented with complexity analysis. Some experimental results showing comparisons with other methods of solving the MMKP will be given in section 4. Section 5 concludes the paper, mentioning some of the possible applications of this algorithm in real time applications.

2 Related Work

There are different algorithms for solving variants of knapsack problems [8]. Actually MMKP is the combination of MDKP and MCKP. The Multi-Dimension 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 criterion for items is restricted. In this variant of KP there are one or more groups of items. Exactly one item will be picked from each group.

There are two methods of finding solutions for an MMKP: one is a method for finding exact solutions and the other is heuristic. Finding exact solutions is NP hard. Using the branch and bound with linear programming (BBLP) technique, Kolesar, Shih, Nauss and Khan presented exact algorithms for 0-1 KP, MDKP, MCKP and MMKP respectively [2, 4, 5, 10, 12]. Although the use of linear programming to determine the feasibility of picking any item of any group reduces the time requirement in the average case, it is not feasible to apply in all practical cases.

A greedy approach has been proposed [5, 8, 13] to find near optimal solutions of knapsack problems. For a 0-1 KP as described in section 1, items are picked from the top of a list sorted in descending order on v_i / r_i [8]. To apply the greedy method to the MDKP Toyoda proposed a new measurement called *aggregate resource consumption* [13]. Khan [4] has applied the concept of aggregate resource consumption to pick a new item in a group to solve the MMKP. His heuristic HEU selects the lowest-valued items by utility or revenue of each group as initial solution. It then upgrades the solution by choosing a new item from a group which has the highest positive Δa_{ij}, the change in aggregate consumed resource (the item which gives the best revenue with the least aggregate resource). If no such item is found then an item with the highest $(\Delta v_{ij})/(\Delta a_{ij})$ (maximum value gain per unit aggregate resource expended) is chosen. Here,

$$\Delta a_{ij} = \sum_k \left(r_{i\rho[i]k} - r_{ijk} \right) \times C_k \Big/ \left| \vec{C} \right|, \text{ increase in aggregate consumed resource.} \quad \textbf{(1)}$$

r_{ijk}=amount of the kth resource consumption of the jth item of the ith group.
$\rho[i]$= index of selected item from the ith group and C_k= amount of the kth resource consumption.
$\Delta v_{ij} = v_{i\rho[i]} - v_{ij}$, gain in total value.
This heuristic for MMKP provides solutions with utility on average equal to 94% of the optimum, with a worst case time complexity of $O\left(mn^2 (l-1)^2\right)$. Here, n = number of groups, l =number of items in each group (assumed constant for convenience of analysis) and m =resource dimension.

Magazine and Oguz [7] proposed another heuristic based on Lagrange Multipliers to solve the MDKP. Moser's [9] heuristic algorithm also uses the concept of graceful degradation from the most valuable items based on Lagrange Multipliers to solve the MMKP. This algorithm is also suitable for real time application of MMKP problems and it performs better than HEU in terms of optimality when both methods reach a solution [4]. It has been observed from experiments that Moser's method does not

always give a solution when there is a solution obtainable by selecting the least valuable items from each group. HEU does not have that difficulty because it starts from the least valuable items. But, HEU will fail to find a solution when the bottommost elements do not give feasible solution, while some items with higher values but with less resource consumption do.

3 Proposed Heuristic Algorithm for MMKP

3.1 Modified HEU (M-HEU)

We have to sort the items of each group in non-decreasing order according to the value associated with each item. Hence, we can say that in each group the bottom items are *lower-valued items* than the top ones. The items at the top can be defined as *higher-valued items* than those in the bottom. If a particular pick of items (one from each group) does not satisfy the resource constraint, we define the solution *infeasible*. A *feasible solution* is a solution that satisfies the resource constraints. For any resource k, C_k / R_k can be defined as infeasibility factor f_k. The kth resource is feasible if the infeasibility factor $f_k \leq 1$, otherwise it is infeasible.

We find that HEU requires additional steps to find a feasible solution if the lowest quality items from each group represent an infeasible solution. HEU finds a solution by only *upgrading* the selected items of each group. There might be some higher-valued items in the MMKP, which make the solution infeasible, but if some of the groups are downgraded we can get a feasible solution. This method of upgrading followed by downgrading may increase the total value of the solution.

Steps in the Algorithm

Step 1: Finding a feasible solution

Step 1.1: Select the lowest-valued items from each group.

Step 1.2: If the resource constraints are satisfied *then* notify "This MMKP has a solution". So go to step 2, *else* the solution is infeasible.

Step 1.3: Find out the resource k_m, which has the highest infeasibility factor. This is the *most infeasible resource*. Select a high-valued item from any group, which decreases f_{k_m}, does not increase the infeasiblity factor, of the other resources, does not make any feasible resource requirement into an infeasible one, and gives the highest Δa_{ij}, which has been defined by (1).

Step 1.4: If an item is found in *step 1.3 then* go to *step 1.2 else* notify "No Solution Found".

Step 2: Upgrading the selected feasible solution

*/*This step is identical to the iteration performed in HEU*/*

Step 2.1: Find a higher valued item from a group than the selected item of that group subject to the resource constraint which has the highest positive $\Delta a_{ij.}$ If no such item is found then an item with the highest $(\Delta v_{ij})/(\Delta a_{ij})$ is chosen.

Step 2.2: If no such item is found in *step 2.1 then* go to *step 3 else* look for another items in *step 2.1*.

Step 3: Upgrading using one upgrade followed by at least one downgrade

Step 3.1: If there are higher-valued items than the selected item in any group *then* find such a higher-valued item (whatever is the resource constraint) which has the highest value of $(\Delta v_{ij})/(\Delta a' r_{ij})$. Here,

$$\Delta a'_{ij} = \sum_k \frac{r_{i\rho[i]k} - r_{ijk}}{R_k - C_k} = \text{ the ratio of increased resource requirement to available resource, where, } R_k = k\text{th resource constraint.} \quad (2)$$

Step 3.2: Find a lower-valued item than the selected item of the groups such that the downgrade still gives higher total value than the total value achieved in Step 2 and has the highest value of $(\Delta a''_{ij})/\Delta v_{ij}$

$$\Delta a''_{ij} = \sum_k \frac{r_{i\rho[i]k} - r_{ijk}}{C_k - R_k} = \text{ the ratio of decreased resource requirement to overconsumed resource.} \quad (3)$$

Step 3.3: If an item is found in *step 3.2 then*

If the item satisfies the resource constraint *then* look for a better solution in *step 2 else* go to *step 3.2* for another downgrade.

Else

Revive the solution we found at the end of *step 2* and terminate.

Endif

3.2 Incremental Solution of the MMKP (I-HEU)

If the number of groups in the MMKP is very large then it is not efficient to run M-HEU once per minute, as a real time system, for example a multimedia system with 10000 sessions might well require. An *incremental* solution is a necessity to achieve better computation speed. By changing the step of finding a feasible solution (Step 1) we can use M-HEU to solve the MMKP *incrementally*, starting from an already solved MMKP with a smaller number of groups. The changed sub-steps in *step 1* are as follows.

Step 1.1: Select the lowest-valued item (according to value) from each new group.

Step 1.3: This step is similar to the *step 1.3* of M-HEU except that here we have to find any item instead of "*Higher-valued Item*".

I-HEU does the feasibility test in step 1 with the lowest-valued items. I-HEU will require pretty much the same time if it finds the near optimal solution after doing a lot of upgrading and downgrading in the older groups as well as new groups. It will give the best performance when the solution is determined by upgrading or downgrading only the new groups of items. Typically, it is unlikely that it will require abrupt changes in the already solved groups while running step 2 and 3 of I-HEU. Thus we can expect some time advantage with this incremental method over M-HEU.

3.3 Analysis of the Algorithm

Non Regenerative Property: The three steps of M-HEU never regenerate a solution that has been found previously. The straightforward reasons for this convergence property are as follows:

- *Step 1* never makes any feasible resource requirement infeasible or infeasible resource requirement more infeasible.
- *Step 2* always upgrades the solution vector with increased total value.
- *Step 3* upgrades one item followed by downgrading one or more items for an increase in total value, thereby excluding the possibility of regenerating a previously determined solution or of infinite looping.

Complexity of the Algorithm: The computational complexity of step 1 will be the worst when there is no feasible solution or there is a feasible solution located at the highest valued item of each group and at each iteration only one item of one group moves one level up. Initially the selected items of each group is the lowest-valued item of each group. So the total upgrade in group i is ($l_i - 1$). The total number of upgrades in step 1 is $\sum_{i=1}^{n}(l-1)$. For convenience of analysis we assume that all groups contain the same number of items, i.e., l_i=l. So step 1.3 and step 1.4 will be executed $(nl-n)$ times each.

Total floating point operations in step 1 is $\sum_{j=0}^{(l-1)n-1}\{(nl-n-j)\times(6m+1)+2m\}$

$$\text{Complexity in step 1}, T_1 = 3n^2(l-1)^2 m + \frac{n^2(l-1)^2}{2} + 5n(l-1)m + \frac{n(l-1)}{2} \quad \textbf{(4)}$$

Step 2 requires the highest computation when there is a feasible solution located at the bottom most item of each group (initial solution), step 2.1 upgrades one level of one group in every execution and upgrading continues until it reaches the highest valued item of each group. So the analysis of step 2 is like step 1.

$$\text{Complexity in step 2}, T_2 = 3n^2(l-1)^2 m + 2n^2(l-1)^2 + 3n(l-1)m + 2n(l-1) \quad \textbf{(5)}$$

We present an upper bound for the computational complexity of step 3. The situation will be the worst when an upgrade is done by step 3.1 and it is followed by all possible down grades by step 3.2 (called from step 3.3). Then it jumps to step 2 to do all possible (n-1)l upgrades before going to step 3 again.

Upper bounds of the computational complexity by one upgrade in step 3.1 and all downgrades in step 3.2 are $n(l-1)\times(4m+1)$ and $n(l-1)\times\{(4m+5)\times(l-1)n+2m\}$ respectively.

$$\text{Upper bound in step 3}, T_3 = n^3(l-1)^3(7m+7) + n^2(l-1)^2(9m+2) \quad \textbf{(6)}$$

Steps 1and 2 share the job of upgrading. So the combined worst case complexity will be expressed by (5) i.e., $O(mn^2(l-1)^2)$. *Step 3* is executed when *step* 2 fails to upgrade the solution. The available resource is very small in this step compared to *step* 2. We expect that this step require less iteration than the previous steps and we can improve the solution with less effort. This is analogous to the hill climbing approach in a plateau, for classical AI problems [14]. Please refer to [16] for a detailed analysis.

4 Experimental Results

4.1 Test Pattern Generation

The knapsack problem has been initialized by the following pseudo random numbers: r_{ijk} = kth weight of jth item of ith group = random(R_c). Value per unit resource p_k = random (P_c). Value for each item $v_{ij} = \sum_k r_{ijk} p_k + \text{random}(V_c)$. Here, R_c, P_c and V_c are the upper bound of any kind of resource requirement, unit price of any resource and the extra value of an item after its consumed resource price. The value of each item is not directly proportional to the resource consumption. The function random(i) gives an integer from 0 to (i-1), following the uniform distribution.

The total constraint for kth resource type $R_k = R_c \times n \times 0.5$, where n = number of groups. If we want to generate a problem for which the solution is known, the following reinitializations have to be done.

ρ_i = Selected item of ith group = random(C_i), C_i= number of items in ith group.

$R_k = \sum_i r_{i\rho_i k}$, i.e., exactly equal to the sum of the resources of the selected items.

The values associated with the selected items are $v_{i\rho_i} = \sum_k r_{i\rho_i k} \times p_k + V_c$.

This initialization ensures maximum value per unit resource for the selected items. Furthermore the total resource constraint is exactly the same as the aggregate of selected item resources. Hence there is no chance of selecting other items for maximization of total value.

4.2 Test Results

We implemented BBLP, Moser's heuristic, M-HEU and I-HEU for solving MMKP, using the C programming language. We tested the programs in a Pentium dual processor 200 MHZ PC with 32 MB RAM running Linux OS (RedHat Package 5.0).

Table 1 shows a comparison among BBLP, Moser's heuristic and our M-HEU. Here, 10 sets of data have been generated randomly for each of the parameters n, l and m. We used the constants R_c=10, P_c=10 and V_c=20 for generation of test cases. Data is not initialized for a predefined maximum total value for the selected items. The column BBLP gives the average value earned from BBLP. The columns *Moser* and *M-HEU* give the average standardized value earned in those heuristics with respect to exact solution, where solutions were found. The column $\overline{b}, \overline{m}, \overline{h}$ shows the number of data sets where BBLP, Moser and M-HEU *fail* to find the solution. We find that Moser's algorithm cannot always find a feasible solution when there is a solution found by M-HEU. In Table 1 row 1, 2, 3 and 7 shows that Moser's method failed to find a solution when the algorithms could. ∂_m and ∂_h are the standard deviation of standardized total value achieved in the 10 sets of data, given to indicate stability. The main observation from this table is the time requirement of the heuristic solutions compared with BBLP. This time requirement of BBLP increases dramatically with

the size of the MMKP, because of the exponential computational complexity of BBLP.

It is impractical to test the performance of BBLP for larger MMKP. In order to determine the percentage of optimality (standardized value earned with respect to BBLP) achieved by the heuristics we used the technique described in the last subsection "Test Pattern Generation". Now if we look at table 1 and 2 we find that our proposed M-HEU always performs better than Moser's heuristic in finding feasible solutions, and in achieving optimality. From table 2 it is also observed that M-HEU performs better in terms of time requirement for larger problem sets. We can also conclude that the stability of the solution performance is almost the same in both the cases.

Table 3 shows the comparative performances of I-HEU and M-HEU for different *batch* sizes. Each batch contains a particular number of groups. For each set of parameters the program starts with no groups and continues until the number of groups reaches 100, after the arrival of several batches. The result from I-HEU is used to solve the MMKP for the next batch. We run M-HEU and I-HEU separately and the average time requirements per batch are shown in the table. The columns headed by $\overline{h}_m$ and $\overline{h}_i$ show the number of cases where M-HEU and I-HEU could not find feasible solutions respectively. The column I-HEU/M-HEU gives the ratio of total values achieved by two heuristics. We find that the performances of I-HEU and M-HEU are almost the same in achieving optimal solutions. This means the incremental approach in computation does not degrade the solution quality and we get better computational speed as observed from the test data. The main reason for the difference in solution quality is different starting points. M-HEU starts from scratch whereas I-HEU starts from an almost-done solution. That is why we are not getting the same result although we are doing the same thing in *step 2* and *step 3* of the algorithms.

Table 1. Comparison among BBLP, Moser's Heuristic and HEU

row no.	n	l	m	BBLP	Moser	M-HEU	t_{BBLP} (ms)	t_{Moser} (ms)	$t_{M\text{-}HEU}$ (ms)	$\overline{b},\overline{m},\overline{h}$	∂_m	∂_h
1	5	5	5	621.60	0.94	0.97	49	0.57	0.53	0 1 0	0.039	0.032
2	7	5	5	946.10	0.93	0.95	400	1.09	1.06	0 1 0	0.010	0.021
3	7	7	5	964.20	0.93	0.96	1161	2.02	1.88	0 1 0	0.031	0.021
4	9	5	5	1163.40	0.95	0.97	1328	1.49	1.70	0 0 0	0.013	0.016
5	9	7	5	1110.20	0.92	0.96	8298	3.47	2.89	0 0 0	0.029	0.016
6	9	9	5	1135.00	0.93	0.95	13884	5.52	4.19	0 0 0	0.011	0.025
7	11	5	5	1341.10	0.93	0.96	4013	2.37	2.49	0 2 0	0.023	0.014
8	11	7	5	1431.20	0.94	0.96	15436	4.89	4.03	0 0 0	0.013	0.017
9	11	9	5	1394.10	0.93	0.96	38309	8.46	6.37	0 0 0	0.018	0.021
10	13	5	5	1648.50	0.94	0.96	12619	3.32	3.62	0 0 0	0.016	0.011
11	13	7	5	1545.30	0.94	0.95	51466	7.04	5.77	0 0 0	0.013	0.016
12	13	9	5	1387.40	0.93	0.95	55429	10.46	8.02	0 0 0	0.018	0.020
13	15	5	5	1803.10	0.94	0.97	39225	3.99	4.47	0 0 0	0.021	0.016
14	15	7	5	2007.30	0.94	0.95	84015	8.42	7.59	0 0 0	0.007	0.015
15	15	9	5	1766.60	0.93	0.95	139150	14.08	11.21	0 0 0	0.020	0.012
16	17	5	5	1995.40	0.93	0.97	49175	5.27	5.79	0 0 0	0.009	0.010
17	17	7	5	2148.90	0.93	0.95	107793	10.89	9.60	0 0 0	0.018	0.013
18	17	9	5	1811.00	0.93	0.96	199418	17.69	14.46	0 0 0	0.012	0.012
19	19	5	5	2218.50	0.94	0.96	75893	6.41	7.22	0 0 0	0.016	0.008
20	19	7	5	2104.40	0.93	0.96	124633	12.89	11.63	0 0 0	0.009	0.006
21	19	9	5	1880.60	0.92	0.96	270580	21.33	19.15	0 0 0	0.025	0.010
22	21	5	5	2578.50	0.95	0.96	64656	7.47	8.19	0 0 0	0.011	0.019

row no.	n	l	m	BBLP	Moser	M-HEU	t_{BBLP} (ms)	t_{Moser} (ms)	$t_{M\text{-}HEU}$ (ms)	$\bar{b}, \bar{m}, \bar{h}$	∂_m	∂_h
23	21	7	5	2174.60	0.93	0.96	114337	16.28	15.39	0 0 0	0.015	0.016
24	21	9	5	2791.50	0.94	0.95	463615	27.67	22.33	0 0 0	0.010	0.009

Table 2. Comparison between Moser's Heuristic and M-HEU

n	l	m	$\bar{m}$	$\bar{h}$	Max value	*Moser*	*M-HEU*	t_{Moser} (ms)	t_{HEU} (ms)	∂_m	∂_h
100	5	5	0	0	11574.9	0.94	0.96	137.4	134.6	0.0165	0.0077
100	15	5	0	0	9925.1	0.92	0.95	1410.4	806.0	0.0211	0.0133
100	25	5	0	0	10773.5	0.93	0.96	4002.5	1798.2	0.0121	0.0112
100	5	15	2	0	34942.6	0.94	0.95	263.2	248.4	0.0118	0.0095
100	15	15	0	0	34673.2	0.93	0.95	2859.2	1686.2	0.0078	0.0092
100	25	15	0	0	34533.5	0.93	0.95	8705.6	3488.7	0.0099	0.0093
100	5	25	9	0	59607.2	0.94	0.95	325.1	331.0	-	0.0035
100	15	25	1	0	59467.8	0.92	0.93	5026.2	2688.3	0.0101	0.0082
100	25	25	0	0	57338.3	0.92	0.94	14922.8	5715.8	0.0132	0.0083
200	5	5	0	0	22587.8	0.94	0.96	539.2	527.4	0.0104	0.0103
200	15	5	0	0	22360.8	0.94	0.96	5625.3	3404.3	0.0122	0.0095
200	25	5	0	0	22979.8	0.93	0.96	17643.3	8794.0	0.0172	0.0128
200	5	15	1	0	74289.9	0.94	0.95	1123.8	1192.4	0.0058	0.0091
200	15	15	0	0	71805.7	0.94	0.95	13901.0	7858.2	0.0080	0.0075
200	25	15	0	0	70007.3	0.94	0.95	37617.8	16023.4	0.0109	0.0081
200	5	25	6	0	113000.5	0.93	0.95	1747.3	1793.8	0.0069	0.0068
200	15	25	0	0	122391.5	0.93	0.94	22427.8	11907.6	0.0043	0.0075
200	25	25	0	0	116578.0	0.93	0.95	60908.8	23828.6	0.0067	0.0087
300	5	5	0	0	31790.7	0.94	0.95	1198.9	1162.5	0.0171	0.0103
300	15	5	0	0	31687.1	0.93	0.95	14641.0	7853.2	0.0221	0.0104
300	25	5	0	0	31908.5	0.93	0.96	43271.1	16561.2	0.0213	0.0091
300	5	15	0	0	102715.7	0.94	0.95	2981.1	2926.5	0.0048	0.0078
300	15	15	0	0	98948.5	0.93	0.95	31634.6	17635.6	0.0065	0.0081
300	25	15	0	0	97911.6	0.93	0.94	86546.2	36148.7	0.0075	0.0082
300	5	25	0	0	177995.2	0.94	0.96	5183.2	4475.2	0.0078	0.0054
300	15	25	0	0	167864.3	0.93	0.95	51186.4	26481.2	0.0089	0.0069
300	25	25	0	0	169079.8	0.93	0.95	132720.0	53770.6	0.0065	0.0072

Table 3. Comparison of I- HEU and M-HEU

Batch Size	l	m	*I-HEU/ M-HEU*	$t_{M\text{-}HEU}$	$t_{I\text{-}HEU}$	$\bar{h}_m$	$\bar{h}_i$
1	5	5	1.001	44.2	2.4	1	2
1	10	5	0.998	118.1	7.4	0	0
1	15	5	0.999	319.2	7.7	0	0
1	5	10	1.001	90.7	5.1	2	2
1	10	10	1.003	216.5	13.3	0	0
1	15	10	1.004	436.6	19.1	0	0
1	5	15	1.002	81.3	7.5	2	2
1	10	15	1.002	271.2	21.4	1	1
1	15	15	1.000	688.3	27.4	2	2
6	5	5	0.995	48.9	4.2	0	0
6	10	5	1.000	130.3	7.2	0	0
6	15	5	0.999	222.0	11.5	0	0
6	5	10	0.995	74.4	8.1	0	0
6	10	10	0.998	199.8	13.2	0	0

Batch Size	l	m	*I-HEU/ M-HEU*	$t_{M\text{-}HEU}$	$t_{I\text{-}HEU}$	$\bar{h}_m$	$\bar{h}_i$
6	15	10	1.000	300.5	22.4	0	0
6	5	15	0.991	84.1	16.9	0	0
6	10	15	0.995	301.3	17.4	0	1
6	15	15	1.001	500.5	30.7	0	0
11	5	5	0.997	61.3	6.5	0	0
11	10	5	1.001	154.8	15.4	0	0
11	15	5	1.003	248.4	36.4	0	0
11	5	10	1.010	88.5	17.8	0	0
11	10	10	0.997	272.5	34.8	0	0
11	15	10	0.988	507.4	51.9	0	0
11	5	15	0.992	143.6	18.7	0	0
11	10	15	1.003	364.4	36.6	0	0
11	15	15	1.003	594.6	71.7	0	0

5 Concluding Remarks

The new heuristics M-HEU and I-HEU perform better than other algorithms considered here. M-HEU is applicable to real time applications like admission control and QoS adaptation in multimedia systems. We include a dummy QoS level with null resource requirement and zero revenue, which is therefore lower-valued than all other

QoS levels. Now selection of that null QoS level by M-HEU or I-HEU indicates rejection, comprising effect an admission control for the underlying system. On the other hand, the dummy QoS level is always feasible because it does not take any resource. So, there will be no problem regarding infeasible solution in this practical problem. Due to the quadratic complexity we can not claim too much about M-HEU's scalability property. However, I-HEU appears to be very effective indeed, as it offers almost the same result with much less time requirement. It therefore could be used with improved performance in a multimedia server system with thousands of admitted sessions. The QoS manager can execute I-HEU once per minute as some sessions are dropped and some new ones seek admission. M-HEU could be applied occasionally, e.g., once per hour for further improvement of the solution.

Only the worst case analysis of M-HEU and I-HEU has been presented here. The average case analysis of the algorithms is a good research topic for future work. The question of distributed algorithms for solving the MMKP is also an interesting unsolved problem. Parallel and distributed versions of MMKP with better computational complexity can improve the scalability and fault tolerance of adaptive multimedia.

References

1. R. Armstrong, D. Kung, P. Sinha and A. Zoltners. A Computational Study of Multiple Choice Knapsack Algorithm. *ACM Transaction on Mathematical Software*, 9:184-198 (1983).
2. P. Koleser, A Branch and Bound Algorithm for Knapsack Problem. *Management Science*, 13:723-735 (1967).
3. K. Dudziniski and W. Walukiewicz, A Fast Algorithm for the Linear Multiple Choice Knapsack Problem. *Operation Research Letters,* 3:205-209 (1984).
4. S. Khan. *Quality Adaptation in a Multi-Session Adaptive Multimedia System: Model and Architecture*. PhD thesis, Department of Electrical and Computer Engineering, University of Victoria (1998).
5. S. Khan., K. F. Li and E.G. Manning. The Utility Model for Adaptive Multimedia System. In *International Workshop on Multimedia Modeling*, pages 111-126 (1997).
6. M. Magazine, G. Nemhauser and L. Trotter. When the Greedy Solution Solves a Class of Knapsack Problem. *Operations Research*, 23:207-217 (1975)
7. M. Magazine and O. Oguz. A Heuristic Algorithm for Multidimensional Zero-One Knapsack Problem. *European Journal of Operational Research*, 16(3):319-326 (1984).
8. S. Martello and P. Toth. Algorithms for Knapsack Problems. *Annals of Discrete Mathematics*, 31:70-79 (1987).
9. M. Moser, D. P. Jokanovic and N. Shiratori. An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem. *IEICE Transactions on Fundamentals of Electronics*, 80(3):582-589 (1997).
10. R. Nauss. The 0-1 Knapsack Problem with Multiple Choice Constraints. *European Journal of Operation Research*, 2:125-131(1978).
11. W.H. Press, S.A. Teukolsky, W. T. Vetterling and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, second edition (1992).
12. W. Shih. A branch and Bound Method for Multiconstraint Knapsack Problem. *Journal of the Operational Research Society*, 30:369-378 (1979).
13. Y. Toyoda. A Simplified Algorithm for Obtaining Approximate Solution to Zero-one Programming Problems. *Management Science*, 21:1417-1427 (1975)
14. G. F. Luger and W. A. Stubblefield. Artificial Intelligence, *Structures and Strategies for Complex Problem Solving*, Second edition, The Benjamin/Cummings Publishing Company, Inc., 1993.
15. R. K. Watson. *Applying the Utility Model to IP Networks: Optimal Admission & Upgrade of Service Level Agreements*. MASc Thesis, Dept of ECE, University of Victoria, 2001, to appear.
16. M. Akbar, E.G. Manning, G. C. Shoja, S. Khan, "*Heuristics for Solving the Multiple-Choice Multi-Dimension Knapsack Problem*", Technical Report DCS-265-1R, Dept. of CS, UVic, March 2001.