A Distributed Heuristic Solution using Arbitration for the MMMKP

Md. Mostofa Akbar[1], Eric. G. Manning[3], Gholamali C. Shoja[2], Steven Shelford[4]
Tareque Hossain[5]

[1]Department of CSE, BUET, Dhaka, Bangladesh
mostofa@cse.buet.ac.bd

[2]Department of CS, PANDA Group, UVic, Victoria, BC, Canada
gshoja@csc.uvic.ca

[3]Department of CS and ECE, PANDA Group, UVic, Victoria, BC, Canada
emanning@csr.uvic.ca

[4]Department of CS, UVic, Victoria, BC, Canada
sshelfor@uvic.ca

[5]Commlink Info Tech Ltd., R&D Group, Dhaka, Bangladesh
tareque@commlinkinfotech.com

Abstract

The Multiple-Choice Multi-Dimension Multi Knapsack Problem (MMMKP) is the distributed version of Multiple-Choice Multi-Dimension Knapsack Problem (MMKP), a variant of the 0-1 classic Knapsack Problem. Algorithms for finding the exact solution of MMKP as well as MMMKP are not suitable for application in real time decision-making applications. This paper presents a new heuristic algorithm, **Arbitrated Heuristic** (A-HEU) for solving MMMKP. A-HEU finds the solution with a few messages at the cost of reduced optimality than that of I-HEU, which is a centralized algorithm. We also discuss practical uses of MMMKP such as distributed Video on Demand service.
.

Keywords: Heuristic, Knapsack, Distributed.

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. The 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 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 *revenue* earned from that service. In mathematical notation, the 0-1 Knapsack Problem is to find V = maximize $\sum_{i=1}^{n} x_i v_i$, subject to the constraint $\sum_{i=1}^{n} x_i r_i \le R$ and $x_i \in \{0,1\}$.

The Multidimensional Multiple-choice Knapsack Problem (MMKP) is a variant of the classical 0-1 KP [5][6]. 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. A resource constraint is the availability of a particular type of resource to pick items for a particular knapsack.

We define a new problem, the Multiple-Choice Multi-Dimension Multi Knapsack Problem (MMMKP) as a distributed version of the MMKP, where the resources are distributed among knapsacks. There is a *solver* associated with each knapsack for picking the items from the group. So, distributed computing techniques will be required for picking items. The following diagram shows an example of the MMMKP.

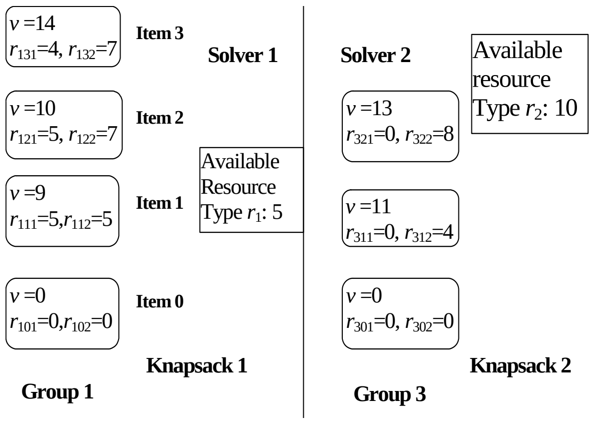

Figure 1 An MMMKP with 2 knapsacks and one resource in each knapsack

To define the MMMKP mathematically we need the following assumptions about the problem.

- There are M knapsacks. M solvers (one for each knapsack) pick items from the groups.
- The dimension of resources in Knapsack s is m_s and it provides resources labelled as μ_s to $\mu_s + m_s$ - 1 inclusive. The total set of resources of the sth knapsack is defined by $\left(R_{\mu_s}, R_{\mu_s+1}, \cdots\cdots\cdots, R_{\mu_s+m_s-1}\right)$.

- Each solver is associated with exactly one knapsack. Only Solver *s* knows the entire state of Knapsack *s* and Solver *s* is solely responsible for allocating the resources of Knapsack *s*. The state information of a knapsack, such as resources used or available, is completely private to that knapsack and its solver, unless explicitly communicated to another solver by messaging.
- There are *n* groups of items. The *i*th group has l_i items. The *j*th item of the *i*th group requires r_{ijk} of the *k*th resource. Each solver knows which resource is served by which knapsack. The value of the *j*th item of the *i*th group is v_{ij}. *n* groups are distributed among *M* solvers. The number of groups in Solvers 1, 2, ...,..., *M* are n_1, n_2, ...,n_s, ..., n_M respectively. The resource consumptions and associated values of the items of the n_s local groups of Solver *s* will not be advertised fully to all the solvers. The *partial resource consumption* of an item for a knapsack is defined by the resource requirement of the item from that knapsack. Thus, partial resource consumption of the *j*th item of the *i*th group for the resources of Knapsack *s* is expressed by the vector $\left(r_{ij\mu_s}, r_{ij(\mu_s+1)}, \cdots, r_{ij(\mu_s+m_s-1)}\right)$. The partial resource consumption of each item for any knapsack is sent to its associated solver. The set of *M* solvers will jointly execute a suitable distributed algorithm to pick exactly one item from each group, so that the total value of the picked items for the entire set of solvers is maximized subject to the resource constraints of each knapsack.

In mathematical notation, the MMMKP can be described as follows.

Maximize $V = \sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}v_{ij}$, total earned value from the picked items of the groups of all servers such that the resource constraints $\sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}r_{ijk} \le R_k$, and $\sum_{j=1}^{l_i} x_{ij} = 1$ are satisfied.

The subscripts are defined as follows:

- $k = \mu_1, \mu_1+1, \ldots, \mu_1+m_1-1, \ldots, \mu_s, \mu_s+1, \ldots, \mu_s+m_s-1, \ldots\ldots, \mu_M, \mu_M+1, \ldots, \mu_M+m_M-1$
- $x_{ij} \in \{0,1\}$, the picking variables
- i =1, 2… *n*; *j* = 1, 2… l_i.

For our example in Figure 1 we can express the problem as follows:

Maximize $V = \sum_{i=1}^{2}\sum_{j=1}^{l_i} x_{ij}v_{ij}$, subject to the resource

constraints $\sum_{i=1}^{2}\sum_{j=1}^{l_i} x_{ij}r_{ij1} \le R_1 = 5,$ and

$$\sum_{i=1}^{2}\sum_{j=1}^{l_i} x_{ij}r_{ij2} \le R_2 = 10$$

1.1 MMMKP for Solving Multimedia Distribution Problems

MMMKP can be easily applied to revenue maximization problems where we find multiple admission controllers for multimedia session requests under a particular multimedia service provider organization. With MMMKP, the admission controllers may work together sharing multimedia session requests and determine the optimal serving strategy for maximum revenue. The following example demonstrates a viable application of MMMKP in Distributed Multimedia Server System.

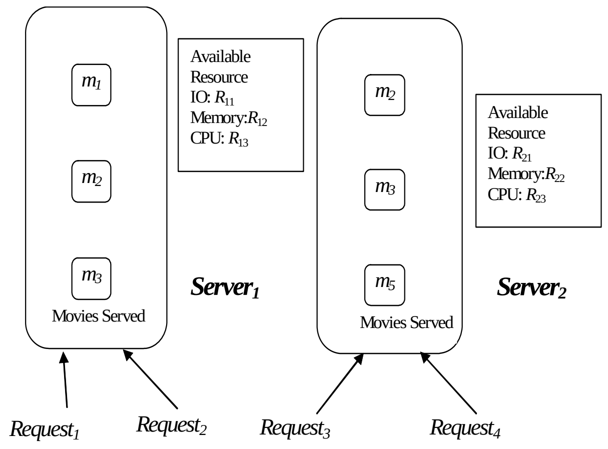
m_1
m_2
m_3
Movies Served
Available Resource IO: R_{11} Memory:R_{12} CPU: R_{13}
Server$_1$
m_2
m_3
m_5
Movies Served
Available Resource IO: R_{21} Memory:R_{22} CPU: R_{23}
Server$_2$
Request$_1$
Request$_2$
Request$_3$
Request$_4$

Figure 2 VoD servers serving requests

Consider two “Video-on-Demand” servers each serving two different collections or sets of movies as shown in Figure 2. A subscriber upon authentication may request for a multimedia session to any of the servers. If the movie does not reside in the server attempting to process it or if the server runs out of resources, the server may forward the session request to the other server.

A multimedia session between the server and the subscriber will require allocation of a number of resources on part of the server. These resources may include but are not limited to: Processing power, physical memory and IO capacity. It is allocation of these resources that determine a session’s quality of service. For sake of simplicity we consider only one level of QoS for each of the servers. Real life situations can be more complex with multiple QoS levels, separating subscribers who pay more for high quality audio-visual feed from those who settle for lesser quality. It is worth mentioning that a server has the full authority to allocate and utilize its resource only, which is one of the basic principles of distributed systems. Hence the problem can be defined as that of distributing multimedia session requests between the two servers so that maximum number of requests can be handled under the given resource constraints, thereby maximizing revenue.

From Figure 2 we find that there are 6 resource dimensions as expressed by (r_{11}, r_{12}, r_{13}, r_{21}, r_{22} r_{23}). The first three indicates the resources of *Server*$_1$ and the remaining three indicates the resources of *Server*$_2$. Figure

3 shows an example of different choices of serving the requests in further details.

Two solvers are considered representing two servers entertaining requests from the customers. $Request_1$ demands service of movie m_2. As we can see, both of the servers have the movie. Hence it is possible to serve the request in two different ways, namely *Choice*1, when served by $Server_1$ & *Choice*2, when served by $Server_2$. A generic representation for *Choice*1 would be (r_{11}, r_{12}, r_{13}, 0, 0, 0) and (0 , 0 , 0 , r_{21}, r_{22} r_{23}) for *Choice*2. Similarly $Request_3$ is for movie m_3 which both $Server_1$ and $Server_2$ offer. Hence we have two choices to have the request satisfied. On the other hand movie m_1 resides only on $Server_1$ and m_5 only on $Server_2$ leaving us with a single choice for servicing requests concerning each of these movies.

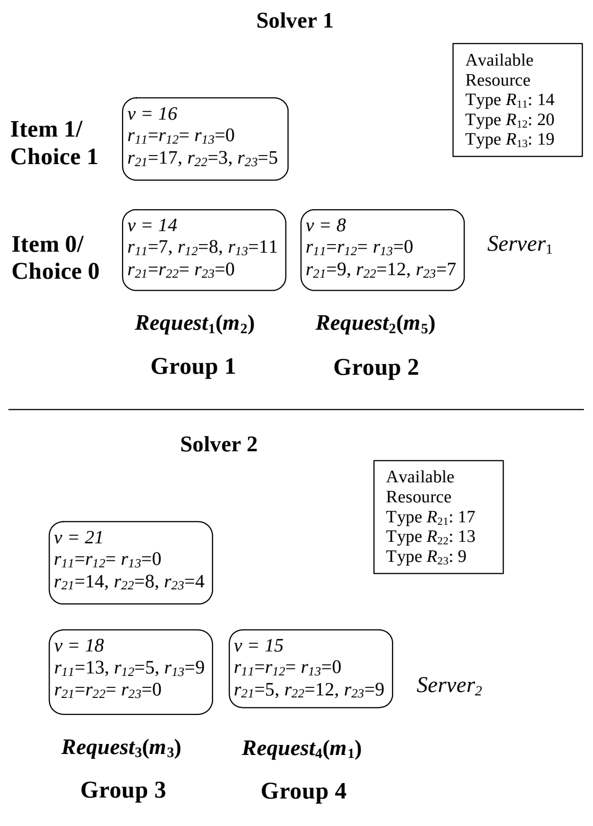

Figure 3 Multimedia distribution system mapped to MMMKP

The goal of MMMKP in such a Distributed Multimedia Server System is to pick exactly one item or QoS from each of the Group representing each request. When working independently, these servers may not choose the combination that is optimal for both of the servers, as already pointed out in the introduction section. MMMKP allows these two servers to share their decisions of resource allocation by passing messages and determines the solution that will yield maximum overall revenue.

2 Related Work on Solving Knapsack Problems

Many practical problems in resource management similar to the one discussed above can be mapped to the MMKP, consequentially their distributed version to MMMKP. But proposed exact solutions for MMKP are so computationally expensive [3][4][8] that they are not feasible for real time applications. In such cases heuristic or approximate algorithms for solving the MMKP and MMMKP play an important role.

Over the years, many heuristics have been proposed with a view to provide real time solution for MMKP. One of the earliest heuristics was HEU, proposed by Khan [7]. Khan has applied the concept of aggregate resource consumption [9] to pick a new candidate item in a group to solve the MMKP. Aggregate resource of the *j*th item of the *i*th group is defined by $a_{ij} = \sum_k r_{ijk} \times C_k \Big/ |C|$, where C_k= amount of the *k*th resource consumption and $|C| = \sqrt{\sum C_k^2}$. His heuristic HEU selects the lowest-valued items by utility or revenue of each group as an initial solution. It then upgrades the solution by choosing a new *candidate 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 (Δv_{ij})/(Δa_{ij}) (maximum value gain per unit aggregate resource expended) is chosen. Here,

$\Delta a_{ij} = \sum_k \left(r_{i\rho[i]k} - r_{ijk} \right) \times C_k$, the increase in aggregate consumed resource.

r_{ijk}= amount of the *k*th resource consumption of the *j*th item of the *i*th group.

$\rho[i]$=index of selected item from the *i*th group and $\Delta v_{ij} = v_{i\rho[i]} - v_{ij}$, is the gain in total value.

Consequently, Akbar et al. [1] proposed another heuristic using the concept of aggregate resource called M-HEU, a modified version of Khan's HEU. In M-HEU the items in each group of the MMKP are sorted in non-decreasing order according to the value associated with each item. Hence, it can be said 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. Picking a higher-valued or lower-valued item than the currently selected item in a group is called an *upgrade* or a *downgrade* respectively. The heuristic focuses on finding an upgrade or downgrade frequently. That is why the items of each group need to be sorted according to the associated values of the items. If a particular pick of items (one from each group) does not satisfy the resource constraints, that solution is defined as *infeasible*. A *feasible solution* is a solution that satisfies the resource constraints. For any resource *k*, *infeasibility factor* f_k is defined as C_k / R_k . The *k*th resource is feasible if the infeasibility factor $f_k \leq 1$, otherwise it is infeasible.

If the number of groups in the MMKP is very large then it is not possible to run M-HEU once every few seconds, as a real time system, (for example a multimedia system with 10,000 sessions) might well require. An *incremental* solution is a necessity to achieve better computation speed. By changing the technique of finding feasible solution M-HEU can be used to solve the MMKP *incrementally*, starting from an already solved MMKP. Akbar et al. named this heuristic I-HEU [1]. The proposed arbitrated heuristic A-HEU applies I-HEU in the solvers to select the probable candidate of the selected items. The steps of I-HEU are briefly described as follows:

Finding Feasible Solution (*Step* 1): In I-HEU a feasible solution is searched by selecting a lower valued item at first. If no feasible solution is found by searching lower valued item then higher valued items are looked up, like M-HEU. In this way most of the solution at hand can be re-used to obtain the new solution with less effort.

Upgrading Feasible Solution (*Step* 2): This is done by iimproving the solution value by selecting a feasible higher-valued item from the groups subject to resource constraints, i.e., by feasible upgrades.

Upgrade followed by Downgrades (*Step* 3): In this step the solution value is improved by one infeasible upgrade followed by one or more downgrades. This is analogous to get rid of local minima in the hill climbing algorithm.

Shahriar [11] presented a scalable solution to run MMKP heuristic using a multiple processor based computing server by distributing the computation among the processing nodes. But the presented algorithm is not intended for running the new problem that we have presented in this article.

In the following section, we present A-HEU, a new arbitrated heuristic, to determine the solution of the MMMKP by arbitrating among the solvers with a lower number of messages. But the total value of the items picked by A-HEU is often less than that of the centralized version.

3 Arbitrated Heuristic (A-HEU) for Solving the MMMKP

This method of solving MMMKP requires a few messages with several rounds of arbitrations; its message passing complexity is *O*(*M*). The solver in each knapsack runs I-HEU independently. The candidate for upgrades and downgrades are calculated based on the value of PCAR (Partial Change of Aggregate Resource) defined as follows:

$$\Delta pa_{ijs} = \sum_{k=\mu_s}^{\mu_s+m_s-1} \left(r_{i\rho[i]k} - r_{ijk}\right) \times C_k$$

Hence, as a simplifying assumption, the resources in other knapsacks are completely ignored in this calculation. To find a feasible solution we first run Step 1 of I-HEU in each solver. If each solver finds a feasible solution and each solver satisfies the resource constraint of all the selected items from each group, then we find a feasible solution. Now, to find upgrades and downgrades, each solver sends its *proposed list* of selected items, calculated by running Step 2 and 3 of I-HEU, to the other solvers. The proposed list is sorted according to change of value per PCAR. It is worth mentioning that to find the globally selected items according to the values of PCAR the list must be sent along with the associated values of PCAR. The items in this sorted list, which satisfy the resource constraints of all the knapsacks, can be selected. Thus arbitration among the solvers is required to select items from the groups.

The same procedure can then be repeated until we run out of real time, to attempt to obtain better total values. Here we present an arbitration technique to select the items which requires only *O*(*M*) message complexity. The following example shows only one arbitration step of A-HEU.

If we run I-HEU in both the solvers of Figure 4 independently, the solution can be shown in Table 1. As the lowest valued items are all zero, we need not find a feasible solution. Picking the *j*th item of the *i*th group can be expressed by (i, j). The proposed lists by Solvers 1 and 2 are $\{(1,3),(2,2)\}$ and $\{(3,2),(4,3)\}$ respectively. These lists are exchanged between the solvers. The sorted global list after merging these proposed lists is $\{(1,3),(4,3),(3,2),(2,2)\}$. Now the feasible picks by Solver 1 are $\{(1,3),(4,3),(3,2)\}$ with $\sum r_1 = 14$. Similarly the feasible picks by Solver 2 are $\{(1,3),(4,3)\}$ with $\sum r_2 = 15$. So Solver 1 and 2 can satisfy the first 3 and 2 picks respectively from the proposed sorted list of selected items. They exchange this information and take the set intersection of their possible solutions; namely, they pick the first 2 items from the proposed list. Hence the solution after the first arbitration is $\{(1,3),(4,3)\}$ with $\sum r_1 = 14$, $\sum r_2 = 15$ and $V = 31$.

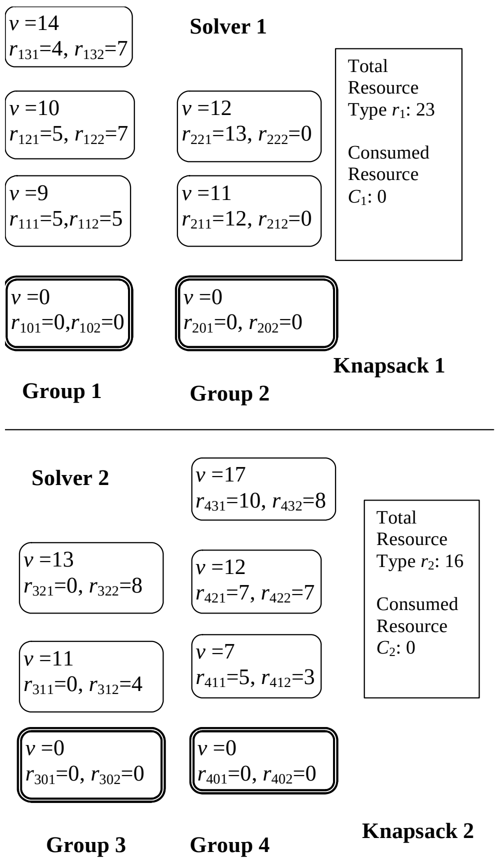
Solver 1
v =14 r_{131}=4, r_{132}=7
v =10 r_{121}=5, r_{122}=7
v =12 r_{221}=13, r_{222}=0
v =9 r_{111}=5, r_{112}=5
v =11 r_{211}=12, r_{212}=0
v =0 r_{101}=0, r_{102}=0
v =0 r_{201}=0, r_{202}=0
Total Resource Type r_1: 23
Consumed Resource C_1: 0
Knapsack 1
Group 1
Group 2
Solver 2
v =17 r_{431}=10, r_{432}=8
v =13 r_{321}=0, r_{322}=8
v =12 r_{421}=7, r_{422}=7
v =11 r_{311}=0, r_{312}=4
v =7 r_{411}=5, r_{412}=3
v =0 r_{301}=0, r_{302}=0
v =0 r_{401}=0, r_{402}=0
Total Resource Type r_2: 16
Consumed Resource C_2: 0
Knapsack 2
Group 3
Group 4

Figure 4 An MMMKP with two knapsacks.

Groups	**Picked Items**	**Change of value per PCAR**	**Resource consumption**
Group 1	3	3.5	Resource consumption in Solver 1: $\sum r_1 = 17$
Group 2	2	0.923	
Group 3	2	1.625	Resource consumption in Solver 2: $\sum r_2 = 16$
Group 4	3	2.125	

Table 1 Items picked by Solver 1 and 2 by running I-HEU independently

3.1 Format of the Messages

The messages required to run A-HEU are listed and briefly described as follows.

Message Type	**Description of the structure**	**Monitor counter associated with the message**
Groups	This is a list of groups. Each group is defined by (solver number, group number, number of items, partial resource requirements for the items)	*no_of_groups_msgs*
Local Total Value	The vector (solver number, total value) indicates the total value of the items picked by a solver.	*no_of_local_total_value_msgs*
Proposed Selected Item List	The following vector is used to define a proposed selected item (solver number, group number, item number, value per PCAR)	*no_of_local_proposed_list_msgs*
Local Feasibility Index	The vector (*s*, *L*) indicates that the first *L* items from the beginning of the global proposed selected item list are feasible with respect to the resources in Solver *s*.	*no_of_local_feasibilty_msgs*
Solution Not Found	This message indicates that a solver could not find any solution while determining proposed selected items for feasible solution.	Not applicable because the solver terminates if this message is received.

Table 2 Different Types of Messages used in A-HEU.

3.2 Sequence of Events in A-HEU

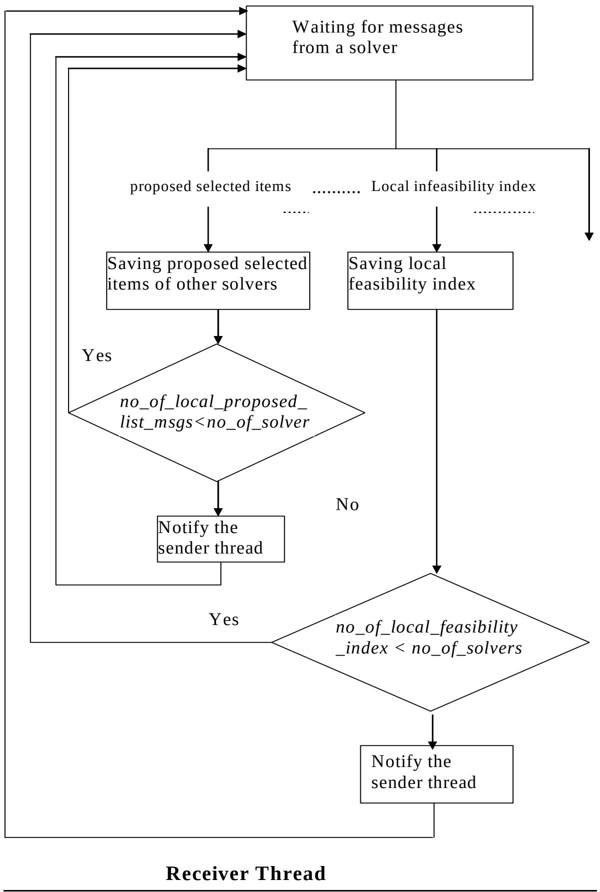
Waiting for messages from a solver
proposed selected items
....
Local infeasibility index
Saving proposed selected items of other solvers
Saving local feasibility index
Yes
no_of_local_proposed_list_msgs<no_of_solver
Notify the sender thread
No
Yes
no_of_local_feasibility_index < no_of_solvers
Notify the sender thread
Receiver Thread

Figure 5 Flow chart of distributed computation by A-HEU (Receiver Thread)

Figure 5 and 6 shows the flow chart of the processes and events in A-HEU during each arbitration. We allocate separate variables for each solver's list of proposed selected items and local feasibility index. Thus, the actions 'saving proposed selected items' and 'saving local feasibility index' executed by the receiver threads need not to be atomic or synchronized. The decision blocks in the flow chart check counters shared by all threads, so, these decision blocks must be synchronized.

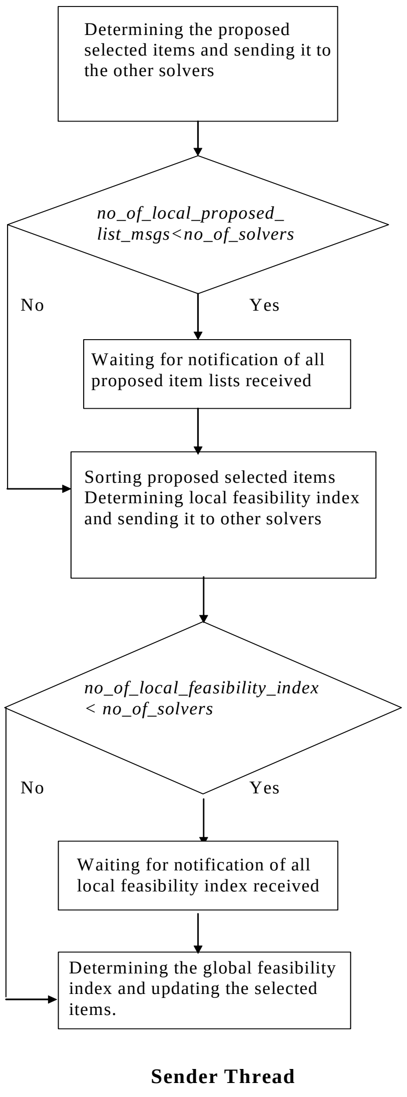
Determining the proposed selected items and sending it to the other solvers
no_of_local_proposed_ list_msgs<no_of_solvers
No
Yes
Waiting for notification of all proposed item lists received
Sorting proposed selected items Determining local feasibility index and sending it to other solvers
no_of_local_feasibility_index < no_of_solvers
No
Yes
Waiting for notification of all local feasibility index received
Determining the global feasibility index and updating the selected items.
Sender Thread

Figure 6 Flow chart of distributed computation by A-HEU (Sender Thread)

3.3 Complexity of A-HEU

Here we present the computational and message passing complexity by one arbitration of A-HEU as the first arbitration yields near optimal total value while the following iterations improve the solution with increased total values.

Computational complexity to run I-HEU on an MMKP with n/M groups and m/M resources is $\frac{3m}{M}\left(\frac{n}{M}\right)^2(l-1)^2$ in Step 2 and $\frac{7m}{M}\left(\frac{n}{M}-1\right)^2(l-1)^2$ to escape from local minima in Step 3.

We require the following messages and computations in each arbitration to find the candidate items for upgrading or downgrading:

$2(M-1)$ messages to send local proposed list of selected items.

$\frac{n(n-1)}{2}$ floating point operations to sort the locally proposed lists to determine the global proposed list.

$2\left(\frac{nm}{M}+M\right)$ comparisons to find the local and global feasibility index. Thus computation and message passing complexities in each solver are $O\left(n^2(l-1)^2\frac{m}{M^3}\right)$ and $O(M)$ respectively.

4 Experimental Results

In order to study the run-time performance of A-HEU to solve the MMMKP we implemented A-HEU along with I-HEU using Java. For simplicity of the implementation, we assume:

- Each group has the same number of items i.e., $l_1 = l_2 = \cdots, \cdots = l_n = 5$
- Each knapsack has the same number of resources i.e., $m_1 = m_2 = \cdots, \cdots = m_M = m = 4$.
- Each solver has the same number of groups i.e., $n_1 = n_2 = \cdots, \cdots = n_M = n_c$.

The algorithms were tested for an MMMKP with 3 knapsacks. Three different machines were used as three different solvers and a fourth machine was used as a generator of the MMMKP. The generator generates the groups of the MMMKP and sends them to the solvers. The generator machine also runs I-HEU on the transformed MMKP from the generated MMMKP.

4.1 Test Pattern Generation

The total amount of resources in the knapsacks, resource consumption by the items, and the values associated with the items are 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 *i*th resource = $n_c \times M \times R_c$.

P_k = Cost of the *k*th resource = $P_c \times$ *Random* (0.0, 1.0)

Random (0.0, 1.0) = A uniform continuous random number from 0.0 to 1.0.

Item 0 with zero value and zero resource consumption, i.e., r_{i0k} = 0.0 and v_{i0} = 0.0 is inserted for each group of the data set. Selection of this item indicates rejection of the group which is similar to the rejection of request in the admission controller. The other items of the groups are initialized by the following random functions:

r_{ijk} = The *k*th resource of the *j*th item of the *i*th group = $R_c \times Random$ (0.0, 1.0)

For initializing item values we use the following functions:

v_{ij} = Value of the *j*th item of the *i*th group $= \sum r_{ijk} \times P_k + Random(0.0,1.0) \times \left(m \times M \times \frac{R_c}{10} \times \frac{P_c}{10} \right)$

4.2 Test Results

The experiment was conducted for different values of n_c, from 100 up to 1000. The following data was collected from the experiments and is presented in Figure 7 to Figure 9.

- Total values of the picked items and time required by I-HEU
- Number of messages required by A-HEU
- Required time, total value of the picked items and number of messages by one, two and three arbitrations of A-HEU

In the experiment we have compared A-HEU with the centralized version I-HEU. There is no algorithm prosed so far in the literature to solve the MMMKP. That is why we present the effectiveness of A-HEU with respect to I-HEU by analyzing experimental results.

Machine name	**CPU speed**	**RAM**	**O/S**	**JDK Versions**
Solver 1	750 MHz	256 MB	Windows 2000	JDK 1.2.2
Solver 2	700 MHz	192 MB	Windows 2000	JDK 1.3.1_03
Solver 3	750 MHz	256 MB	Windows 2000	JDK 1.2.2
Generator	700 MHz	192 MB	Windows 2000	JDK 1.3.1_03

Table 3 Specifications of the solvers and generator of the MMMKP using IBM PC compatible.

4.3 Observations

- The total value of the items picked by A-HEU is approximately 90% of the total value of the items picked by D-HEU, but A-HEU requires a less time compared to D-HEU.
- Figure 9 shows the effect of message passing time in the overall complexity of the algorithm. For smaller data sets computational time is less compared to message passing. For larger data sets quadratic computation complexity of the algorithms dominates over linear message complexity. That is why better performance is observed for the larger data sets.
- We can easily conclude that A-HEU scales better than centralized I-HEU. We observe significant reduction in time requirement using A-HEU as reported by the time requirement data plotted in the exponential scale of the vertical axis.
- An irregular behaviour for the result of the set with 700 groups is observed in the figures showing the experimental results. Our complexity analysis presenting in this article is based on the worst case scenario. The actual computational time mostly depends on the data sets. A particular data set may lead to quick or late convergence to find the solution of the MMMKP showing exceptional behaviour in the result.
- For almost all the MMMKP data sets the total value of the items picked by first arbitration of A-HEU is more than 95% of the total value of the items finally picked by A-HEU.

Number of groups in each solver	$\frac{V^1_{A-HEU}}{V_{I-HEU}} \times 100$	$\frac{V^2_{A-HEU}}{V_{I-HEU}} \times 100$	$\frac{V^3_{A-HEU}}{V_{I-HEU}} \times 100$	$\frac{V_{A-HEU}}{V_{I-HEU}} \times 100$
100	92.63	93.07	93.07	93.07
200	90.71	91.15	91.15	91.15
300	92.27	92.56	92.56	92.56
400	91.43	91.52	91.52	91.52
500	91.35	91.35	91.35	91.35
600	92.43	92.43	92.43	92.43
700	74.40	87.80	87.91	88.12
800	92.05	92.05	92.05	92.05
900	91.64	91.71	91.71	91.71
1000	92.57	92.77	92.80	92.80

Table 4 Ratio of total value of the items picked by A-HEU with respect to I-HEU. V^i_{A-HEU} indicates the total value of the items picked by the *i*th arbitration of A-HEU. V_{A-HEU} and V_{I-HEU} indicates the total value of the items picked by A-HEU and I-HEU.

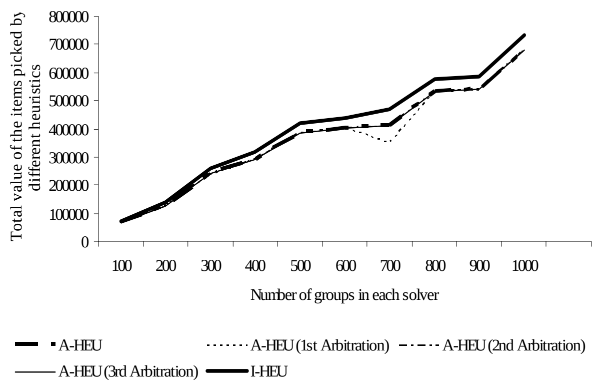

Figure 7 Total value of the items picked by A-HEU, D-HEU and I-HEU

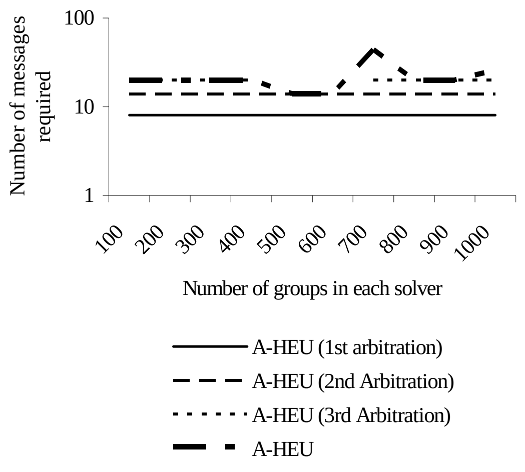

Figure 8 Number of messages required by distributed algorithms to solve the MMMKP.

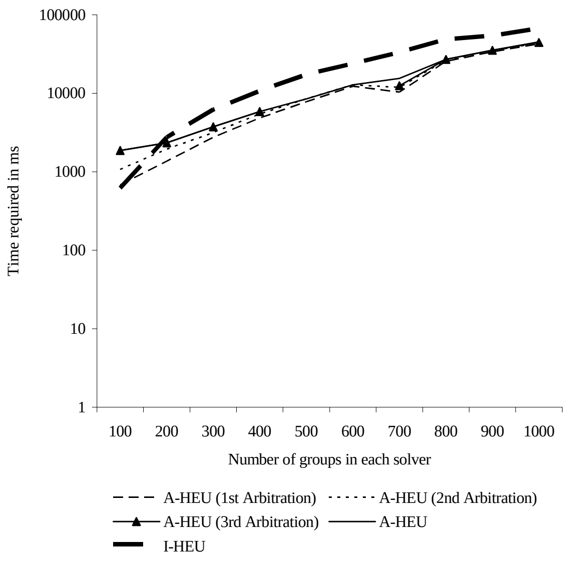

Figure 9 Time required by different algorithms to solve the MMMKP and MMKP

4.4 Discussion of the Performance of A-HEU

- An arbitration in A-HEU requires a few messages compared to any other regular distributed computing algorithms. So an iteration of A-HEU can be easily applicable for on line admission control algorithms. The admission control algorithm can execute further arbitration for better values if it has a more relaxed time constraint. Thus this algorithm is very suitable for an online system that requires quick decisions with a sub optimal total value, with the possibility of using more available time to improve the quality of the computation.
- The main reason for sub optimality in A-HEU lies in not considering all resource requirements in the preliminary selection.
- The arbitration technique for finding global feasibility index is another reason for sub optimality. We give up upgrading from the proposed list of selected items if we get one infeasible upgrade in the list. The very next item might be feasible. A new arbitration technique with $O(nM)$ message complexity might do that. However, if we do the arbitration again, with a newly calculated proposed selected item list for better total value, the competent items will get a chance to be selected. That is why we prefer multiple iterations of the arbitration, to a new arbitration technique with $O(nM)$ message complexity.
- If the items of the groups consume all the resources of all knapsacks uniformly then A-HEU is unlikely to get better total value in the next arbitration, because a particular resource has already been exhausted in the previous arbitration.

In practical cases, such as multimedia servers and Enterprise Networks, a QoS level of a session requires resources of a particular server or a particular network link. So there is a chance to allocate available resources to other selected items of the groups in the next arbitration.

5 Conclusion

In this article we have presented a new variant of knapsack problems which requires distributed algorithm to solve the problem. Our proposed new algorithm A-HEU achieves almost 90% optimality of I-HEU in $1/O(M^3)$ of the computation time required by I-HEU and O(M) message passing where M is the number of solvers in the system. The experimental results show that the message passing time can be ignored for a larger problem set. Thus the new algorithm presents a scalable with the scope of distributed control. This particular problem is very much applicable for different resource sharing system in the distributed real time systems. This algorithm is a potential candidate for admission controlling in distributed real time systems.

6 References

[1] Akbar, M., Manning, E. G., Shoja, G. C. and Khan, S. (2001): Heuristic solutions for the multiple-choice multi-dimension knapsack problem. *Proceedings of the International Conference on Computational Science* 659–668, San Francisco, Calif, USA, May 2001.

[2] Akbar, M., Manning, E. G., Shoja, G. C. (2001): Admission Control and QoS adaptation in Distributed Multimedia Server System. *ITCom* 2001 Denver, USA, August 2001.

[3] Armstrong, R., Kung, D., Sinha, P. and Zoltners, A. (1983): A Computational Study of Multiple Choice

Knapsack Algorithm. *ACM Transaction on Mathematical Software* 9:184-198.

[4] Koleser, P. (1967): A Branch and Bound Algorithm for Knapsack Problem. *Management Science* 13:723-735.

[5] Nauss, R. (1978): The 0-1 Knapsack Problem with Multiple Choice Constraints. *European Journal of Operation Research* 2:125-131.

[6] Magazine, M. and Oguz, O. (1984): A Heuristic Algorithm for Multidimensional Zero-One Knapsack Problem. *European Journal of Operational Research* 16(3):319-326.

[7] Khan, S., Li, K. F. and Manning, E.G. (1997): The Utility Model for Adaptive Multimedia System. *International Workshop on Multimedia Modeling* 111-126.

[8] Shih, W. (1979): A Branch and Bound Method for Multiconstraint Knapsack Problem. *Journal of the Operational Research Society* 30:369-378.

[9] Toyoda, Y. (1975): A Simplified Algorithm for Obtaining Approximate Solution to Zero-one Programming Problems. *Management Science* 21:1417-1427.

[10] Hifi, M and Michrafy, M. (2006): A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society* 57:718-726.

[11] Shahriar, M.A.Z, Akbar, M.M., Rahman, M.S. and Newton, M.A.H. (2008): A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. *The Journal of Supercomputing* 43(3): 257-280.

[12] Hifi, M., Michrafy, M. and Sbihi, A. (2004): Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *Journal of the Operational Research Society* 55:1323–1332.