SOLVING THE KNAPSACK PROBLEM FOR ADAPTIVE MULTIMEDIA SYSTEMS

SHAHADAT KHAN, KIN F. LLERIC G. MANNING AND MD MOSTOFA AKBAR

Abstract. The Multiple-Choice Multi-Dimension Knapsack Problem (MMKP) is a variant of the 0-1 knapsack problem, an NP-Hard problem. Due to its high computational complexity, algorithms for exact solution of the MMKPs are not suitable for most real-time decision-making applications, such as quality adaptation and admission control for interactive multimedia systems, or service level agreement (SLA) management in telecommunication networks. This paper presents a heuristic for finding near-optimal solutions of the MMKP, with reduced computational complexity, and is suitable for real-time applications. Based on Toyoda’s concept of aggregate resource, the heuristic employs an iterative improvement procedure using savings in aggregate resource and value per unit of extra aggregate resource. Experimental results suggest that this heuristic finds solutions which are close to the optimal (within 6% of the optimal value), and that it out-performs Moser’s heuristic for the MMKP in both solution quality and execution time.

Keywords: Knapsack, Multimedia, Heuristic.

1. Introduction

The multiple-choice multi-dimension knapsack problem (MMKP) is one of the harder variants of the 0-1 knapsack problem. Not extensively studied in the past, it is now of great practical interest, because it can be used to model the allocation of resources in a computer network, and the dynamic adaptation of system resources to provide Quality of Service guarantees as required by multimedia traffic [Chen99]. It has also been successfully used as the mathematical basis of revenue-optimal admission controllers for internets [Watson01]. The MMKP can be stated as follows.

Suppose there are $n$ groups (stacks) of items. Group $i$ has $l_i$ items. Item $j$ of group $i$ has value $v_{ij}$, and requires resources given by vector $r_{ij} = (r_{ij1}, r_{ij2}, ..., r_{ijm})$. The amounts of available resources are given by $R = (R_1, R_2, ..., R_m)$. The MMKP is to pick exactly one item from each group in order to maximize the total value of the pick, subject to the resource constraints.
Formally, the MMKP is expressed as follows:

\[
V_{\text{max}} = \text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij},
\]

such that

\[
\sum_{j=1}^{l_i} x_{ij} r_{ijk} \leq R_k, \quad k = 1, \ldots, m,
\]

\[
\sum_{j=1}^{l_i} x_{ij} = 1, \quad i = 1, \ldots, n,
\]

\[
x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n; j = 1, \ldots, l_i.
\]

\(x_{ij}\) is either 0, implying item \(j\) of group \(i\) is not picked, or 1 implying item \(j\) of group \(i\) is picked. Figure 1.1 illustrates an MMKP with three groups of items and two resources. Here each item has a value \(v\), and two resource components \(r\) and \(p\).

A pick denotes a selection of \(n\) items, one from each stack. The resource vector denoting the sum of resources required by the picked items is called the resource-usage vector and is denoted with symbol \(\mathbf{C}\). For any resource \(k\), \(C_k / R_k\) is called its feasibility factor. Resource \(k\) is considered feasible if \(C_k / R_k \leq 1\); otherwise, the resource is called infeasible.

A pick where the feasibility factor of each resource is less than or equal to 1 is called a feasible solution of the knapsack problem. The sum of the values of the items picked in a feasible solution is called the value of the solution or solution-value for short. Thus the solution of the MMKP is the feasible solution which maximizes the sum of the value of the selected items.

The rest of the article is organized as follows. Section 2 presents a model for adaptive multimedia systems and shows the mapping of an adaptive multimedia problem onto an MMKP. Section 3 presents a brief discussion of the relation of this work to other work in the literature. Section 4 presents a heuristic for a near-optimal solution of the MMKP suitable for time-critical applications. It also presents a computational analysis of the heuristic, and performance results. Finally section 5 concludes the article.

2. Adaptive Multimedia System as an MMKP

In order to capture the issues of resource management in adaptive multimedia systems (AMSS) with multiple concurrent sessions, where the qualities of individual sessions are dynamically adapted to the available resources and to the
Solving the Knapsack Problem for Adaptive Multimedia Systems

Figure 1.1: The multiple-choice multi-dimension knapsack problem.

User preferences, Khan et al. proposed a mathematical model, called the Utility Model [Khan97, Khan98]. It provides a unified and computationally feasible way to solve the admission problem for new multimedia sessions, and to solve the dynamic quality adaptation and integrated resource allocation problems for existing sessions.
The main concepts of the Utility Model are illustrated in Figure 1.2:

1. Each user specifies a quality profile, which is the set of acceptable operating qualities for her session. The quality profile of a session is a sequence of acceptable operating qualities in increasing order of preference (from minimum acceptable quality to the highest desired quality). Mathematically, the quality profile of session $i$ can be expressed as a vector,

\[ P_i = (q_{i1}, q_{i2}, q_{i3}, \ldots, q_{i\ell_i}) \]

where $\ell_i$ is the number of qualities in $P_i$, $q_{i1}$ is a vector denoting the minimum acceptable quality (e.g. phone-quality sound, black and white video) and $q_{i\ell_i}$ is the highest desired quality (e.g. surround sound, High Definition Colour TV).
2. A session’s operating qualities are considered to be mapped uniquely to the required resources. Let us assume only three resources within a system: processor cycles (\(p\)), main memory (\(m\)), and network bandwidth (\(b\)). Then the required resources \(r_i\) of session \(i\) can be expressed by a three-tuple \(r_i = (p_i, m_i, b_i)\). In vector notation, the quality-resource mapping can be expressed as

\[
\mathbf{r}_i = \mathbf{r} (\mathbf{q}_i).
\]

3. A session’s operating quality \(q_i\) is also assumed to be mapped uniquely to a session utility \(u_i(q_i)\).

4. The system utility is expressed as some function of all session utilities, often the sum,

\[
U = \sum_{i=1}^{n} u_i(q_i).
\]

5. The system is subject to the system resource constraints. For each resource, the sum of the quantities of the resource allocated to all the sessions cannot exceed the total available quantity of the resource\(^1\). Suppose the available system resources are expressed as a vector \(\mathbf{R} = (P, M, B)\). The resource constraints are expressed as

\[
\sum_{i=1}^{n} \mathbf{r}(q_i) \leq \mathbf{R}.
\]

Now the main problem in an adaptive multimedia system can be expressed as follows: Find the operating quality \(q_i\) of each session \(i\) which \textit{maximizes} the system utility \(U\) under the system \textit{resource constraints}.

### 2.2. Quality Adaptation as a Knapsack Problem

The above mentioned Adaptive Multimedia Problem (AMP) can be mapped to an MMKP as follows.

- A session’s quality profile \(\mathbf{P}_i = (q_{i1}, q_{i2}, q_{i3}, \ldots, q_{in})\) can be viewed as a stack of \(l_i\) items. Thus \(n\) sessions’ quality profiles map to \(n\) stacks of items, and item \(j\) of stack \(i\) denotes operating quality \(q_{ij}\) of session \(i\).
- When session \(i\) operates at quality \(q_{ij}\), its session utility \(u_{ij}\) is denoted as value \(v_{ij}\), and the amount of resource \(k\) consumed by session \(i\) is denoted as \(r_{ijk}\) (quality-resource mapping).
- The system utility \(U\) of the AMP is mapped to the value of the pick \(V\).

\(^1\)In this way, and unlike the current practice in the Internet, UNIX system and airlines, where overbooking is common, we guarantee Quality of Service.
The resource constraints of the AMP are equivalent to the resource constraints of the MMKP.

Finding an operating quality for each session can be viewed as picking (exactly) one item from each stack.

2.3. Applications of the Utility Model

The Utility Model may be used in session admission, dynamic quality adaptation and integrated resource management in real-time multimedia systems.

- Session Admission: Suppose the system has $n$ sessions, and a user requests another session. Provided that there are enough system resources available for the new session, the achievable system utility $U'$ with $(n + 1)$ sessions is compared to the current system utility $U$ with $n$ sessions. If $U' > U$, then the new session should be admitted; otherwise the new session should be rejected.

- Quality Adaptation: Suppose a session is dropped or the amount of some resource available is changed due to external and/or uncontrollable reasons. The system computes the solution of the Utility Model under the new constraint with the former solution and some of the existing sessions may have to change their operating qualities.

- Integrated Resource Management: Since the Utility Model considers allocation of all the system resources in an integrated way, it avoids situations such as a session getting sufficient network bandwidth to carry a video stream, but unable to utilize this bandwidth effectively because the system cannot allocate to it sufficient CPU cycles to decode and render the video stream.

3. Related Work

Variants of knapsack problems comprise an important class of combinatorial optimization problems with such diverse applications as capital budgeting, industrial production, menu planning, cargo loading, information systems and resource allocation [Martello87]. Please refer to [Martello87] and [Chu97] for comprehensive surveys of algorithms for the knapsack problem and some of their variants.

The multi-dimension knapsack problem (MDKP) is a generalization of the classical 0-1 KP for multiple resource constraints (or dimensions). The Multiple Choice Knapsack Problem (MCKP) is another KP where the picking criterion for items is more restricted. In this variant of KP there are one or more
groups of items. Exactly one item will be picked from each group. In a way, MMKP is the combination of MDKP and MCKP.

Since the MMKP is an NP-Hard problem, the worst-case computation time of the optimal solution grows exponentially with the size of the problem. For this reason, there are two types of solutions proposed: optimal solutions, and near-optimal solutions. Near-optimal solutions are close to optimal, but require computation times which are much shorter than those of the optimal solutions.

Most algorithms for optimal solution of KP variants are based on the branch and bound search technique. The main idea behind this technique is the exploration of a tree of possible solutions. Each node represents a solution state and the upper bound of the state is calculated by using linear programming methods. Using this 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 [Kolesar67, Shih79, Nauss78, Khan98]. Although the use of linear programming to determine the requirement of further exploration of a node reduces the time requirement in the average case, it is not feasible to apply it in all practical cases, especially in time-critical decision-making applications.

A greedy approach has been suggested for approximate solution of the knapsack problems [Martello87, Shih79, Toyoda75]. For the classical 0-1 KP, it involves two steps: (1) sort the items in descending order of value-resource ratio \( \frac{v_i}{r_i} \) and (2) pick as many items as possible from the beginning of the list until the resource constraint is violated.

To apply the greedy method to MDKP, Toyoda proposed a new measurement called aggregate resource consumption \( a_i = \frac{r_i \cdot C}{|C|} \), where \( C = (C_1, C_2, \ldots, C_m) \) denotes the current resource usage vector and \( r_i = (r_{i1}, r_{i2}, \ldots, r_{im}) \) denotes the resource requirements of item \( i \). The solution of the MDKP involves iterative picking of items until the resource constraint is violated. Initially no item is picked, \( C = 0 \) and \( a_i = r_i \). In each iteration the values of \( a_i \) are calculated, and the item with the greatest value of \( v_i / a_i \) is picked. This selection criterion penalizes the items giving low revenue with high resource consumption. It is highly probable that the item giving the highest revenue per unit resource consumption will lead us to a near-optimal solution.

Moser [Moser97] proposed a heuristic for the MMKP using the concept of graceful degradation from the most valuable items based on Lagrange Multipliers. The basic steps of this method are:

1. Selecting the most valuable item from each pile of items. This might violate some resource constraints, so an iterative perturbation technique is used to reach a feasible solution. In every iteration, an item is exchanged
such that it would decrease the infeasibility of the most infeasible resource constraint. If no such item is found in an iteration and the current pick is still infeasible the heuristic exits with no solution.

2. Iterative improvement of the solution, by replacing a selected item with another one with a higher value in each iteration.

4. A Heuristic Solution for the MMKP: HEU

4.1. Concepts

The heuristic developed for the MMKP is based on the following concepts:

- It starts with finding a feasible solution. At first it tries a solution where from each group, the item with the smallest value \(v_{ij}\) is picked. If this is not feasible it tries an iterative procedure of replacing one picked item with another to find a feasible solution.
- It uses Toyoda’s concept of aggregate resource [Toyoda75] for selecting items to pick. Here the main idea is to use weighted penalization for the not-yet-picked items depending on the current resource usage and the resource requirement of the items. It applies a large penalty factor for a heavily used resource, and a small penalty factor for a lightly used resource.
- It uses iterative improvement of the solution by using exchange of picked items. In each exchange, the status of two items (one selected and the other not-selected) in a group are swapped. In this sense, this is a 2-interchange heuristic.

Our approach in this heuristic is to find the optimal or near-optimal solution by upgrading the feasible solution. The algorithm will work if we start upgrading from any feasible solution. The heuristic proposed by Moser however, finds a feasible solution starting from the highest-valued items. Experimental results show that this approach does not perform well for practical problems like AMP or SLA admission control, where the resource consumption by items follows the monotone feasibility property (i.e., the item with higher resource consumption gives higher value of revenue), in most cases. It is easy to find a feasible solution by checking the lowest valued items in the cases where the resource consumption follow the monotone feasibility property. So, we start from the lowest valued items and try to find a feasible solution by upgrading if the initial pick is not feasible.

SIU 2001
4.2. Procedure HEU

Procedure HEU is presented in Figure 4.3. It is assumed that, within a group \( i \), the items \( j = 1, \ldots, l_i \) are arranged in nondecreasing order of value, that is if \( j' < j'' \) then \( v_{ij'} \leq v_{ij''} \). This does not affect the generality of the heuristic, because even if this is not true for a given instance of the MMKP, the ordering may be achieved by a little preprocessing.

Let us introduce some definitions and notations that will be used in the heuristic. Suppose \( \rho = (\rho_1, \rho_2, \ldots, \rho_n) \) denotes the current resource vector. Then the current state of resource usage is denoted by the resource usage vector \( C = \sum_{i=1}^{n} r_i \rho_i \).

We consider an exchange \( X = (i, j) \) where item \( j \) is picked from group \( i \) instead of item \( \rho_i \). Suppose \( \rho|X \) denote the solution vector after exchange \( X \) from \( \rho \). Using Toyoda’s concept, we can say that this exchange will provide us a saving in aggregate-resource given by \( \Delta a(\rho, i, j) = \frac{(r_{ij} - r_{ij})}{|C|} \times C \). Since the gain in solution-value from this exchange is \( v_{ij} - v_{ij} \), then value gain per unit of extra resource can be expressed as \( \Delta p(\rho, i, j) = \frac{v_{ij} - v_{ij}}{|\Delta a(\rho, i, j)|} = \frac{v_{ij} - v_{ij}}{\Delta a(\rho, i, j)} \).

We also define savings in total-per-unit-resource as \( \Delta t(\rho, i, j) = \sum_{k=1}^{n} \frac{r_{ij} - r_{ij}}{R_k - C_k} \). Here savings in total-per-unit-resource denotes the sum of the ratios of resource savings to available resource for all resources. Then value gain per unit of total-per-unit-resource can be expressed as \( \Delta p'(\rho, i, j) = \frac{v_{ij} - v_{ij}}{\Delta t(\rho, i, j)} \).

The heuristic has three steps as follows.

1. Finding a feasible solution Step 1 starts with a pick which includes the lowest-valued item from each stack. If this pick is feasible then we proceed to the next step for iterative improvement.

   If the initial pick is not feasible, we find the resource with the highest feasibility factor. Suppose it is resource \( \alpha \). The next step will be to find a higher valued item such that if this item is picked (instead of the current item picked from its group) then:
   (a) it will decrease the feasibility factor of resource \( \alpha \),
   (b) it will not increase the feasibility factor of any previously infeasible resource, and
   (c) it will not make any previously feasible resource infeasible.

   If such an item cannot be found, the procedure returns with “no solution found”. If more than one candidate items are found, we pick the item that maximizes the savings in aggregate resource \( \Delta a \). If the current solution is feasible, we proceed to step 2. If the current solution is not feasible,
we repeat the above step until either a feasible solution is reached or the heuristic is returned with “no solution found”.

// Symbols:
// n: groups, i: items in group i, m: system resources,
// v: value vector, r: required resource vector,
// R: total resource vector, C: resource usage vector,
// p: current solution vector, P[Z]: solution vector after exchange Z from p
// Δα: aggregate resource savings, Δτ: total-per-unit-resource savings,
// Δρ: value gain per unit of extra resource
// U: total value (utility) of the current pick.
1. Finding a feasible solution.
   1.1 Start with a pick of items having the smallest value in each group:
      ∀i = 1, ..., n : ρi = 1. Compute C = \sum_{i=1}^{n} riρi.
   1.2 Find α where Cα / Rα = \max_{k=1,2, ..., m} Ck / Rk. If Cα / Rα ≤ 1, go to step 2.
   1.3 Consider an exchange X = (i, j) from p which picks item j from group i
      instead of item ρi. Define Δα(ρ, i, j) = (\rho_{i,j} - \rho_{i,j}) C. Let ρ' = ρ[X].
      Find β and γ such that Δα(ρ, β, γ) = \max_{k=1,2, ..., m, i,j=1,2, ..., n} \Delta(\rho, i, j) such that
      C_{α}(ρ') < C_{α}(ρ).
      if k ≠ α and C_{α}(ρ) ≤ R_{k}(ρ') then C_{α}(ρ') ≤ R_{k}(ρ')
      if k ≠ α and C_{α}(ρ) > R_{k}(ρ) then C_{α}(ρ') ≤ C_{α}(ρ).
      If β and γ are found, set ρ = ρ[(β, γ)] and go to step 1.2.
      Otherwise exit procedure with “No solution found”.
2. Iterative improvement using feasible upgrades.
   2.1 Consider an upgrade X = (i, j) where item j is picked from group i
      instead of item ρi. Define Δρ(ρ, i, j) = v_{i,j} - v_{j,i}.
      Find feasible upgrade X' = (δ, η) that maximizes Δα(ρ, i, j).
      If X' is found and Δα(ρ, δ, η) > 0, set ρ = ρ[X'] and go to step 2.1.
   2.2 Find feasible upgrade X' = (δ, η) that maximizes Δρ(ρ, i, j).
      If X' is found and Δρ(ρ, δ, η) > 0, set ρ = ρ[X'] and go to step 2.1.
3. Iterative improvement using upgrades followed by one or more downgrade(s).
   3.1 Find an upgrade Y = (δ', η') which maximizes Δρ(ρ, i, j)
      where Δρ(ρ, i, j) = \frac{v_{i,j} - v_{j,i}}{\Deltaα(ρ, i, j)} and Δτ(ρ, i, j) = \sum_{k=1}^{m} \frac{r_{i,j,k} - r_{j,i,k}}{R_{k} - C_{k}}. Set ρ' = ρ[Y].
   3.2 Find a downgrade Y' = (δ'', η'') which minimizes Δρ(ρ', i, j)
      such that U(ρ' [Y']) > U(ρ).
      where Δρ(ρ', i, j) = \frac{v_{i,j} - v_{j,i}}{\Deltaτ(ρ', i, j)} and Δτ(ρ', i, j) = \sum_{k=1}^{m} \frac{r_{i,j,k} - r_{j,i,k}}{R_{k} - C_{k}}.
      If Y' is found and (ρ' [Y']) is feasible, set ρ = ρ'[Y'] and go to step 2.
      If Y' is not feasible, set ρ' = ρ'[Y'], and go to step 3.2.
   3.3 Exit procedure with solution vector ρ.

Figure 4.3: Procedure HEU: A heuristic for MMKP.

2. Iterative improvement using feasible upgrades

SIU 2001
Replacing an item in a solution with another one from the same group is called an exchange. An exchange where the value of the solution is increased is called an upgrade. Conversely, an exchange that decreases the value of the solution is called a downgrade. An exchange is called feasible if the solution after the exchange is feasible; otherwise it is called infeasible.

In this step, we try to improve the solution using iterative search for feasible upgrades. In each iteration, finding a feasible upgrade involves the following steps:

- Find the savings in aggregate resource ($\Delta a$) for all feasible upgrades.
- If there exists at least one feasible upgrade which provides savings in aggregate resource (that is the extra aggregate resource computed for the upgrade is negative), then HEU chooses the upgrade which maximizes the savings in aggregate resource. This is plausible as it improves the feasibility of the resources and frees up resources thus making room for possibly more feasible upgrades.
- However, if there exists no feasible upgrade which provides savings in aggregate resource, then HEU chooses the upgrade which maximizes the value gain per unit of extra aggregate resource ($\Delta p = \frac{\Delta r}{\Delta a}$).

3. Iterative improvement using upgrades followed by one or more downgrade(s)

In this step, we try iterative improvement of the solution using one upgrade followed by one or more downgrades in each iteration. This is an attempt to avoid getting stuck on local maxima. If the heuristic fails to improve the solution in one iteration, it returns the current solution, and terminates.

The upgrade is chosen in order to maximize $\Delta p'$, the gain in solution value per unit of extra total-per-unit-resource ($\Delta t'$). The downgrade is chosen in order to minimize $\Delta p''$, loss in solution value per unit of extra total-per-unit-resource ($\Delta t''$).

4.3. Computational Analysis

All the three steps of the heuristic may require iteration. First we should investigate whether the solution will converge.
4.3.1. Non-Regenerative Property

The three steps of HEU never regenerate a solution, which has been found previously. If the two solutions of an MMKP have different total values or different resource consumptions it is obvious that the items picked for these two solutions are not the same. The obvious reasons for this convergence property are as follows:

- Step 1 never makes any feasible resource requirement infeasible, or infeasible resource requirement more infeasible. This step does not generate a solution state more than once. To prove this by contradiction, suppose HEU generates two infeasible solutions $S_i$ and $S_j$ at the $i$th and $j$th iteration of step 1, and $S_i = S_j$ and $j > i$. The resource consumptions by these solutions can be expressed as $(C_{i1}, C_{i2}, \ldots, C_{iM})$ and $(C_{j1}, C_{j2}, \ldots, C_{jm})$. Since $S_i = S_j$, for each feasible resource $x$, $C_{ix} = C_{jx}$. Now step 1.3 requires that there is at least one infeasible resource $\alpha$ for which $C_{i(\alpha+1)} < C_{i\alpha}$. Step 1.3 also requires that $C_{jx} \leq C_{(j-1)\alpha} \leq \cdots \leq C_{(j+2)\alpha} \leq C_{(j+1)\alpha}$. We observe that this contradicts with $C_{j\alpha} = C_{i\alpha}$, and thereby proves our claim.

- Step 2 always upgrades the solution vector with increased total value. Let the solution vectors and revenues in the iterations be $(S_1, U_1), (S_2, U_2), \ldots (S_i, U_i) \ldots (S_N, U_N)$. Since $U_1 < U_2 < \cdots < U_i < \cdots < U_N$, it can be concluded that $S_1 \neq S_2 \neq \cdots \neq S_i \neq \cdots \neq S_N$.

- Step 3 improves the solution using a sequence of upgrades followed by one or more downgrade(s). We note that when step 3.1 chooses an infeasible upgrade, step 3.2 stops looking for downgrades as soon as a downgrade causes the solution value to go lower than the solution value of a previously saved feasible solution $\rho$. Thus steps 3.1 and 3.2 cannot cause infinite looping.

We note that HEU allows jumping to step 2.1 from step 3.2 only when another feasible solution with a larger solution value is found. Since there are only a finite number of feasible solutions, the iterative solution improvement in steps 2 and 3 together cannot cause any infinite looping.

4.3.2. Worst-Case Complexity

In this subsection we present upper bounds for the worst-case computational complexity of the three steps in our MMKP heuristic with $n$ groups and $m$ resources. For simplicity, suppose $l_1 = \cdots = l_n = l$.

In step 1.1, computation of $C$ has complexity $O(nm)$. Finding $\alpha$ in step 1.2 may require $O(m)$ comparisons. In step 1.3, finding $\beta$ and $\gamma$ may require
Solving the Knapsack Problem for Adaptive Multimedia Systems

a maximum of \( n(l - 1) \) tests where each test will require \( m \) computations of \( C_k \) and \( m \) comparisons of \( C_k \) and \( R_k \). This may require up to \( O(n(l - 1) \times m) \) operations. Steps 1.2 and 1.3 may be repeated at most \( n(l - 1) \) times since there could be at most \((l - 1) \) updates for each group. Thus the worst-case complexity of step 1 is \( O(n^2 (l - 1)^2 m) \).

Computing \( \Delta a \) and \( \Delta p \) in step 2.1 requires \( O(m) \) operations. Finding \( X' \) in step 2.2 may require up to \( n(l - 1) \) computations of \( \Delta p \). Thus each iteration of steps 2.1 and 2.2 has complexity of \( O(n(l - 1)m) \). Since there could be at most \( n(l - 1) \) feasible upgrades, the maximum complexity of step 2 is \( O(n^2 (l - 1)^2 m) \).

The amount of available resources is expected to be very small in step 3, and therefore the heuristic is expected to spend the least time in this step. This is analogous to the hill climbing approach in a plateau, for classical AI problems. We present the worst case complexity of an escape from local maxima using one upgrade followed by one or more downgrades. In step 3.1, the complexity of finding upgrade \( Y \) is \( O(n(l - 1)m) \) because it may require up to \( n(l - 1) \) computations of \( \Delta p' \). There might be \((n - 1)(l - 1) \) possible downgrades. In step 3.2, each downgrade requires computational complexity \( O(n(l - 1)m) \) to find the lowest \( \Delta p'' \). Thus the complexity to escape from local maxima is \( O(n^2 (l - 1)^2 m) \).

Step 3 acts as an escape technique from local maxima as used in different local search heuristics. This step transfers the control to step 2 whenever it gets an upgraded feasible solution. Combined complexity of steps 1 and 2 gives the computational behavior of the algorithm. Step 1 finds the feasible solution and also upgrades the solution. Step 2 upgrades this feasible solution. The orders of the computational complexity in step 1 and step 2 are the same. Step 2 starts right after step 1 and with the feasible solution vector calculated from step 1. Thus the combined worst case complexity of step 1 and step 2 is \( O(n^2 (l - 1)^2 m) \).

4.4. Performance Results

In order to study the run-time performance of HEU we implemented it along with two other algorithms: (1) an optimal algorithm based on branch and bound search using linear programming for upper bound computation (BBLP) and (2)
Moser’s heuristic (MOSER) based on Lagrange Relaxation. Moser’s heuristic is a practically applicable algorithm. Although BBLP is infeasible in practical application for larger data sets, we run this algorithm to determine the optimality of the heuristics by finding an upper bound using the linear programming approach. 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 (that is \( l_1 = l_2 = \cdots = l_n = l \)). We used the Simplex Method code from [Press92] for linear programming.

We ran the algorithms on a 700 MHz Pentium III IBM ThinkPad with 196 MB of RAM running Windows 2000. We have performed experiments on an extensive set of problem sets. We used randomly generated and correlated MMKP instances for our tests. For each set of parameters \( n, l \) and \( m \), we

<table>
<thead>
<tr>
<th>MMKP parameters</th>
<th>solution value</th>
<th>execution time</th>
<th>solution not found</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( l )</td>
<td>( m )</td>
<td>( V_{\text{BLP}} )</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>551.80</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5</td>
<td>609.70</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>586.40</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>5</td>
<td>1126.90</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>5</td>
<td>1339.30</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>5</td>
<td>1374.20</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>5</td>
<td>2003.10</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>5</td>
<td>1751.10</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>5</td>
<td>2074.80</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>5</td>
<td>2864.90</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>5</td>
<td>2893.80</td>
</tr>
<tr>
<td>20</td>
<td>9</td>
<td>5</td>
<td>2305.20</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>5</td>
<td>3311.90</td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>5</td>
<td>3286.30</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>5</td>
<td>2685.70</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>5</td>
<td>3631.50</td>
</tr>
<tr>
<td>30</td>
<td>7</td>
<td>5</td>
<td>3498.20</td>
</tr>
<tr>
<td>30</td>
<td>9</td>
<td>5</td>
<td>3946.60</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison of BLP, HEU and MOSER for correlated data sets. For each set of parameters \( n, l \) and \( m \), we used 10 MMKP instances, and the last three columns \( m, h \) and \( b \) show the number of instances where MOSER, HEU and BLP failed to find a feasible solution.
Solving the Knapsack Problem for Adaptive Multimedia Systems

Table 2: Performance comparison of BBLP, HEU and MOSER for random data sets. For each set of parameters \( n, l \) and \( m \), we used 10 MMKP instances, and the last three columns \( m, h \) and \( b \) show the number of instances where MOSER, HEU and BBLP failed to find a feasible solution.

<table>
<thead>
<tr>
<th>MMKP parameters</th>
<th>solution value</th>
<th>execution time</th>
<th>solution not found</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n ) ( l ) ( m )</td>
<td>( V_{\text{max}} ) ( V_{\text{max}} % ) ( V_{\text{avg}} % )</td>
<td>( t_{\text{BBLP}} ) ( m ) ( h ) ( b )</td>
<td></td>
</tr>
<tr>
<td>5 5 5</td>
<td>274.64</td>
<td>98.34</td>
<td>98.53</td>
</tr>
<tr>
<td>5 7 5</td>
<td>296.54</td>
<td>94.06</td>
<td>96.58</td>
</tr>
<tr>
<td>5 9 5</td>
<td>324.94</td>
<td>93.48</td>
<td>98.24</td>
</tr>
<tr>
<td>10 5 5</td>
<td>587.56</td>
<td>95.69</td>
<td>99.94</td>
</tr>
<tr>
<td>10 7 5</td>
<td>633.83</td>
<td>92.84</td>
<td>98.92</td>
</tr>
<tr>
<td>10 9 5</td>
<td>617.48</td>
<td>94.26</td>
<td>97.32</td>
</tr>
<tr>
<td>15 5 5</td>
<td>903.52</td>
<td>92.93</td>
<td>99.06</td>
</tr>
<tr>
<td>15 7 5</td>
<td>988.21</td>
<td>94.31</td>
<td>99.15</td>
</tr>
<tr>
<td>15 9 5</td>
<td>1001.29</td>
<td>92.83</td>
<td>98.91</td>
</tr>
<tr>
<td>20 5 5</td>
<td>1274.62</td>
<td>91.76</td>
<td>99.52</td>
</tr>
<tr>
<td>20 7 5</td>
<td>1393.10</td>
<td>96.15</td>
<td>99.57</td>
</tr>
<tr>
<td>20 9 5</td>
<td>1374.04</td>
<td>94.62</td>
<td>98.99</td>
</tr>
<tr>
<td>25 5 5</td>
<td>1455.58</td>
<td>94.28</td>
<td>99.03</td>
</tr>
<tr>
<td>25 7 5</td>
<td>1579.44</td>
<td>92.92</td>
<td>99.13</td>
</tr>
<tr>
<td>25 9 5</td>
<td>1713.74</td>
<td>91.49</td>
<td>99.29</td>
</tr>
<tr>
<td>30 5 5</td>
<td>1892.68</td>
<td>97.51</td>
<td>99.86</td>
</tr>
<tr>
<td>30 7 5</td>
<td>2093.57</td>
<td>93.87</td>
<td>99.80</td>
</tr>
<tr>
<td>30 9 5</td>
<td>2153.96</td>
<td>94.75</td>
<td>99.00</td>
</tr>
</tbody>
</table>

The MMKP problems were generated using pseudo-random number generators. We used the following symbols and functions to generate the test instances:

- \( R_c \) = Maximum amount of a resource consumption by an item
- \( P_c \) = Maximum cost of any unit resource
- \( R_i \) = Total amount of the \( i \) th resource = \( n \times R_c \times 0.5 \)
- \( P_i \) = \( \text{rand}(P_c) \) = A uniform random number between 0 and \( P_c \)
- \( r_{ij} \) = The \( j \) th resource of the \( i \) th item of the \( i \) th group = \( \text{rand}(R_c) \)
- For random initialization of item values:

generated 10 random and 10 correlated MMKP instances. We tested BBLP, MOSER and HEU on all 10 instances, and tabulated the averages of solution-value and execution time.
Table 3: Performance comparison of HEU and MOSER with an upper bound for larger correlated data sets.

<table>
<thead>
<tr>
<th>MMKP parameters</th>
<th>solution value</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( l )</td>
<td>( m )</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>130</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>190</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>220</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>250</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>280</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>310</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>340</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>370</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4: Performance comparison of HEU and MOSER with an upper bound for larger random data sets.

<table>
<thead>
<tr>
<th>MMKP parameters</th>
<th>solution value</th>
<th>execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( l )</td>
<td>( m )</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>130</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>190</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>220</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>250</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>280</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>310</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>340</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>370</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

For uncorrelated data: \( v_{ij} = \text{value of the } j \text{th item of the } i \text{th group} = \text{rand}(m \times \frac{R}{2} \times \frac{P}{2}) \times \frac{j+1}{l} \)

SIU 2001
Solving the Knapsack Problem for Adaptive Multimedia Systems

Figure 4.4: Scatter diagram of solution values for 15 correlated data sets with \( n = 200, l = 10 \) and \( m = 10 \).

For correlated data (linear correlation between the resource consumption and item values):

\[
v_{ij} = \sum r_{ijk} \times P_k \times \text{rand}(m \times 3 \times \frac{R_c}{10} \times \frac{P_c}{10})
\]

For the experimental results reported in this article, we used the following values: \( R_c = 10 \) and \( P_c = 10 \). Please see ftp://panoramix.univ-paris1.fr/pub/CERMSEM/hifi/MMKP for some benchmark data sets on the MMKP.

Table 1 and Table 2 show the performance of the algorithms for correlated and random data sets respectively for number of groups ranging from 5 to 30. Here the solution-values of HEU and Moser are normalized with the optimal value (as found by BBLP), and computation times are presented in milliseconds. The computation times for the heuristics are not significant (less than a millisecond) with respect to BBLP. We used the ftime() function of Visual C++ to measure the elapsed time used by the algorithms. This function gives the current time with millisecond accuracy.

Table 3 and Table 4 compare the performance of HEU and MOSER for number of groups ranging from 40 to 400. For this range, the computation times for
optimal solution by BBLP are too large for practical interest because it takes too long (days or years) on the average to do computation. So the solution-values of HEU and MOSER have been normalized by the upper bound.

To verify the consistency of the results we present a scatter diagram showing the upper bounds, and total values from HEU and MOSER for 15 data sets in Figure 4.4 and Figure 4.5.

One can make the following observations from the tables.

- Heuristic HEU produced solutions which are very close to the optimal solutions provided by algorithm BBLP. Table 1 shows that HEU achieved an objective function value within 6% of the optimal value. But in Table 2 we find that HEU achieved a value within 4% of the optimal value in almost every case.
- We also find from Table 1 to Table 4 that for correlated data sets, all algorithms took more time than for randomly generated data sets.
- We can give a plausibility argument of the different behaviour between random and correlated data sets. When the data are fully correlated, the items of a group lie on a straight line. The items with high resource

Figure 4.5: Scatter diagram of solution values for 15 random data sets with $n = 200$, $l = 10$ and $m = 10$. 
consumption and high values are picked first. So the algorithm goes to step 3 when there are significant amounts of resources available. Step 3 tries to improve the solution with an upgrade followed by downgrades. This could potentially be computationally expensive. But if the data sets are random then the picking of items will not be biased (both high and low valued items will be picked with the same probability) and we get nearly optimal solutions with a comparatively large numbers of iterations in step 2 and a few iterations in step 3. When a data set is correlated there is a chance that almost every combination of the picks is feasible. Again, for a random data set some of the picking combinations are infeasible and the search tree can be pruned extensively. It can be concluded that we can get a solution with fewer iterations for random data sets than for correlated data sets.

- The execution time requirements for BBLP in Table 1 and Table 2 show that BBLP is practically infeasible for applications requiring real time decision making such as online admission of customers to networks where execution times of less than 1 second are required.

- Heuristics HEU and MOSER may fail to find a feasible solution where BBLP succeeds. However, for the same set of problem instances in our experiments reported in Table 1 and Table 2, MOSER failed to find a feasible solution in a few cases, while HEU was able to find solutions to all the instances.

- For most instances, HEU produced solution-values which are 1 to 8% better (closer to optimal) than the solution-value produced by MOSER.

- If we observe the solution-value columns of Table 1 to Table 4 carefully we find that on an average HEU required computation times which are 30 to 50% less than that required by MOSER.

- We find some irregularities in the data on computation times. As the computation times required by the heuristic algorithms and BBLP depend on the data set, it may happen that smaller data sets take longer than larger data sets. The time requirements for \( n = 25 \) in Table 1 show this unlikely situation in practice. If the data set consists of very few feasible solutions it might take less time to get a solution.

- In order to find how execution-time grows with the variation of problem size, we have plotted \( t_{H_{eu}} \) with \( n \) using data from table 4, and observed that execution-time grows quadratically with problem size. This is a partial validation of our complexity analysis presented in section 4.3.2.

For our intended applications, such as the AMP, SLA admission, QoS adaptation, etc., typical problems are of medium size that HEU handles.
well. If the problem size grows larger, for example in a large-scale computer network, then a distributed version of HEU may be more suitable. Our on-going research is focusing on a distributed implementation of HEU.

5. Conclusion

We presented a heuristic HEU for near-optimal solution of the MMKP. The heuristic employs an iterative improvement procedure using the concepts of savings in aggregate resource and value per unit of extra aggregate resource, based on Toyoda’s concept of aggregate resource.

Our results show that HEU provides solution values which are within 6% of the optimal value obtained by an exhaustive search technique such as BBLP. Our heuristic out-performs Moser’s heuristic both in solution-value and computation time. HEU also has better success in finding a feasible solution than MOSER.

HEU would be a very good candidate for time-critical applications such as adaptive multimedia systems where a near-optimal solution is acceptable, and fast computation is more important than guaranteeing the truly optimal value. However in applications where optimality of solution cannot be compromised, BBLP would be more appropriate.

We have applied heuristic HEU for an implementation of a simulated adaptive multimedia environment using the Utility Model [Khan98, Chen99]. In this prototype, the solution of the MMKP is used in making session admission and quality adaptation decisions. Currently we are working on a distributed implementation of HEU to enable scalability and fault tolerance.

Acknowledgement. The authors would like to thank the referees of the paper for their valuable remarks and comments.

References


Solving the Knapsack Problem for Adaptive Multimedia Systems


Authors addresses:

Shahadat Khan,
Eyeball.com, Suite 409-100 Park Royal,
West Vancouver BC , Canada
mailto:skhan@eyeball.com

Kin F. Li,
Department of ECE, Universityif of Victoria,
P.O. Box 3055 STN CSC,
Victoria, B.C. V8W 3P6, Canada
mailto:kinli@ieee.org

Eric G. Manning,
ISP, P.Eng., F.IEEE, FEIC
Director, PANDA Laboratory
Computer Science & ECE
University of Victoria
PO Box 3055, Victoria BC V8W 3P6
mailto:Eric.Manning@engr.UVic.ca
Md. Mostofa Akbar,
Department of Computer Science
University of Victoria,
PO Box 3055, Victoria BC V8W 3P6 Canada
mailto:mostofa@csc.uvic.ca