RC 21331 (96102) revised November 10, 1998
Computer Science/Mathematics

IBM Research Report

Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions

M. Dawande[1] J. Kalagnanam[1] P. Keskinocak [1] R. Ravi[2] F. S. Salman [2].

Abstract

Motivated by a real world application, we study the multiple knapsack problem with assignment restrictions (MKAR). We are given a set of items, each with a positive real weight, and a set of knapsacks, each with a positive real capacity. In addition, for each item a set of knapsacks that can hold that item is specified. In a feasible assignment of items to knapsacks, each item is assigned to at most one knapsack, assignment restrictions are satisfied, and knapsack capacities are not exceeded. We consider the objectives of maximizing assigned weight and minimizing utilized capacity.
We focus on obtaining approximate solutions in polynomial computational time. We show that simple greedy approaches yield 1/3-approximation algorithms for the objective of maximizing assigned weight. We give three different 1/2-approximation algorithms: the first one is a modified version of the greedy algorithm, the second one solves single knapsack problems successively and the third one is based on rounding the LP relaxation solution. For the bicriteria problem of minimizing utilized capacity subject to a minimum requirement on assigned weight, we give two approximation algorithms with performance ratios (1/3,2) and (1/2,3).
Keywords: multiple knapsack, approximation algorithms.

IBM Research Division
Almaden · Austin · T.J. Watson · Beijing · Haifa · Tokyo · Zurich

[1]IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York 10598, {milind,jayant,pinar}@watson.ibm.com
[2]GSIA, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, {ravi,fs2c}@andrew.cmu.edu

1 Introduction

In this paper, we study the *Multiple Knapsack Problem with Assignment Restrictions* (MKAR), which is a variant of the well-known *Multiple Knapsack Problem* (MKP) [Kar72], [MT90], [MT80], [MT81a], [HF78], [FMW96].

In MKAR, we are given a set of items $N = \{1, \ldots, n\}$ and a set of knapsacks $M = \{1, \ldots, m\}$. Each item $j \in N$ has a positive real weight w_j and each knapsack $i \in M$ has a positive real capacity c_i associated with it. In addition, for each item $j \in N$ a set $A_j \subseteq M$ of knapsacks that can hold item j is specified. Let $B_i \subseteq N$ be the set of items that can be assigned to knapsack i. We say item j is *admissible* to knapsack i, if $j \in B_i$.

In a feasible assignment of items to knapsacks, for each knapsack $i \in M$, we need to choose a subset S_i of items in N to be assigned to knapsack i, such that:

(1) All S_i's are disjoint. (Each item is assigned to at most one knapsack.)

(2) Each S_i is a subset of B_i, for $i = 1, \ldots, m$. (Assignment restrictions are satisfied.)

(3) $\sum_{j \in S_i} w_j \le c_i$, for $i = 1, \ldots, m$. (Total weight of items assigned to a knapsack does not exceed the capacity of the knapsack.)

Note that in the MKP, $B_i = N$ for all $i \in M$, so that any item can be assigned to any knapsack. The MKAR problem generalizes MKP by allowing assignment restrictions.

The main motivation for the study of the MKAR problem came from inventory matching applications in the steel industry [KDTL97][DK98]. In these applications, production planning involves assigning a given set of orders to the production units in the inventory. Manufacturability considerations such as the compatibility of orders and production units in terms of quality, size, etc. impose additional assignment constraints. Thus, in general, a given order may not be assignable to all the production units. As production operations involve more complex processes and a larger product variety, the problem becomes more constrained.

One of the main objectives for such a problem is to find a feasible assignment with the objective of maximizing assigned weight $\sum_{i \in M} \sum_{j \in S_i} w_j$. In many applications, in addition to maximizing assigned weight, minimizing the total capacity of the utilized knapsacks is also important. We also study MKAR considering the objectives of minimizing utilized capacity $\sum_{i : S_i \neq \emptyset} c_i$ (or equivalently, minimizing *waste*) and maximizing assigned weight simultaneously. We refer to this problem as the *Bicriteria Multiple Knapsack Problem with Assignment Restrictions (BMKAR).* For the objective of minimizing total utilized capacity alone, BMKAR is a variant of the variable-size bin packing problem [CGJ97], [CGJ84], [MT89], [FL86].

Both of MKAR and BMKAR are NP-hard in the strong sense. Therefore we focus on obtaining approximate solutions for these problems in polynomial computation time. We give several algorithms whose solutions are within a small constant times the optimal value for all instances. To the best of our knowledge, these are the first approximation algorithms for these problems.

The paper is organized as follows. In Section 1.1, we define some terms which are frequently used later in the paper and we give a summary of our results in Section 1.2. Next, we present some hardness results for MKAR in Section 2. In Section 3, we present our algorithms and their analyses for MKAR with the objective of maximizing assigned weight. We present our results on the bicriteria problem BMKAR in Section 4. We conclude in Section 5 and give some future research directions. Due to space limitations, all the proofs are presented in the Appendix of this extended abstract.

1.1 Preliminaries

Without loss of generality we assume that $w_j \leq c_i$, $\forall j \in B_i$, otherwise j can be removed from B_i. The problem becomes trivial if all A_j's are disjoint, or if $\sum_{j \in B_i} w_j \leq c_i$, $\forall i \in M$. In the case where all B_i's are disjoint, the problem decomposes into m single 0-1 knapsack problems. Thus, we exclude these cases from consideration.

The assignment restrictions can be represented by a bipartite graph, where the two disjoint node sets of the bigraph correspond to the sets N and M. Let $G = (V, E)$ be the corresponding bipartite graph with $V = N \cup M$. Then, there exist an edge $(i, j) \in E$ between nodes i and j if and only if $j \in B_i$. We define the *density* of a bipartite graph G as the ratio of the number of edges in G to the number of edges in a complete bipartite graph on the vertices of G, i.e. the density of G is equal to $|E|/|N||M|$. We call a graph *sparse*, if its density is low. Note that in the related multiple knapsack problem, the underlying bipartite graph is always complete.

An *α-approximation algorithm* produces a solution with objective function value at most (at least) α times the optimum for all instances, for a minimization (maximization) problem. The factor α is called the *performance ratio* or the *worst case factor* of the algorithm.

1.2 Summary of Results

We first consider the single objective of maximizing assigned weight under assignment restrictions. We show that this problem is NP-hard in the strong sense, even when all knapsack capacities are equal and the density of the underlying bipartite graph approaches zero. We show that simple greedy algorithms have a worst-case factor of 1/3 for MKAR. The analyses for these algorithms are existentially tight. Then we give 3 different 1/2-approximation algorithms. The first one is based on a modified greedy algorithm, the second one on solving single knapsack problems successively,[3] and the third is based on rounding the linear programming (LP) solution. The LP rounding result also implies that the bound obtained from the LP relaxation of the integer programming (IP) formulation is at most twice the optimal IP value.

Next, we consider BMKAR with the objective of minimizing utilized capacity, subject to a lower bound on the assigned weight. We show that it is not possible to have a polynomial algorithm with constant performance ratio, which satisfies the lower bound exactly, unless P=NP.

We present two algorithms, with performance ratios 2 and 3, whose solutions satisfy at least 1/3

[3] A performance ratio arbitrarily close to the claimed ratio can be obtained in polynomial time, if a fully-polynomial time approximation scheme (FPTAS) is employed to solve the single knapsack problems.

and 1/2 of the assigned weight lower bound, respectively. The first algorithm solves single knapsack problems[3] while the second algorithm is a polynomial time greedy algorithm.

2 Complexity

In this section, we show that MKAR is NP-hard in the strong sense, even when all knapsack capacities are equal and the bipartite graph representing the assignment restrictions is very sparse.

Theorem 1 *MKAR is NP-hard in the strong sense, even when all of the knapsack capacities are equal.*

The proof follows by a reduction from the Numerical 3-Dimensional Matching (N3DM) problem.

Next, we show that MKAR is NP-complete, even if the bipartite graph representing the assignment restrictions has arbitrarily low egde density.

Theorem 2 *MKAR problem is NP-hard even if the density of the underlying bipartite graph tends to zero.*

The proof follows by slightly modifying the proof of Theorem 1.

Recently, Caprara [C98] communicated to us that he showed that there exists no PTAS for the MKP even when item weights equal profits. This result implies that there exists no PTAS for MKAR, either.

3 Maximizing Assigned Weight under Assignment Restrictions

In this section, we first show that solutions with a certain property, which we call the *half-full property*, are 1/3-approximations. We present a greedy algorithm framework which produce solutions with the half-full property, i.e. the performance ratio of such algorithms is 1/3. We also show that this performance ratio is existentially tight.

Next, we present three different algorithms all having performance ratio 1/2. The first algorithm is a modified greedy algorithm, the second algorithm solves m single knapsack problems and the third algorithm is based on rounding an optimum vertex solution of the LP relaxation.

3.1 Greedy Approaches

We first characterize a property common to the solutions generated by certain greedy approaches.

Definition 3.1 *A solution for MKAR has half-full property if the following condition holds: For every unassigned item in the solution, any knapsack to which this item is admissible is at least half full.*

We show that any solution with the half-full property has at least one third of the assigned weight of an optimum solution.

Lemma 3 *Any solution with half-full property is a 1/3-approximation.*

Greedy algorithms based on the following framework generate solutions that have the half-full property.

Algorithm Greedy

Sort items by weight in non-increasing order. Sort knapsacks in any order. At each step, assign the next item in the list to the first eligible knapsack (if any) considering capacity and assignment restrictions.

Theorem 4 *Algorithm Greedy has performance ratio 1/3 for MKAR.*

The proof follows by showing that the solutions generated by algorithm Greedy have the half-full property. Note that if we sort the knapsacks in non-decreasing order of capacity in Algorithm Greedy, it is a 1/2-approximation algorithm for MKP with the objective of maximizing assigned weight, whereas its performance ratio is 1/3 for MKAR, which has assignment restrictions. The performance ratio 1/3 is tight, as shown by an example presented in the Appendix.

3.2 Modified Greedy

The performance ratio of the greedy algorithms discussed in Section 3.1 was relatively low, because those algorithms assign items to the "wrong" knapsacks in certain instances. They create assignments, in which the remaining unassigned items are admissible only to the knapsacks that are already filled by the items, which are also admissible to some of the unutilized knapsacks. To overcome this drawback, we modify the greedy algorithm by giving priority to the items which are admissible only to few knapsacks.

Algorithm Modified Greedy:

(1) For each knapsack i, compute remaining admissible weight RA_i.

(2) Pick a knapsack, say knapsack k.

(2.a) If the admissible weight to knapsack k, excluding the admissible weight to the remaining unfilled knapsacks is larger than $c_k/2$, assign as much as you can from that weight to knapsack k greedily (sorting the items by decreasing order of weight).

(2.b) If (2.a) does not hold, then greedily assign as much as you can to knapsack k. Go to step (1).

Theorem 5 *Algorithm Modified Greedy is a 1/2-approximation for MKAR.*

3.3 Solving Single Knapsack Problems

Consider an algorithm that runs through the knapsacks in any order and solves a single knapsack problem for each knapsack to find a maximum weight packing of admissible items. We will call such an algorithm a *successive knapsack* algorithm. The generic algorithm is as follows.

Successive Knapsack Algorithm:

Initialize: $S = N$, $Weight_i = 0$, for $i = 1, \ldots, m$.
For each knapsack i,
Solve a single knapsack problem for knapsack i with item set $S \cap B_i$ to maximize assigned weight.
Let S_i be the set of items packed with total weight $Weight_i$.
Remove S_i from S.

The algorithm requires the solution of m single knapsack problems, which can be solved in pseudo-polynomial time by dynamic programming. Note that instead of solving the single knapsack problems exactly, we can obtain $1/(1+\epsilon)$-approximate solutions for any $\epsilon > 0$ by a FPTAS [IK75]. In this way, we can get $1/(2+\epsilon)$-approxima te solutions to MKAR in fully-polynomial time.

Theorem 6 *Successive Knapsack Algorithm is a 1/2-approximation algorithm for MKAR.*

3.4 LP rounding: A $\frac{1}{2}$ approximation algorithm

The IP formulation of the MKAR problem is as follows:

$$\begin{aligned}
\max \quad & \sum_{i \in M} \sum_{j \in B_i} w_j \, x_{ij} \\
\text{st} \quad & \\
& \sum_{j \in B_i} w_j \, x_{ij} \le c_i, \;\; i \in M \\
& \sum_{i \in A_j} x_{ij} \le 1, \;\; j \in N \\
& x_{ij} \in \{0,1\}, \; i \in A_j, \; j \in N
\end{aligned}$$

where the 0-1 variable x_{ij} denotes whether item j is assigned to knapsack i. Let us denote the LP relaxation by LP-MKAR, which is obtained by replacing the constraints $x_{ij} \in \{0,1\}$ with the constraints $0 \le x_{ij} \le 1$. LP-MKAR can be solved by a maximum flow algorithm.

Let x be an optimal basic feasible solution to LP-MKAR and let f denote the vector of fractional variables of x. We denote the subgraph of the bigraph representation G induced by the edges in f by R_G and call it the *residual graph.*

Lemma 7 *The residual graph, R_G, of the LP relaxation is a forest.*

We can consider each connected component of R_G separately. Let OPT denote the optimum value of LP-MKAR and let $AW(K_i)$ denote the total weight assigned to knapsack K_i in x. In R_G, if a knapsack has degree 1, then we refer to that knapsack as a *singleton* knapsack.

Lemma 8 *If R_G has a singleton knapsack node, say K_1, then there is a rounding step for knapsack K_1 such that $R_1 \ge OPT - 1/2AW(K_1)$, where R_1 denotes the objective function value of the solution obtained after the rounding step.*

Note that after the rounding step for knapsack K_1, node K_1 is removed from R_G and the new residual graph is still a forest, as the rounding step does not add any new edges to R_G. We call the above rounding step a "type 1" step.

Next we define a second type of rounding step: a "type 2" step.

Lemma 9 *If R_G has no singleton knapsack node, then the fractional solution corresponding to R_G can be perturbed such that (i) the objective function value does not change and (ii) the number of arcs in R_G reduces by at least 1.*

Now using the two types of rounding steps, we have the following rounding scheme.

LP Rounding Algorithm:

Step 0: Obtain a vertex solution x to the LP-MKAR. Construct the residual graph R_G based on the fractional variables.
Step 1: If there exists a singleton knapsack node in R_G, perform a type 1 rounding. Update R_G.

Repeat Step 1 until there are no singleton knapsacks in R_G.

Step 2: Pick a knapsack node, say K, of degree at least 2. Perform a type 2 rounding. Update R_G.

Go to Step 1.

Theorem 10 *LP Rounding Algorithm is a 1/2-approximation algorithm for MKAR.*

Corollary 11 *The LP-IP integrality gap for MKAR is 2. That is, the optimal objective value of the LP relaxation for the MKAR is at most 2 times the optimal objective value of MKAR.*

The IP-LP gap of 2 is tight, as shown by the following simple instance. Consider the instance with one knapsack of capacity $2M$, and only 2 items of weight $M + \epsilon$ and M. The optimum IP solution has assigned weight $M + \epsilon$, whereas optimum value of the LP relaxation is $2M$.

4 The Bicriteria Problem

In the surplus inventory matching application that motivated our study of the MKAR problem, in addition to maximizing assigned weight, minimizing total utilized capacity (or equivalently, total unused capacity of the utilized knapsacks) is also an important objective that needs to be considered. With the two objectives, the choice of the knapsack to which an item is assigned becomes more critical. As the problem gets sparser, any solution that maximizes assigned weight does not necessarily utilize small capacity.

In approximating a bicriteria problem, it is common practice to choose one of the criteria as the objective and to bound the value of the other by means of a constraint. In this section, we consider BMKAR with the objective of minimizing total utilized capacity (total capacity of all the bins that are at least partially filled), subject to a lower bound T on the assigned weight. It is easy to construct an instance of this capacity minimization problem (CM) for which the algorithms we have presented so far fail to perform well in terms of minimizing utilized capacity.

One natural question to ask is: what is the best possible performance guarantee of an algorithm for the objective of minimizing utilized capacity, if the weight constraint must be satisfied exactly? The following theorem shows that the best possible performance ratio for any polynomial time algorithm is $\log\ n$.

Theorem 12 *Any algorithm which satisfies the assigned weight constraint cannot have a performance ratio better than* $\log\ n$ *(where n is the number of items) for BMKAR with the objective of minimizing utilized capacity, unless unless P=NP.*

The proof is by a reduction from the Set Cover problem.

In the remaining part of this section, we present two algorithms, which relax the assigned weight constraint in order to minimize the utilized capacity. An (α, β)-*approximation algorithm* for CR generates solutions with utilized capacity at most β times the optimum capacity and with assigned weight at least α times the lower bound T, where $\beta \geq 1$ and $0 < \alpha \leq 1$. The algorithms we present are (1/3,2)- and (1/2,3)-approximation algorithms.

4.1 Solving Single Knapsack Problems

We use the successive knapsack algorithm of subsection 3.3 with a specific minimum ratio rule to select the next knapsack.

Given a set $S \subseteq N$ of items, let $Weight_i$ denote the maximum weight of items from S that can be packed into knapsack i. Note that $Weight_i$ can be found by a pseudo-polynomial algorithm that solves a single knapsack problem (or can be approximated by a FPTAS). Let us denote the unused capacity, or waste, in this knapsack by $Waste_i = c_i - Weight_i$. Let AW denote the total assigned weight by the following algorithm.

Selective Successive Knapsack (SSK) Algorithm:

Initialize: $S = N$, $R = M$, $Weight_i = 0$, $i \in M$, $AW = 0$.

Phase 1:

(1) For all $i \in R$, calculate $Weight_i$ and $Waste_i$ by solving a single knapsack problem for knapsack i with item set $S \cap B_i$.

(2) Pick the knapsack with minimum ratio of $Waste_i/Weight_i$, say knapsack k.

(3) If $Weight_k + AW \geq T/3$, terminate Phase 1.

(4) Pack items into knapsack k to obtain $Weight_k$, add $Weight_k$ to AW.

(5) Remove assigned items from S and knapsack k from R.

(6) Go to step (1), if R is nonempty.

Phase 2:

(1) Sort knapsacks in R in non-decreasing order of capacity.

(2) Pick the next knapsack, say k.

(3) If $Weight_k + AW \geq T/3$, pack items from $S \cap B_k$ to obtain $Weight_k$, update AW and stop.

If the SSK algorithm ends with $AW < T/3$, then we can conclude that there exists no assignment of value at least T for CM. Since the algorithm is a successive knapsack algorithm, by Theorem 6, we should be able to assign at least $T/2$, if there is any feasible solution with assigned weight T or more. If the algorithm outputs $AW \geq T/3$, in order to make sure that the capacity of the knapsack picked in Phase II is small enough, we run the algorithm by excluding the largest capacity knapsack. We repeat the runs by excluding the $(k-1)$ largest capacity knapsacks in the kth run. We stop the runs when AW gets smaller than $T/3$. Among the solutions output over all runs, the minimum capacity solution is picked.

Theorem 13 *If the CM problem is feasible, a (1/3,2)-approximate solution is obtained by running the selective successive knapsack algorithm at most m times.*

4.2 Bicriteria Modified Greedy

In this section, we present a greedy algorithm which selects a knapsack with large admissible weight and small capacity at each step. Similar to the SSK algorithm, the algorithm has two phases. In the first phase, we assign items to knapsacks such that the total assigned weight at the end of Phase 1 is close to, but does not exceed half of the target weight. In the second phase, we choose one more knapsack and assign items to it, such that the the total assigned weight at the end of the second phase is at least half of the target weight.

Algorithm Bicriteria Modified Greedy:

Phase 1:
(1) Apply Algorithm Modified Greedy by picking the knapsack with the largest RA_k/c_k at Step 2, until the total assigned weight exceeds $T/2$ or until no more items can be assigned.
(2) Remove the items assigned to the last knapsack and go to Phase 2.

Phase 2:
(1) Sort the remaining knapsacks in non-decreasing order of capacity.
(2) Pick the next knapsack, say k.
(3) Greedily assign items to knapsack k.
(4) If the total assigned weight exceeds $T/2$, stop; otherwise go to Step 2.

In Section 3.2 we have shown that Algorithm Modified Greedy can assign at least half of the maximum possible assigned weight. Hence, if the total assigned weight does not exceed $T/2$ at the end of Phase 1 of Algorithm Bicriteria Modified Greedy, this means that an assignment with total weight T is not possible, i.e. the CM problem is infeasible.

In the analysis of the algorithm, we first show that the total capacity used in the first phase is at most twice the capacity in an optimum solution.

Lemma 14 *The capacity used in the first phase of the algorithm is at most twice the optimum capacity needed to assign weight T.*

We can guarantee that the capacity of the knapsack picked in Phase 2 of Algorithm Modified Greedy is at most the optimum capacity, by running the algorithm m times as discussed in subsection 4.1. Then, using Lemma 14, we get the following theorem.

Theorem 15 *Algorithm Modified Bicriteria Greedy assigns at least half of the target weight and uses at most three times of the optimum capacity. Hence, it is a (1/2,3)-approximation.*

5 Conclusions

In this paper, we generalized the classical multiple knapsack problem by allowing assignment restrictions. For the objective of maximizing assigned weight, we presented a 1/3-approximation algorithm and three 1/2-approximation algorithms. Whether a performance ratio better than 1/2 can be achieved is an interesting open question.

For the bicriteria problem, where we consider minimizing utilized capacity as well as maximizing assigned weight, we give two approximation algorithms with performance ratios (1/3,2) and (1/2,3), respectively. Again, it would be interesting to improve these performance ratios.

References

[AMO93] A.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, New Jersey, 1993.

[AS 97] S. Aurora and M. Sudan. Improved low degree testing and its applications. Proc. 29th ACM Annual Symp. on Theory of Computing, 485-495, 1997.

[C98] A. Caprara. Personal communication, 1998.

[CGJ84] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49 – 106. Springer-Verlag, Wien, 1984.

[CGJ97] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing: A survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-har d Problems*, pages 46 – 93. PWS Publishing Company, Boston, 1997.

html.

[DK98] M. W. Dawande and J. R. Kalagnanam. The multiple knapsack problem with color constraints. Technical Report, RC21138, IBM T. J. Watson Research Center, Yorktow n Heights, NY, 1998.

[F96] U. Feige. A threshold of $\ln n$ for approximating set cover. Proc. $28^t h$ annual ACM symposium on theory of computing, 314-318, 1996.

[FMW96] C.E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM J. Optimization*, 6(3):858 – 877, 1996.

[FL86] D.K. Friesen and M.A. Langston. Variable sized bin packing. *SIAM J. Computing*, 15:222 – 230, 1986.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability:A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, 1979.

[HF78] M.S. Hung and J.C. Fisk. An algorithm for 0-1 multiple knapsack problems. *Naval Res. Logist. Quarterly*, 24:571–579, 1978.

[IK75] O.H. Ibarra and C.E. Kim. Fast approximation for the knapsack and sum of subset problems. *J. ACM*, 22:463 – 468, 1975.

[Kann91] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Process. Lett.*, 37: 27-35, 1991.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85 – 103. Plenum Press, New York, 1972.

[KDTL97] J. Kalagnanam, M. Dawande, M. Trumbo, and H.S. Lee. The surplus inventory matching problem in the process industry. Technical Report RC21071, IBM T. J. Watson Research Center, Yorktown

Heights, NY, 1997.

[MT80] S. Martello and P. Toth. Solution of the zero-one multiple knapsack problem. *Euro. J. Oper. Res.*, 4:322–329, 1980.

[MT81a] S. Martello and P. Toth. A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Math.*, 3:275–288, 1981.

[MT81b] S. Martello and P. Toth. Heuristic algorithms for the multiple knapsack problem. *Computing*, 27:93–112, 1981.

[MT89] S. Martello and P. Toth. *Knapsack Problems.* John Wiley and Sons,Ltd., New York, 1989.

[MT90] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Math.*, 28:59–70, 1990.

[RS 97] R. Raz and S. Safra, A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. Proc. 29th Annual ACM Symp. on Theory of Computing, 314-318, 1997.

6 Appendix

Proof of Theorem 1 We use a reduction from Numerical 3-Dimensional Matching (N3DM).

Numerical 3-Dimensional Matching (N3DM)

INSTANCE: Integers n, d and a_i, b_i, c_i, $i = 1, \ldots, n$ satisfying

$$\sum_i (a_i + b_i + c_i) = nd \text{ and } 0 < a_i, b_i, c_i < d \text{ for } i = 1, \ldots, n.$$

QUESTION: Are there permutations ρ and σ of $\{1, \ldots, n\}$, such that

$$a_i + b_{\rho(i)} + c_{\sigma(i)} = d \text{ for } i = 1, \ldots, n.$$

This problem is NP-complete in the strong sense (Garey and Johnson [GJ79]).

Given an instance I of N3DM, construct the following instance I' of MKAR. For each a_i, b_i, c_i in N3DM, there is an item of weight a_i, $b_i + d$, $c_i + 2d$, respectively, in MKAR. There are n knapsacks in MKAR with capacity $4d$ each. An item of weight a_i can be assigned to knapsack i only, $i = 1, \ldots, n$. The remaining items can be assigned to any knapsack (see Figure 1(a)). We will show that there exists an assignment of total weight $4nd$ for I' if and only if there exists a 3-dimensional matching for I.

First, suppose that there exists an assignment of weight $4nd$ for I'. In this assignment (1) all items must be assigned, (2) each knapsack must contain exactly one c item, since no two c items can be put to the same knapsack, (3) each knapsack must contain exactly one a item because of assignment restrictions, and hence (4) each knapsack must contain exactly one b item. Items in each knapsack correspond to a triple of a matching.

Similarly, given a matching for I, one can easily construct an assignment for I', in which all the items are assigned. Simply assign items $i, \rho(i)$ and $\sigma(i)$ to knapsack i. □

Proof of Theorem 2 Given an instance of N3DM, we modify the instance I' of MKAR defined in the proof of Theorem 1 to obtain instance I''. We add knapsacks $n+1, \ldots, n+k$, and items $1, \ldots, k+1$, where k is an arbitrary positive integer. Each new knapsack has unit capacity. The first new item has weight $1/nd$ and can be assigned to knapsacks n and $n+1$. Items 2 through $k+1$ have unit weight. New knapsack $n+i$ can only hold new items i and $i+1$, for $i = 1, \ldots, k$ (see Figure 1 (b)). Note that the bipartite graph representation of I'' is connected. It is easy to show that there exists a solution to N3DM if and only if there is an assignment of value $4nd + k$ for I''.

Given an assignment of value $4nd + k$, new items must have been assigned to new knapsacks, hence the arguments in the proof of Theorem 1 are still valid. A complete bipartite graph with the item and knapsack sets of I'' has $(3n + k + 1) * (n + k)$ edges. On the other hand, in the bipartite graph representation of I'', there are $2n^2 + n + 2k$ edges. As k gets arbitrarily large (in particular, an order of magnitude larger than n), the edge density of the bigraph representation of I'' will be $O(2/k)$, which is the order of minimum density for a connected bigraph. Hence, there are very sparse instances of MKAR which are as hard as the N3DM problem. □

Proof of Lemma 3 Consider a solution with the half-full property. Let U be the set of items not assigned to any knapsack and A_U be the set of knapsacks eligible to accommodate at least one of

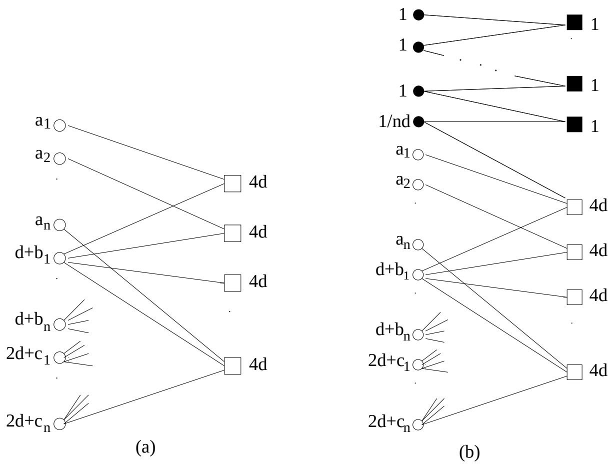

Figure 1: Instances of MKAR

the items in U, i.e. $A_U = \cup_{j \in U} A_j$. Let $C(A_U)$ be the total capacity of the knapsacks in A_U and $W(A_U)$ be the total weight of the items assigned to the knapsacks in A_U in the solution. Let W be the total assigned weight. By the half-full property, at least half of the capacity of any knapsack in A_U is filled in the solution, i.e. $C(A_U) \leq 2W(A_U)$.

In an arbitrary feasible solution, let $U' \subseteq U$ be the set of items which are assigned to knapsacks and $W(U')$ be the total weight of these items. In any solution, including the optimum solution, $W(U') \leq C(A_U)$ since items in U' are admissible only to knapsacks in A_U. Thus, in any solution the maximum weight one could obtain by assigning items in U to the knapsacks is $C(A_U)$. The total weight of the remaining items, i.e. the items in $N - U$, is W. Hence, $W + C(A_U)$ is an upper bound on the total weight of any solution.

Since $W + C(A_U) \leq W + 2W(A_U) \leq 3W$, the result follows.

Proof of Theorem 4 The proof follows by showing that the solutions generated by algorithm Greedy have the half-full property. Consider a solution generated by algorithm Greedy. In that solution, let U be the set of items not assigned to any knapsack and A_U be the set of knapsacks eligible to accommodate at least one of the items in U. Pick any item $j \in U$. All the knapsacks in A_j (i.e. the knapsacks to which item j is admissible) must be (partially) filled; otherwise, item j would be assigned. Suppose there is a knapsack $i \in A_j$, which has more than half of its capacity c_i free. Since item j is unassigned, $w_j > c_i/2$, and by the greedy choice of the algorithms, there must be another item k with $w_k \geq w_j > c_i/2$, which has already been assigned to knapsack i. In this case, again at least half of the capacity of knapsack i is filled.

Proof of Theorem 5 Based on the assignment generated by the algorithm, let us partition the knapsacks as follows:
Type 1. Knapsacks which are assigned weight in Step (2.a)
Type 2. Knapsacks which

are assigned weight in Step (2.b) and are at least half full.
Type 3. Knapsacks which are assigned weight in Step (2.b) and are less than half full.

Let C_t be the total capacity and W_t be the total assigned weight of type t knapsacks, $t = 1, 2, 3$.

Let $W = W_1 + W_2 + W_3$ denote the total weight assigned by Algorithm Modified Greedy and let W^* denote the weight assigned in an optimum solution.

Let W_2' be the total weight of the remaining unassigned items, which are admissible to type 2 knapsacks. (Note that any remaining unassigned item is admissible only to type 1 or type 2 knapsacks.)

The proof follows from the following claims.

Claim 1: The weight assigned in an optimum solution is at most $C_1 + W_2 + W_2' + W_3$.

Claim 2: $W_2' \leq C_2/2$.

Claim 3: The total weight assigned by modified greedy is at least $C_1/2 + W_2 + W_3$.

So, the ratio W/W^* is at least

$$\frac{C_1/2 + W_2 + W_3}{C_1 + W_2 + W_2' + W_3} = \frac{C_1 + 2W_2 + 2W_3}{2(C_1 + W_2 + W_2' + W_3)} \geq \frac{C_1 + 2W_2 + 2W_3}{2(C_1 + 2W_2 + W_3)} \geq \frac{1}{2}$$

because $W_2 \geq C_2/2 \geq W_2'$.

Proof of Claim 1: The maximum weight that can be assigned to type 1 knapsacks is at most C_1. Any remaining unassigned items can only be assigned to type 2 knapsacks, and the total weight of those items is at most W_2'.

Proof of Claim 2: At the j-th execution of Step (2.b), the unassigned weight admissible to the knapsack under consideration, say knapsack k, excluding the admissible weight to all the remaining knapsacks is less than $c_k/2$.

Proof of Claim 3: Type 1 knapsacks are at least half full since items of total weight at least half the capacity are assigned greedily.

The following example shows that if we sort the knapsacks in non-decreasing order of capacity in Algorithm Greedy, its performance ratio of 1/3 is tight for MKAR.

Example 1: There are three items with weights $M + \epsilon$, M and M, and two knapsacks with capacities $2M$ and $M + \epsilon$. Item 1 could be assigned to either of the knapsacks whereas items 2 and 3 can only be assigned to knapsack 1. Algorithm Greedy would assign item 1 to knapsack 1 (no other assignments) with total assigned weight $M + \epsilon$. In the optimum solution, item 1 is assigned to knapsack 2 and items 2 and 3 are assigned to knapsack 1, with total weight $3M + \epsilon$.

Proof of Theorem 6 Let AW denote the total weight assigned by the algorithm, i.e. $AW = \sum_{i \in M} Weight_i$. Let S be the set of items remaining unassigned at the end of the algorithm. The proof follows from the following claim. In any feasible solution to MKAR, total weight of items from S that are assigned is at most AW. To prove this claim consider any feasible solution x. Suppose the weight of items in S that are assigned in this solution equals $AW + \epsilon$ for some $\epsilon > 0$. Then in solution x, there is at least one knapsack, say i, with weight greater than $Weight_i$ assigned to it from S. However, this is a contradiction because when we filled knapsack i in the algorithm, we used an item set including S, and chose a maximum weight assignment of value $Weight_i$. Thus, any feasible solution can have total assigned weight at most $2AW$.

Proof of Lemma 7 Suppose R_G contains a cycle, say C. N ote that C has to be even. Let $C = I_1, K_1, I_2, K_2, \ldots, K_l, I_1$, where I_j are item nodes (with weight w_j) and K_i are knapsack nodes in the cycle. Let $f_C = (f_1, \ldots, f_k)$ denote the vector of fractional values on the edges of C, when C is traversed in the above order. Keeping all other values fixed, we can perturb the variables on this cycle such that x is a convex combination of two feasible solutions, which is a contradiction. The two feasible solutions x^1 and x^2 are obtained by adding the vectors f^1 and f^2 to f_C, respectively. The first perturbation vector is $f^1 = (\varepsilon, -\frac{w_1}{w_2}\varepsilon, \frac{w_1}{w_2}\varepsilon, \ldots, -\varepsilon)$ and the second one is $f^2 = -f^1$. The solutions x^1 and x^2 are feasible to LP-MKAR since the perturbations preserve both the capacity and the assignment constraints. In addition, x is a convex combination of x^1 and x^2 since $x = 1/2(x^1 + x^2)$.

Proof of Lemma 8 Let S denote the set of items which have been assigned (integrally) to knapsack K_1 by the solution x. Note that S is nonempty, otherwise there would be no fractional variable incident on K_1. Let $W(S)$ denote the total weight of items in S. In addition, let I_1 be the item which is fractionally assigned to knapsack K_1 with value f_1. If $W(S) \geq w_1 f_1$, then we round f_1 down to zero. Now K_1 has no fractional variable incident on it. Let us denote the new solution by x_1. Then, x_1 has objective value $R_1 = OPT - w_1 f_1$. Since $AW(K_1) = w_1 f_1 + W(S) \geq 2w_1 f_1$, it is true that $R_1 \geq OPT - 1/2AW(K_1)$. On the other hand, if $w_1 f_1 > W(S)$, then we unassign all items in S and round f_1 to 1. We also round all other fractional variables incident on I_1 to 0. Then, $R_1 \geq OPT - W(S) \geq OPT - 1/2AW(K_1)$.

Proof of Lemma 9 Suppose there exists no singleton knapsack at the current bigraph R_G. Consider a knapsack node K of degree at least 2. Let e_1 and e_2 be two edges incident on K. Let P_1 and P_2 be the two longest paths originating from node K and starting with edges e_1 and e_2, respectively. These paths must be disjoint since R_G has no cycles. Each of these paths must have an item node as its endpoint, for otherwise, we would have a singleton knapsack node. The key observation is that for any of these paths, the leaf node has an assignment constraint which is not tight. Along each path we can perturb the solution such that we obtain a feasible solution having one less fractional variable and the same objective value. The perturbations are as follows. Let $f^1 = (f_1, \ldots, f_k)$ denote the vector of fractional values on the edges of P_1 and let $I_1, \ldots, I_k$ be the item nodes of P_1. Similarly, let $f^2 = (f_{k+1}, \ldots, f_p)$ denote the fractional vector and $I_{k+1}, \ldots, I_p$ denote the items of P_2. We modify the current solution x in f^1 and f^2 by adding the vectors f^1_ε and f^2_ε, respectively. The first perturbation vector is $f^1_\varepsilon = (\frac{w_k}{w_1}\varepsilon, -\frac{w_k}{w_1}\varepsilon, \frac{w_k}{w_2}\varepsilon, \ldots, -\frac{w_k}{w_{k-1}}\varepsilon, \varepsilon)$ and the second one is $f^2_\varepsilon = (-\frac{w_k}{w_{k+1}}\varepsilon, \frac{w_k}{w_{k+1}}\varepsilon, -\frac{w_k}{w_{k+2}}\varepsilon, \ldots, \frac{w_k}{w_{p-1}}\varepsilon, -\frac{w_k}{w_p}\varepsilon)$. (See Figure 2). Note that the new solution is feasible to LP-MKAR since the perturbations preserve both the capacity and the assignment constraints. Now we can increase ε until one of the fractional values hits 0 or 1. This new solution will have at least one less fractional variable. Thus, at least one edge is removed from R_G. It is easy to check that the perturbations do not change the objective function value. Therefore, a new feasible solution with the same objective value is obtained.

Proof of Theorem 10 After every type 1 rounding step, a knapsack node is removed from R_G. After every type 2 rounding step, at least one edge is removed from R_G. Therefore, the rounding algorithm has at most $mn+n$ iterations each of which takes at most $O(mn)$ time. The LP relaxation can be solved by a maximum flow algorithm and a vertex solution can be found in polynomial time (Ahuja, Magnanti and Orlin [AMO93]).

After any type 2 step, the objective value does not change (Lemma 9). Let R_k denote the objective function value after kth type 1 step. Suppose we pick knapsack K_i at the ith type 1 step. Then,

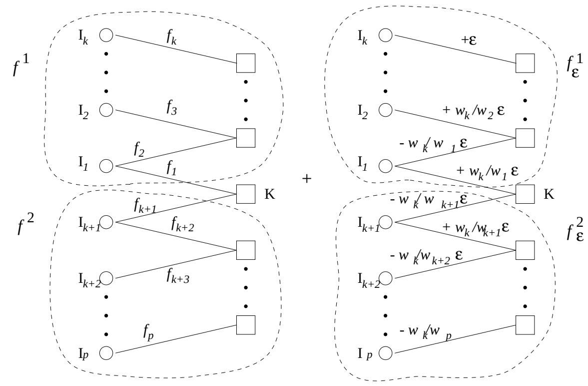

Figure 2: Perturbation of the fractional variables

due to Lemma 8 we have $R_k \geq OPT - 1/2(\sum_{i=1}^{k} AW(K_i))$. Let p denote the last type 1 step. Note that $p \leq m$. Then,

$$R_p \geq OPT - 1/2(\sum_{i=1}^{p} AW(K_i)) \geq OPT - 1/2OPT = 1/2OPT.$$

Hence, the feasible integral solution output by the algorithm has objective value at least half of the LP relaxation value.

Proof of Corollary 11 Given an instance I of MKAR, let $LP(I)$ denote the optimal value for the LP relaxation, and let $OPT(I)$ denote the optimal value for MKAR. In addition, let $H(I)$ denote the value of the solution output by the rounding algorithm for instance I. By Theorem 10, $1/2LP(I) \leq H(I)$. Since $H(I) \leq OPT(I)$, it follows that $LP(I) \leq 2OPT(I)$, for any instance I.

Proof of Theorem 12 The reduction is from the *minimum set cover*. Given an instance of the minimum set cover, i.e. a set S, with $|S| = n$ and a collection of subsets $C_1, C_2, \ldots, C_k$ of S, construct an instance of BMKAR as follows.

For each element of S, create an item of weight $1/n$. For each subset C_i, create a knapsack with capacity 1. Items corresponding to elements of C_i are admissible to this knapsack. Let the minimum weight we want to assign equal $T = 1$.

The minimum capacity needed to assign weight 1 is equal to the minimum number of knapsacks to assign all items, which is equal to the cardinality of the minimum set cover. However, minimum set cover is not approximable within $(1 - \epsilon) \log n$ for any $\epsilon > 0$ unless P=NP [AS 97][RS 97][F96].

Proof of Theorem 13 Suppose the problem is feasible. Consider the run which output the solution with minimum capacity. This solution has AW at least $T/3$. We need to show the bound on utilized capacity.

Consider a solution x_H output by the algorithm and an optimal solution x_{OPT}. Modify x_{OPT} to obtain solution $x_{OPT'}$ as follows. Let OPT be the set of knapsacks used by the optimal solution. Let H be the set of knapsacks picked in Phase I of the algorithm. Remove from OPT all knapsacks that are also in H. Among items assigned to remaining knapsacks of OPT, unassign those assigned to H in x_H.

Claim 1: Weight of items removed from OPT is at most $2T/3$.

Proof of Claim 1: Let W_H denote the weight assigned to H in x_H. Clearly, weight of items assigned in both x_H and x_{OPT} is at most W_H. Let S be the set of items that are assigned to a knapsack in $H \cap OPT$ in x_{OPT} but are not assigned anywhere in x_H. We need to bound the weight of S. Let S_i be the items from S assigned to knapsack $i \in H$ in x_{OPT}. These items were available when knapsack i was filled by the algorithm. Therefore, weight of S_i can be at most $Weight_i$. Thus, weight of S is at most W_H. By the stopping condition of Phase 1, $W_H < T/3$. Hence, total weight removed from OPT is at most $2W_H < 2T/3$.

For a knapsack subset I, let $C(I)$ denote $\sum_{i \in I} c_i$. For a solution x, let $AW(x)$ denote total weight assigned in x and $Waste(x)$ denote the total waste in x. The following are true.

1) $C(OPT') \leq C(OPT)$ since $OPT' \subseteq OPT$.
2) $AW(x_{OPT'}) \geq T/3$ because of Claim 1 and $AW(x_{OPT}) \geq T$.
3) $Waste(x_{OPT'}) + T/3 \leq C(OPT')$ by the definition of waste and 2).

For any knapsack $i \in H$,

$$\frac{Waste_i}{Weight_i} \leq \frac{Waste(x_{OPT'})}{AW(x_{OPT'})} \leq \frac{Waste(x_{OPT'})}{T/3}.$$

The first inequality follows since the ratio of a knapsack picked by the algorithm is smaller than the ratio of knapsacks in OPT' when that choice was made. The average ratio of knapsacks in OPT' is at least as big as the individual ratios when the choice was made, since some items have been removed in OPT'. The second inequality follows directly from 2). By summing these inequalities over i's, we obtain

$$\sum_{i \in H} Waste_i \leq \frac{Waste(x_{OPT'})}{T/3} \sum_{i \in H} Weight_i.$$

Since $\sum_{i \in H} Weight_i \leq T/3$,

$$\sum_{i \in H} Waste_i \leq Waste(x_{OPT'}).$$

Using this inequality together with 1) and 3), we get

$$C(H) = \sum_{i \in H} Waste_i + \sum_{i \in H} Weight_i \leq Waste(x_{OPT'}) + T/3 \leq C(OPT') \leq C(OPT).$$

We can also bound the capacity of the knapsack picked in Phase 2 by $C(OPT)$. The reason is as follows. Let p be a knapsack in OPT with maximum capacity. There is a run of the algorithm which excludes knapsacks of capacity greater than c_p. In that run, the capacity of the knapsack picked in Phase 2 is at most the capacity of any knapsack in OPT. Since the run with minimum capacity is chosen capacity utilized by x_H is at most twice the capacity utilized by x_{OPT}.

Proof of Lemma 14 Based on the assignment in Phase 1, let us partition the knapsacks into four types:
Type 1. Knapsacks which are assigned weight in Step (2.a).
Type 2. Knapsacks which are assigned weight in Step (2.b) and are at least half full.
Type 3. Knapsacks which are assigned weight in Step (2.b) and are less than half full.
Type 4. Knapsacks which are not utilized.

Let S_t be the set of type t knapsacks, W_t be the total weight assigned to, C_t be the total capacity and k_t be the number of type t knapsacks, $t = 1, \ldots, 4$. Let $W = W_1 + W_2 + W_3$.

Claim 1: Type 3 knapsacks (if any) are the last k_3 knapsacks filled in Phase 1.

This is true because the algorithm chooses the knapsacks in non-increasing order of RA_i/c_i.

Let W_2' be the total weight of the items, which are admissible to type 2 knapsacks, but either not assigned at all or assigned to type 1 knapsacks.

Claim 2: $W_2' \leq C_2/2$.

Proof of Claim 2: At any execution of Step (2.b), the remaining weight admissible to the knapsack under consideration, say knapsack i, not admissible to the remaining knapsacks, is less than $c_i/2$.

Claim 3: Any item assigned to a type 1 knapsack is not admissible to type 3 or 4 knapsacks.

Proof of Claim 3: When an item is assigned to a type 1 knapsack, that item is admissible only to the knapsacks that are considered before. By Claim 1, type 3 and type 4 knapsacks are not considered until all type 1 and type 2 knapsacks are assigned, so the items assigned to type 1 knapsacks are not admissible to type 3 and type 4 knapsacks.

Let $A(S)$ be the total weight admissible to the knapsacks in set S (and possibly to other knapsacks as well). Let $RA\{1,2\}(S)$ denote the total remaining weight admissible to the knapsac ks in set S after type 1 and type 2 knapsacks are assigned by the heuristic.

Claim 4: If $k_3 > 0$, then for any subset S of type 3 and type 4 knapsacks, $A(S_2 \cup S) \leq 2W_2 + RA\{1,2\}(S)$.

Proof of Claim 4: From claims 2 and 3, we have $A(S_2 \cup S) \leq W_2 + W_2' + RA\{1,2\}(S)$. To see why this is true, let us partition the admissible weight as follows: $A(S_2 \cup S) = W(S_2 - S) + W(S - S_2) + W(S_2 \cap S)$, where

$W(S_2 - S)$: total weight admissible to S_2 but not S.
$W(S - S_2)$: total weight admissible to S but not S_2.
$W(S_2 \cap S)$: total weight admissible to both S_2 and S.

We further partition these weights based on the type of knapsacks these items are assigned to. We define the following.

$W21$: total weight of the items assigned to type 1 knapsacks, which are also admissible to type 2 knapsacks, but not to the knapsacks in S.
$W20$: total weight of the remaining unassigned items, which are admissible to type 2 knapsacks.
$WS1$: total weight of the items assigned to type 1 knapsacks, which are also admissible to the knapsacks in S, but not to type 2 knapsacks.
$W(2S)1$: total weight of the items assigned to type 1 knapsacks, which are admissible to type 2 knapsacks and to the knapsacks in S.
$W(2S)2$: total weight of the items assigned to type 2 knapsacks, which are also admissible to knapsacks in S.
$W22$: total weight of the items assigned to type 2 knapsacks, which are not admissible to the knapsacks in S.

$W(2S)S$: total weight of the remaining unassigned items admissible to the knapsacks in S, which are also admissible to type 2 knapsacks.
WSS: total weight of the remaining unassigned items admissible to the knapsacks in S, which are not admissible to type 2 knapsacks.

From Claim 3, we know that $WS1$ and $W(2S)1$ are zero, so we have $W(S - S_2) \leq WSS$ and $W(S_2 - S) \leq W21 + W20 + W22$. We also have $W(S_2 \cap S) \leq W(2S)2 + W(2S)2$.

Summing up both sides of the three inequalities above, we get

$$A(S_2 \cup S) \leq W21 + W20 + W_2 + WSS + W(2S)S \leq 2W_2 + RA\{1,2\}(S)$$

since $W21 + W20 = W_2' \leq C_2/2 \leq W(2)$ by Claim 2, and $WSS + W(2S)S = RA\{1,2\}(S)$.

Claim 5: If $k_3 > 0$, maximum weight one can assign to type 1 and type 2 knapsacks and to a subset S of type 3 and type 4 knapsacks in any solution i s at most $C_1 + 2W_2 + RA\{1,2\}(S)$.

Proof of Claim 5: Maximum weight one can assign to type 1 knapsacks is C_1. Maximum weight one can assign to type 2 knapsacks and S is at most $A(S_2 \cup S) \leq 2W_2 + RA\{1,2\}(S)$ by Claim 4.

Corollary 5: If $k_3 > 0$, then the maximum weight one can assign to type 1, type 2 and type 3 knapsacks in any solution is at most $C_1 + 2W_2 + W_3 \leq 2W - W_3 \leq T - W_3$.

Let S_t^*, $t = 1,2,3,4$ denote the subset of type t knapsacks used by an optimum solution and let C^* denote the total capacity of those knapsacks. Clearly, S_4^* is not empty, if $k_3 > 0$ (from Corollary 5). Let $W^*(S)$ denote the weight assigned to the knapsacks in set S by the optimum solution.

Let $RA\{1,2,3\}(S)$ be the remaining admissible weight to a subset S of type 4 knapsacks after the type 1,2,3 knapsacks are filled by algorithm modified greedy.

Claim 6: If $k_3 > 0$, $RA\{1,2,3\}(S_4) \geq W_3$.

Proof of Claim 6: From Claim 5,

$$T \leq W^*(S_1^*) + W^*(S_2^*) + W^*(S_3^* \cup S_4^*) \leq C_1 + 2W_2 + RA\{1,2\}(S_3^* \cup S_4^*).$$

We also have $T \geq C_1 + 2W_2 + 2W_3$ from Corollary 5. Hence,

$$C_1 + 2W_2 + 2W_3 \leq W^*(S_1^*) + W^*(S_2^*) + W^*(S_3^* \cup S_4^*) \leq C_1 + 2W_2 + RA\{1,2\}(S_3^* \cup S_4^*)$$

and this implies $RA\{1,2\}(S_3^* \cup S_4^*) \geq 2W_3$. Furthermore, we have

$$2W_3 \leq RA\{1,2\}(S_3^* \cup S_4^*) = RA\{1,2\}(S_3^*) + RA\{1,2,3\}(S_4^*) = W_3 + RA\{1,2,3\}(S_4^*)$$

and hence $RA\{1,2,3\}(S_4^*) \geq W_3$.

Claim 7: If $k_3 > 0$, then $C(S_4^*) \geq C_3$.

Proof of Claim 7: From Claim 6, S_4^* is such that $RA\{1,2,3\}(S_4^*) \geq W_3$ and $C(S_4^*)$ is minimum. By the choice of the algorithm, $RA_i/c_i \geq RA\{1,2,3\}(S_4^*)/C(S_4^*)$ for every type 3 knapsack i used by the algorithm. Hence, $W_3/C_3 \geq RA\{1,2,3\}(S_4^*)/C(S_4^*)$, and since $RA\{1,2,3\}(S_4^*) \geq W_3$, we have $C(S_4^*) \geq C_3$.

Claim 8: Let $H = S_1 \cup S_2 \cup S_3$. If $k_3 > 0$, then $C(H) \leq 2C^*$.

Proof of Claim 8: We consider two cases.
Case 1.$C_3 \geq C_1 + C_2$. In this case, $C(H) \leq 2C_3 \leq 2C(S_4^*) \leq 2C^*$ by Claim 7.
Case 2.$C_3 < C_1 + C_2$. We have $C(H) = C_1 + C_2 + C_3 \leq 2[C_1 + C_2] \leq 2T \leq 2C^*$ since type 1 and type 2 knapsacks are at least half full and weight assigned in Phase 1 is at most $T/2$.

Claim 9: If $k_3 = 0$, then $C(H) \leq C^*$.

Proof of Claim 9: Total capacity used by the first phase of the heuristic is $C_1 + C_2 \leq T$ since each knapsack is at least half full.

From Claims 8 and 9 it follows that $C(H) \leq 2C^*$.