New polynomial time instances to various knapsack type problems

Isto Aho*
Dept. of Computer and Information Sciences,
FIN-33014 University of Tampere, Finland.
Email: isto.aho@uta.fi

9th April 2002

Abstract

We describe a special case of the interactive knapsack optimization problem (motivated by the load clipping problem) solvable in polynomial time. Given an instance parameterized by k, the solution can be found in the polynomial time, where the polynomial has degree k. In the interactive knapsack problem k is connected to the length induced by an item. A similar construction solves a special case of the 0–1 multi-dimensional knapsack and the 0–1 linear integer programming problems in polynomial time. In these problems the parameter determines the width of the restriction matrix, which is a band matrix. We extend the 0–1 multi-dimensional knapsack solution to 0–n multi-dimensional knapsack problems (and to 0–n IP problems). Our algorithms are based on the (resource bounded) shortest path search: we represent restrictions efficiently in a form of a graph such that each feasible solution has a path between given source and target vertices.

1 Introduction

The knapsack problem and its variants are much studied combinatorial optimization problems. Both multi-dimensional and interactive knapsack problems involve n items, which we try to pack in m knapsacks. The multi-dimensional knapsack problem (MDKP) has many applications; for example in situations where different periodic resource requirements must be satisfied over several periods (see [28]), or in

*The author is grateful to Erkki Mäkinen for his time and comments

query optimization (see [22], and the references of [12]). Interactive knapsacks are motivated by the load clipping problem [2, 5]. Lin [28] presents a survey of some well-known non-standard knapsack problems. Many integer and combinatorial optimization problems are formulated as general linear integer programming problems. There exist several monographs devoted to (linear) integer programming and they describe numerous applications. For general introduction, see for example [32].

The interactive knapsack (IK) model is a generalization of the model behind the generalized assignment problem (GAP) [2] (more on GAP below). An instance of the interactive knapsack problem contains several knapsacks somehow connected together: inserting an item in one of the knapsacks also affects the other knapsacks (in a way to be described below). Most reasonable problems using the IK model are strongly NP-complete.

In the IK model we have an ordered group of knapsacks, also called a knapsack array. When an item j is put in a knapsack i, it is copied into knapsacks $i+1, \ldots, i+c$ ($c \geq 0$). The inserted item and its copies together are called a *clone*. Moreover, an item can also "radiate", that is, some portions of the item can be copied into knapsacks $i-u, \ldots, i-1$ and $i+c+1, \ldots, i+c+u$ ($u \geq 0$). This behavior (copying to form clone and radiation) can model, for example, the controls made in the electricity management, like in the load clipping (see [1, 2, 3, 5]). We allow the same item to be inserted several times in the knapsack array but not into the same knapsack. In load clipping, this corresponds the situation where we can make several controls with the same utility. Other (still) hypothetical applications of IK-model are described in [1].

Figure 1 depicts a sample control of a controlling utility in load clipping in realistic and theoretical situations. The control period is seen as a decreased load corresponding to the clone, and after the control period there is increased consumption for awhile, which corresponds to the radiation.

One possible approach to avoid computational difficulties is to restrict the problems to use one item at a time leading to interactive knapsack heuristic optimization (IKHO) problems. Unfortunately, some IKHO problems are NP-complete, too. IKHO can be seen as a special case of 0–1 MDKP (more on MDKP below), although

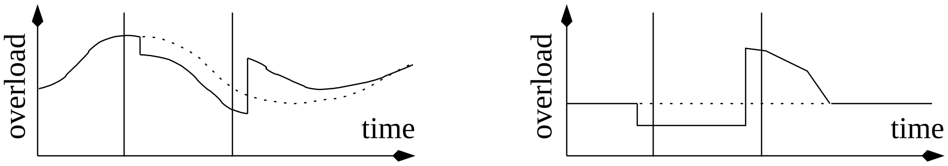

Figure 1: A control made in realistic and theoretical load clipping situations.

IKHO and 0–1 MDKP equal each other (more on this in Section 2). (See [2].)

In GAP we are to maximize $\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij}$ subject to $\sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i$ $(i = 1, \ldots, m)$, $\sum_{i=1}^{m} x_{ij} \leq 1$ $(j = 1, \ldots, n)$, and $x_{ij} = 0$ or 1, $(i = 1, \ldots, m,\ j = 1, \ldots, n)$. In GAP the profit and weight of an item depend on the knapsack, and an item is selected to a knapsack. Notice that all values p_{ij}, w_{ij} and b_i are positive integers. For a survey of GAP, see, for example, [30, 31].

In the multi-dimensional knapsack problem (MDKP) [11], selection of an item means that we add an amount to every knapsack, as opposite to GAP, where items are inserted in separate knapsacks. In 0–1 MDKP we are to maximize $\sum_{j=1}^{n} p_j x_j$ subject to $\sum_{j=1}^{n} w_{ij} x_j \leq b_i$ $(i = 1, \ldots, m)$, and $x_j = 0$ or 1 $(i = 1, \ldots, m,\ j = 1, \ldots, n)$. The parameters are positive integers.

The formulation of linear integer programming (IP) is otherwise the same as MDKP, but now we allow negative parameters as well. Thus, p_j, w_{ij}, and $b_i \in \mathbb{Z}$. If we use restriction $x_j = 0$ or 1 for each item, we call the problem 0–1 IP.

MDKP and IP (and 0–1 MDKP and 0–1 IP) have several exact and heuristic algorithms, see [36, 32]; dynamic programming [38, 33] and genetic algorithms [25], to name few. Osorio, Glover and Hammer [34] give an overview of MDKP referencing several paradigms, including branch and bound, enumeration, different relaxation techniques with branch and bound, like Lagrangean, surrogate, composite and linear programming, and different heuristics like tabu search and the genetic algorithms, to solve the problem. Other extensive survey can be found in [9] and [14]. Population heuristics include some of the above methods [6]. Hill climbing is considered in [26]. Further, Dyer *et al.* [17] consider the number of solutions to MDKP. The dynamic programming algorithms are the most relevant to our work.

A fixed parameter algorithm for the classical 0–1 knapsack problem has been considered in [18]. Caprara *et al.* [10] argue that a special case of 0–1 MDKP is not fixed-parameter tractable, and hence, neither 0–1 MDKP. Thus, we can rule out the existence of fixed-parameter tractable algorithms for IKHO and IKO problems as well. Note that both 0–1 MDKP and IKHO can be solved pseudopolynomially by dynamic programming and by enumerative methods. IKO, in turn, is strongly NP-complete [2].

If m is fixed, 0–1 MDKP has a polynomial time approximation scheme (PTAS) [20]. Chekuri and Khanna [12] show that 0–1 MDKP is hard to approximate within a factor of $\Omega\big(m^{(1/(\lfloor B \rfloor+1))-\epsilon}\big)$, for every fixed ϵ, where $B = \min_i b_i$. Srinivasan [37] shows how to obtain $\Omega\big(t^{B/(B-1)}\big)$ solutions in polynomial time, where $t = \Omega\big(z^*/m^{1/B}\big)$ and z^* is the optimal solution. GAP is APX-hard, but it can be 2-approximated. The APX-hardness is shown for a very restricted case of GAP [13]. Since IKO is a generalization of GAP [2], it is also APX-hard and does not have a polynomial time approximation scheme. IKHO is closely related to 0–1 MDKP, where m is not fixed, and hence, IKHO is APX-hard, too [4].

Our methods are based on extensive use of the resource bounded longest paths. Loosely speaking, our method codes some of the restrictions into the sets of vertices, and some into the weights abound to edges, while the profits are coded into the lengths of the edges.

Fortunately, the graphs that we construct are acyclic and directed and hence, the longest weight-constrained path problem equals the shortest weight-constrained path problem. The resource bounded shortest path problems have been studied extensively. Other names for the problem are the restricted shortest paths and the shortest weight-constrained paths. The shortest weight-constrained path problem (with one weight constraint) has a fully polynomial time approximation scheme (FPTAS) [24]. Also k-shortest paths, Lagrangean relaxation and dynamic programming has been applied, see [16] for an overview. Ziegelmann [40] gives also a recent survey of the constrained shortest paths.

The reason for using the longest path problems is that their algorithms are easier than constructing the dynamic programming solutions directly to IKHO and IKO problems. We use the term longest path instead of the shortest, because the use of the term "longest" is more natural in the context of maximizing. We apply the one resource bounded longest path problem in Section 2, and the several resource case in Section 3, while we need them both in Section 4.

Section 2 deals with IKHO problem: Subsection 2.1 introduces the model and problem, and recalls some basic results, while Subsection 2.2 gives new polynomial time instances for IKHO. The instances are connected to the cases, where the length of clone and radiation is at most logarithmic with respect to the knapsack array. In load clipping, this corresponds to the case, where the control length is at most logarithmic with respect to the optimization time interval.

After one item case, we turn to IKO in Section 3. First, we introduce the model and the optimization problem in Subsection 3.1, and then, we introduce the new polynomial time instances for IKO. Subsection 3.2 contains the case with no radiation, and Subsection 3.3 handles the general case. The results in this section are generalizations of the corresponding results obtained for IKHO.

The presented techniques are applied to 0–1 MDKP (and 0–1 IP) problems in Section 4. We obtain polynomial time algorithms for the cases, where the restriction matrix is a band matrix and the bandwidth of the matrix is at most logarithmic with respect to the size of the matrix. We also show, how to efficiently solve instances with variables ranging between 0 and $2^n - 1$, where n is fixed, thus giving algorithms for 0–n MDKP (and 0–n IP).

We neglect the question of the sizes of instance parameters (that is, the sizes of weights, profits and knapsack sizes); in the case of load clipping, for instance, we may assume that every parameter including the optimal solution fits into a word. However, the detailed analysis for the MDKP and IP instances should be made. We have checked that the results will not change tremendously, and leave the exact running times open.

In a way, these logarithmic size bounds on structures (the problem still being polynomial time solvable) are best we can hope, since now some of the parameters of the problems can grow with the other instance parameters. Note that we describe instances for 0–1 MDKP, where both the number of items and the number of knapsacks grow, which, in general, is an APX-hard case. From the practical point of view, the situation is similar in 0–1 IP: if we have an instance with $m \times m$ bandwidth restriction matrix of fixed width w (that is, $w = O(1)$), we can solve the instances in time bounded linearly by $m \log(m)$ and $w2^{2w}$ by using the methods to be presented. However, by applying the dynamic programming solution presented in [36], pp. 264–265, or in [35] to these instances, we end up with much larger graphs and slower running times.

Dynamic programming solution to the 0–1 knapsack problem has been interpreted as a shortest path problem [36], pp. 261–263, or as a maximum-weight path problem [32], pp. 420–422. The shortest paths have been applied to IP problems as well [36], pp. 264–265. See also [39] and the graphs therein. The novelty in our results is that we give an efficient transformation of the described instances to a simpler graph problems.

2 The IKHO problem

2.1 Problem and earlier results

Let IKHO consist of

m,	the number of knapsacks,
$x_i \in \{0, 1\}$,	the decision variables ($i = 1, \ldots, m$),
$p_i \in \mathbb{N}$,	the profits of items ($i = 1, \ldots, m$),
$w_i \in \mathbb{N}$,	the weights of items ($i = 1, \ldots, m$),
c,	the number of copies ($c + 1$ is the length of a clone),
u,	the length of radiations,
$I_i : \mathbb{N} \to \mathbb{Q}$,	interaction functions that map the distance from i to a rational number ($i = 1, \ldots, m$),
$b_i \in \mathbb{N}$,	capacities of the knapsacks ($i = 1, \ldots, m$),
K,	a positive integer that gives the upper bound for the times we can insert an item into the knapsack array.

By using c and u we (usually) mean that I_i equals one, for knapsacks $i, \ldots, i + c$, and is arbitrary for knapsacks $i-u,\dots,i-1$ and $i+c+1,\dots,i+c+u$. For all other knapsacks, I_i is zero. This interpretation is used most of the time, and the exceptions are pointed out. The main reason for using c and u is restriction (3) below, also shown in Figure 2 as overlapping radiations and non-overlapping clones. Recall that in load clipping, the controls cannot overlap, while overlapping is allowed for some of the effects of controls.

An IKHO problem is to

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{m} x_i \sum_{k=i-u}^{i+c+u} I_i(k) p_k && (1)\\
\text{subject to} \quad & \sum_{i=1}^{m} w_i I_i(l) x_i \le b_l, && (2)\\
& x_k = 0,\ \text{for } i < k \le i+c, \text{ when } x_i = 1, && (3)\\
& x_i = 0 \text{ or } 1, \text{ and} && (4)\\
& \sum_{i=1}^{m} x_i \le K, && (5)
\end{aligned}
$$

where $l = 1,\dots,m$ in (2) and $i = 1,\dots,m$ in (3) and (4). Restriction (2) checks that the knapsacks are not over filled, (3) checks that the clones are separate, (4) checks that exactly one item can be put in a knapsack, and (5) gives the maximum number of times an item can be put in the knapsack array.

The above formulation is closely related to 0–1 MDKP. In this form, IKHO can be transformed into 0–1 MDKP, and 0–1 MDKP can be transformed into IKHO [4] (which is shortly demonstrated in Section 4). Hence, why to study IKHO problem at all? The answer is that IKHO (1)–(5), among many other problems, can be seen as a specialized 0–1 MDKP. Normally, an IKHO instance uses restrictions (3) and (5) that take a special form in 0–1 MDKP. If we transform an 0–1 MDKP instance to IKHO, these restrictions are not normally used: we will have $c = 0$ and $K = m$.

Another motivation for studying IKHO is that (1)–(5) is the simplest formulation of the problems in a larger family of IKHO-type problems of practical interest. Other family members use c and u differently: they can depend on the knapsack or c can be a variable for each inserted clone (and the value of u depends on the value chosen for c). There are also other variants, which can be motivated by the load clipping application.

Figure 2 reflects the above problem setting (1)–(5). We have inserted an item into knapsack i. The same item is also inserted into knapsack $i+10$. The length of clone is $c+1=5$ and the length of radiation is 5. In the methods presented in the next subsection, we distinguish between left and right radiation: they are denoted with u_l and u_r. In the case of Figure 2, we have $u_l = 3$ and $u_r = 5$. Define ℓ to

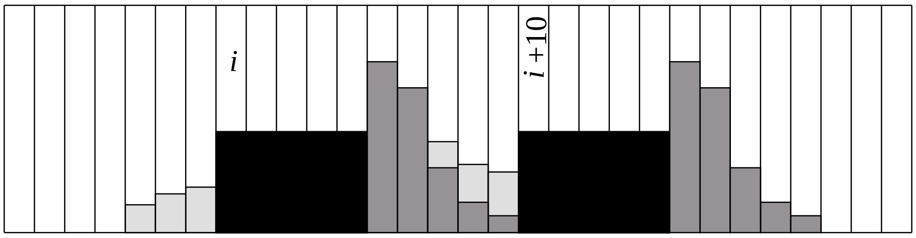

Figure 2: A sample clone inserted in the knapsacks i and $i+10$.

be the total length of the clone, and left and right radiations together. Hence, the total length in the example is $\ell = 3+5+5 = 13$.

If the knapsacks were larger, we could insert the item third time into knapsack $i+5$: restriction (3) says that the parts drawn in black cannot overlap while the radiation (parts drawn in grey in Figure 2) can overlap.

An IKHO instance with $K = O(1)$ or $u = 0$ is solvable in polynomial time [2]. If K is fixed ($K = O(1)$), we can enumerate every bit string of length m having at most K ones in polynomial time ($O(x^K)$). If $u = 0$, the problem reduces to the search for the longest weight-constrained path in polynomial time as will be shown in Theorem 2.

Before turning to the case $u = 0$, we shortly discuss about the longest path problem. Let $G = (V, E)$ be a graph, where each edge $e \in E$ has length $l_e \in \mathbb{N}$ and weight $w_e \in \mathbb{N}$. The longest weight-constrained path problem is to find the longest path P between given vertices such that the weight constraint is not violated: $\sum_{p \in P} w_p \leq K$. Note that the longest and shortest weight-constrained path problems are not equivalent in general; for example, while they both are NP-complete, they do not have similar special cases solvable in polynomial time (see [2], appendix).

The following result is stated for the shortest weight-constrained paths.

Theorem 1 *[21], p. 214. Consider a graph $G = (V, E)$ with each edge having a length l_e and a weight w_e, where $l_e, w_e \in \mathbb{N}$. If the weights are equal and the graph is acyclic and directed, the longest weight-constrained path can be found in polynomial time.*

We can relax the assumption of equal weights when we design algorithms for IKHO; this case is handled in Theorem 4. In the following theorem we suppose that c is variable and restricted to be between a given minimum and maximum lengths. It means that the problem setting (1)–(5) has to be fixed accordingly, but the changes are minor. In load clipping, it is reasonable to assume that some controlling utilities can have variable controlling lengths [5, 3].

We represent the following theorem and its corollary, because we were able to make the proof simpler by constructing a simpler graph than in [2]. Moreover, the presented technique helps one to understand the new methods to be presented.

Theorem 2 *[2]* IKHO $\in$ P*, for* $u = 0$ *and variable* c *(*$cmin \leq c \leq cmax$*).*

Proof. Let $G = (V, E)$ be a directed graph with edges E and vertices V. We assign weights and lengths ($\in \mathbb{Z}^+$) to each edge. Vertices are divided into two groups A and B of m and $m+1$ vertices, respectively. Vertices in A are labeled as $A_1, A_2, \ldots, A_m$ and in B as $B_1, \ldots, B_m, B_{m+1}$. Knapsack i in the knapsack array corresponds to a vertex pair consisting of A_i and B_i.

Group B has forwarding edges of length 0: these are (B_i, B_{i+1}), where $1 \leq i \leq m$. There are also links from group B to A; those are edges (B_i, A_i) of length 0, where $1 \leq i \leq m$. Link (B_i, A_i) corresponds to the decision to include the item in knapsack i $(x_i = 1)$.

Define $p'_{i,j} = \sum_{k=i}^{i+j} p_k$, where $\mathrm{cmin} + 1 \leq j \leq \mathrm{cmax} + 1$. The sum stands for an item of length j put in knapsack i (and the associated profit of the item and its clone).

An edge (A_i, B_{i+j}), where $(i + \mathrm{cmin} + 1) \leq j \leq (i + \mathrm{cmax} + 1)$, has length $p'_{i,j}$ and corresponds to the item put in knapsack i, where the clone length $(c + 1)$ is j. If $p'_{i,j}$ is negative, then we do not insert the edge. Our choice for j ensures that restriction (3) is not violated. Path $B_i, A_i, B_{i+\mathrm{cmin}+1}$ is always longer than path $B_i, B_{i+1}, \ldots, B_{i+\mathrm{cmin}+1}$, because of the positive profits given to the edges from group A to group B.

Each edge has weight 1. Each insertion of an item into the knapsack array increases the path weight by 2 as we traverse from group B to A and back again. Now we should find the longest weight-constrained path between vertices B_1 and B_{m+1}, where the weight of the path is at most $2K$ (to comply with (5)). As all weights are equal, this can be solved in polynomial time by Theorem 1 (see [21], p. 214, and also [2], appendix). $\square$

Corollary 3 *[2]* IKHO $\in$ P*, for* $u = 0$ *and fixed values of* $c > 0$*.*

Proof. We have $\mathrm{cmin} = \mathrm{cmax} = c$ in the proof of Theorem 2. Only one edge (A_i, B_{i+c+1}) departs from group A to B. $\square$

Figure 3 illustrates the graph construction for IKHO with $u = 0$. We can go from B_1 to B_{m+1} by only using the vertices of group B meaning that no item is inserted in any knapsack (this occurs, for instance, when each $p'_{i,j}$ is negative). We may insert an item into as many knapsacks as it fits (at most $\min(K, m/(\mathrm{cmin}+1))$ times). Figure 3 does not show weights as they are all 1. Ones, zeros, and $p'_{i,\mathrm{cmax}+1}$ indicate the lengths.

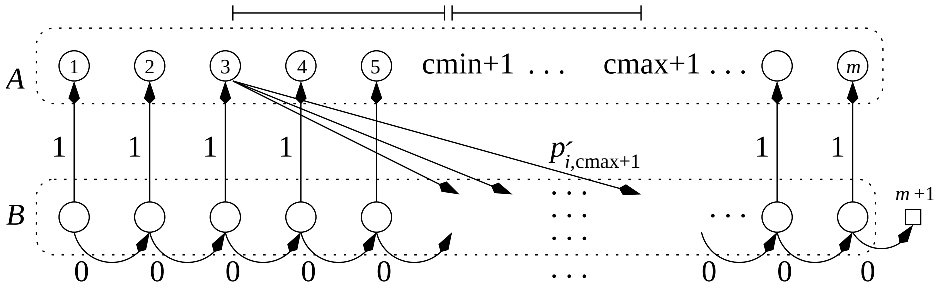

Figure 3: The graph constructed in Theorem 2. If we insert an item into the third knapsack, we can next go to the vertices which lay at least $\text{cmin} + 1$ vertex-pairs to the right.

2.2 A special case of IKHO

When $K = O(m)$ and $u = m$, IKHO becomes NP-complete [2]. This can be seen by a transformation from the 0-1 knapsack problem. Moreover, this version of IKHO is APX-hard having the same approximability properties as 0–1 MDKP [4].

We now solve the open problem [4] of the NP-completeness of IKHO in the case, where the total length of clone and radiation parts has logarithmic size, that is, $\ell = O(\log(m))$. We also parameterize the total length ℓ.

First we note that there is no need for the equal weight assumption made in Theorem 1. (And as already said, since we suppose that the graph is acyclic and directed, we can use the shortest weight-constrained paths methods.)

Theorem 4 *Consider an instance of the longest weight-constrained path problem with a graph $G = (V, E)$, where each edge e has length l_e and weight w_e, where $l_e, w_e \in \mathbb{N}$. Suppose further that the weight-constraint K is polynomial on the number of vertices $|V|$. If G is acyclic and directed, the longest weight-constrained path between two vertices can be found in polynomial time.*

By redesigning the dynamic programming solution of the knapsack problem, we can solve the longest weight-constrained path in $O(K|V|^2)$ time. We advise the reader to see [36], pp. 261–263 and [32], pp. 420-422. There the graphs consist of one parameter edges (length) and the second parameter (weight) is coded into the labels of the vertices. Other possibility is to directly form graphs, whose edges have both length and weight.

However, the shortest weight-constrained path can be solved in $O(K|E|)$ time, if the weights are positive (see [16] and the references therein). This holds also for acyclic directed graphs, and therefore the time bound is valid for the longest weight-constrained paths in acyclic directed graphs. We will use the $O(K|E|)$ bound later. Note that since $|E| = |V|(|V| - 1)$ is possible in directed acyclic graphs, in general, the space consumption and running time are $O(K|V|^2)$ for the longest weight-constrained path problem in acyclic directed graphs.

Now we are ready to handle the special case of IKHO, where the total length of the clone and radiation is $\ell = O(\log(m))$. In the proof we construct a graph with a source and target vertices s and t, and sets of vertices divided into groups. In each group the vertices are identified by bit strings of length ℓ. The bit string can be interpreted as a selection of items to different knapsacks: for example, v_{10100} corresponds to a situation where $\ell = 5$, $x_i = x_{i+3} = 1$ (an item is selected for knapsacks i and $i+3$) and $x_{i+2} = x_{i+4} = x_{i+5} = 0$. The value of i depends on the vertex group; two different vertices may share a subscript if they are in different groups. In what follows, a selection is a bit string of length ℓ and it is interpreted as above.

Lemma 5 *Let the maximum total length of the clone and radiation parts be ℓ. An item selection $v_{1,\ldots,\ell}$ (ignoring any other items in other knapsacks) can be checked against restrictions* (2) *and* (3) *in polynomial time.*

Proof. Suppose we put an item in a knapsack. The corresponding interactions can be calculated in $O(\ell)$ time. A selection $v_{1,\ldots,\ell}$ can contain at most ℓ ones (that is, we can put an item at most ℓ times into the knapsack array). Hence, checking (2) takes $O(\ell^2)$ time. Restriction (3) ensures that ones in the selection $v_{1,\ldots,\ell}$ are not too close to each other. One pass of the selection is enough (in $O(\ell)$ time) to check this. □

Consider next a group of all selections of length ℓ containing 2^ℓ bit strings. We may conclude directly by using the proof of Lemma 5 that we need $O(\ell^2 2^\ell)$ time to comply with restrictions (2) and (3). Yet, by tabulating calculations, we obtain a better bound.

Lemma 6 *Let the maximum total length of the clone and radiation parts be ℓ. Every item selection $v_{1,\ldots,\ell}$ (ignoring any other items in other knapsacks) can be checked against restrictions* (2) *and* (3) *in $O(\ell 2^\ell)$time.*

Proof. We build up a result table having 2^ℓ entries, each of length ℓ. Each entry corresponds to ℓ consecutive knapsacks with fillments (fillings) corresponding to the selections label (bit string).

First, we calculate the effects of an inclusion of an item for every position in the selection. That is, first we calculate ℓ entries in the result table to the following selections: $v_{0\cdots01}$, $v_{0\cdots010}$, $v_{0\cdots0100}$, and at last, $v_{10\cdots0}$, all in $O(\ell^2)$ time.

Next, we form every bit string of length ℓ with the pre-calculated strings, and construct the rest of the result table. We can combine two bit strings and the corresponding results to a new bit string (and to a new result entry) in $O(2\ell)$ time.

There are $\binom{\ell}{2}$ bit strings having two ones. The bit string and especially the entries in the result table can be constructed in

$$\binom{\ell}{2} O(2\ell)$$

time. We can use the bit strings having 1 ones and $k-1$ ones to combine them to a string having k ones. Thus, the result entries corresponding to k ones need

$$\binom{\ell}{k} O(2\ell) \tag{6}$$

time. Summing over (6) we have

$$\sum_{k=0}^{\ell} \binom{\ell}{k} O(2\ell) = O(2\ell) \sum_{k=0}^{\ell} \binom{\ell}{k} = O(2\ell)O(2^{\ell}) = O(\ell 2^{\ell}).$$

Hence, the total time consumption is $O(\ell^2) + O(\ell 2^{\ell}) = O(\ell 2^{\ell})$. □

The calculations of the profits take the same time as checking (2) and (3). Thus, to check restrictions and to calculate profits for every selection of length $O(\ell)$, we need $O(\ell 2^{\ell})$ time.

Our next step is to construct a graph, which contains every selection so that each legal (according to (2) and (3)) item combination is achievable. Lengths of the edges correspond to profits given by the inclusions of the items and hence, we want to find out long paths. We construct the graph with the following three lemmas, which form $O(m)$ selection groups each containing every selection of length ℓ. Thus, there are totally 2^{ℓ} selections in the graph. A selection group corresponds to a knapsack.

In each of the following lemmas, let I_i, p_i, w_i, b_j, m and K form an instance of IKHO. Further, we refer with vertex set or with vertex group to the selection group mentioned in the previous paragraph. When we say that the selections can be constructed in polynomial time, we mean that the graph with vertices and edges can constructed, and the optimal path along with the optimal solution can be found in polynomial time (on m).

We name the vertices in each vertex set $v_{x_i \cdots x_{i+\ell}}$, where $x_i \cdots x_{i+\ell}$ may have 2^{ℓ} different bit combinations.

Lemma 7 *Suppose that the total length of the clone part and radiation is ℓ, where $\ell = \log(m)$. Selections (and optimal solution) for the first ℓ knapsacks of an instance of* IKHO *can be constructed in polynomial time.*

Proof. We start from vertex s and insert $2^{\ell} = m$ outgoing edges to the first vertex set, which in turn, corresponds to the first ℓ knapsacks.

The vertices in the first vertex set correspond to every item selection combination for the first ℓ knapsacks. If some combination violates (2) or (3), the corresponding edge is not inserted. The checks against (2) and (3) can be done in polynomial time (on m) for each candidate edge by Lemma 6. The total time for all m edges is $O(\ell 2^{\ell}) = O(m \log(m))$.

To find out the length of an edge, note that $\ell = u_l + c + 1 + u_r$, where u_l and u_r are the lengths of the left and right radiation, and $c+1$ is the length of the clone part.

We take into account only profits of the items that are included in the first u_l knapsacks. Thus, the profit, or the length of an edge, is $\sum_{i=1}^{u_l} x_i \sum_{k=i-u_l}^{i+c+1+u_r} I_i(k) p_k$. Moreover, the weight of an edge equals to the number of ones in the first u_l positions in its label.

If, on the other hand, we do not include the item in the first u_l knapsacks ($x_1 = \cdots = x_{u_l} = 0$), the profit and the weight are zero.

We calculate the profits at the same time when we form vertices and edges, and make checks against the restrictions. Hence, the time needed is $O(m \log(m))$.

By using the longest weight-constrained path algorithm, we can check (5) and find out the longest path (optimal solution) in polynomial time (on m). Our graph construction complies with (4) automatically. $\square$

As an example, consider an instance, where $c = 0$, the left radiation has length 2, and the right radiation has length 4. Now the length of a selection is $\ell = 2+1+4 = 7$ and the vertex set contains $2^7 = 128$ vertices.

The lengths (or profits) of edges departing from s use the left radiation parts and the first vertex set corresponds to the inclusions of items into the first u_l knapsacks. Edge $(s, 1110000)$ would have length that corresponds to the inclusion of an item in knapsacks 1 and 2, that is, $x_1 = x_2 = 1$. Restrictions, on the other hand, are checked against the decision made for all seven knapsacks.

The algorithm described in the following lemma is an example of inductive algorithm design principle [29]. Our induction hypothesis is that every legal selection can be reached so that the longest weight-constrained path gives the maximum profits while maintaining the feasibility of the solution.

Lemma 8 *Suppose that the total length of the clone part and radiation is ℓ, where $\ell = \log(m)$. Selections for the first $m - (c + u_r)$ knapsacks of an instance of* IKHO *can be constructed and the optimal path can found in polynomial time.*

Proof. We prove the lemma by induction. By Lemma 7, we can construct the first vertex group for the first ℓ knapsacks and find out the longest weight-constrained path (the optimal solution) in polynomial time, calculating the profits for the first u_l knapsacks and taking into account restrictions (2)–(5) for the first ℓ knapsacks.

Now we make an induction hypothesis that the optimal solution can be found for the first q knapsacks, where $q = u_l, \ldots, m - (c + u_r)$. In other words, the induction hypothesis is that every legal selection can be reached, the profits are calculated for the first q knapsacks correctly, and the restrictions are checked for the first $q+c+1+u_r$ knapsacks. Let $g = q - u_l + 1$. (Now g refers to a vertex group.) In the terms of vertex sets, Lemma 7 proves the claim for the first vertex set, $g = 1$, and after it, we have made an induction hypothesis for the first $g = 1, \ldots, m-(c+u_r+1)$ vertex sets.

We construct a vertex group $g+1$ from the vertices of vertex group g by inserting edges $(v_{x_g \cdots x_{g+\ell}}, v_{x_{g+1} \cdots x_{g+\ell+1}})$, if $v_{x_{g+1} \cdots x_{g+\ell+1}}$ is a legal combination not violating (2) and (3). There will be at most two edges for each vertex in group g. By Lemma 6, we can insert edges in $O(m \log(m))$ time.

The edges from group g to $g+1$ correspond to the choice made for knapsack $g+u_l$. If $x_{g+u_l} = 1$ (we include an item in knapsack $g+u_l$), the profit is $\sum_{k=g+u_l-u_l}^{g+u_l+c+u_r} I_{g+u_l}(k) p_k$. Moreover, the weight of the edge is one. On the other hand, if $x_{g+u_l} = 0$, the length and the weight of the edge are zero. Note that the profit for the edge is defined by the state of the c^{th} bit in the vertex label (the next bit after the left radiation part). The profits can be calculated at the same time when we check the restrictions.

Suppose that a vertex $v_{x_{g+1} \cdots x_{g+\ell+1}}$ in vertex group $g+1$ is legal. The vertex has at most two incoming edges, which can depart from vertices $v_{x_g=0, x_{g+1} \cdots x_{g+\ell}}$ and $v_{x_g=1, x_{g+1} \cdots x_{g+\ell}}$ from vertex group g. Since $v_{x_{g+1} \cdots x_{g+\ell+1}}$ is legal in group $g+1$, vertex $v_{x_g=0, x_{g+1} \cdots x_{q+\ell}}$ in group g has $\ell - 1$ common knapsacks $x_{g+1} \cdots x_{g+\ell}$ with the vertex in group $g+1$, and bit x_g can be zero, we conclude that vertex $v_{x_g=0, x_{g+1} \cdots x_{g+\ell}}$ is legal, and therefore, the graph contains an edge $(v_{x_g=0, x_{g+1} \cdots x_{q+\ell}}, v_{x_{g+1} \cdots x_{g+\ell+1}})$ from group g to $g+1$. By the induction hypothesis, we can reach every legal selection in the first g vertex groups, and thus also the vertex $v_{x_g=0, x_{g+1} \cdots x_{g+\ell}}$, and further, an arbitrary legal vertex $v_{x_{g+1} \cdots x_{g+\ell+1}}$ in vertex group $g+1$.

Now we have a system that contains every selection of length ℓ in the knapsack array. This contains also the last vertex group $m-(c+u_r)$. We have linked selections so that every legal selection is attainable. The number of these selections in every set is $O((m-(c+u_r)) m \log(m)) = O(m^2 \log(m))$.

By Lemma 7, we can find out the solutions for the first vertex set in polynomial time. Then, we inductively form the graph that has a polynomial number of vertices. By Lemma 6, we can form each vertex group and the incoming edges (length and weights) in polynomial time. Our graph is acyclic and directed. By Theorem 4, the optimal solution can be found in polynomial time on the number of vertices in the graph (recall that $K \leq m/(c+1) \leq m$). Hence the claim. □

The first vertex set (and also the last vertex sets, as we will see) is a special case, because it may have more than one item giving profits and weights, as opposite to the profit calculations made in the previous lemma.

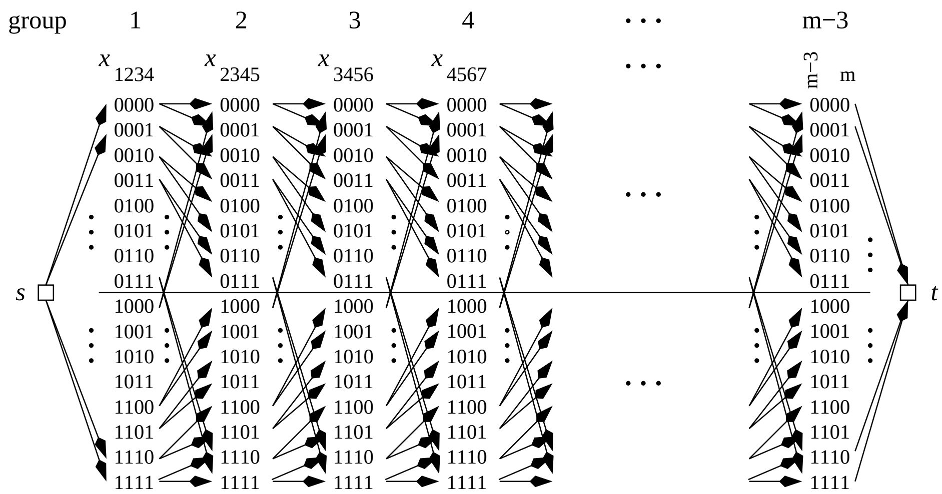

Figure 4: The graph structure for Theorem 10, when $\ell = 4$.

Suppose that we have formed vertex set g. To obtain another set of vertices, we drop the first knapsack g and add a new knapsack $g+u_l+c+1+u_r+1$. Consecutive vertex sets contain $\ell - 1$ common knapsacks. Let the common subselection be $x_{g+1}, \ldots, x_{g+u_l+c+1+u_r}$. The first item x_g can have two values, zero or one, and the new item $x_{g+u_l+c+1+u_r+1}$ can also have two values. Hence, we always have two source vertices as well as two target vertices for each vertex in every vertex set (except for the first and last vertex sets).

Let us continue the example started before Lemma 8. The profits for the vertices between vertex sets are obtained by using the left most position of the clone part in a selection, that is, the knapsack into which an item is inserted. In our example, an edge departing from vertex 1110000 in the first vertex set would have length that corresponds to the profit of choosing $x_3 = 1$. Hence, it corresponds to the inclusion of the item into the third knapsack, when we calculate the profits. Other values of the selection are not noticed and one may think that an edge from 1110000 is handled like filter "??1????", when we calculate the profits.

Figure 4 gives another example. It demonstrates a case, where $\ell = 4$ ($c = 0$, $u_l = 0$, and $u_r = 3$). Each vertex set has 16 vertices and we have at most 32 edges between the vertex sets.

In the example presented in Figure 4 the common subscript part between the first two vertex sets is 234. The first knapsack is moved off and it is replaced by the fifth knapsack, when constructing the second vertex set from the first one. The selection of an item to the knapsack i ($x_i = 1$ in the leftmost position of the label of the vertex — the left radiation is not used in this example) is shown by an edge pointing below the horizontal line. The weight of these edges is one. Note that only the coming knapsack can make a choice whether to include an item: the selection is already fixed for the common part. Thus, the profit of including an item is calculated later than the decision to include the item.

Lemma 9 *Suppose that the total length of the clone part and radiation of an instance of* IKHO *is ℓ, where $\ell = \log(m)$. Lengths (profits) can be calculated in polynomial time for the edges departing from the last vertex group $m - (c + u_r)$ to the target vertex t.*

Proof. Vertex t collects paths. The restrictions are already checked (also) for every selection in the last vertex group $m - (c + u_r)$. Profits (lengths) use the bits occupying the positions to the right (and including) the c^{th} bit.

By Lemma 6, we calculate the lengths (profits) in polynomial time for each vertex departing from the last vertex group. The weight of an edge equals the number of ones in the bits after the left radiation. $\square$

We continue with the example started before Lemma 8. The length of edge $(0001101, t)$ corresponds to the profit of selection $x_{m-3} = x_{m-2} = x_m = 1$ and $x_{m-4} = x_{m-1} = 0$. If we had $x_{m-4} = 1$, we would calculate it into the profits.

In the example presented in Figure 4 we use every bit in the selection, when we calculate the profits of edges from the last vertex group to t, because the length of left radiation is zero.

Theorem 10 IKHO $\in$ P, *when $\ell = O(\log(m))$, where m is the number of knapsacks.*

Proof. Let I_i, p_i, w_i, b_j, K, and m form an instance of IKHO. Suppose that the total length of the clone part and radiation is at most ℓ, where $\ell = \log(m)$. We form a graph G with source vertex s and target vertex t in polynomial time as described in Lemmas 7–9.

By Lemma 4, we find the best set of selections by constructing the longest weight-constrained path between vertices s and t in polynomial time. By using weight-constraints, we also check the condition (5). Because $K \leq m/(c+1) \leq m$, we find the longest weight-constrained path in polynomial time by Theorem 4. $\square$

To analyze the running times, note that we have $m-(c+u_r) = O(m)$ vertex sets. Each set has 2^ℓ vertices. Hence, the total number of vertices is $2+(m-(c+u_r))2^\ell = O(m2^\ell)$. Further, we have 2^ℓ incoming edges to the first vertex set and as many incoming edges to the vertex t. Other $m-(c+u_r)-1$ vertex sets have $2 \cdot 2^\ell$ edges. We can construct edges (and vertices) for one vertex set in time $O(\ell 2^\ell)$. Hence, we need $O(m\ell 2^\ell)$ time to construct the graph. To find the longest weight-constrained path, we need time $O(K|E|) = O(Km2^\ell)$ (in the graph every weight is positive, see [16] and the references therein for the method). Thus, the total time to construct the graph and to search for the path is $O(m\ell 2^\ell) + O(Km2^\ell) = O\big((K+\ell)m2^\ell\big)$.

If $m = 2^\ell$, we can construct the graph and find out the path in $O\big((K + \log(m))m^2\big)$ time. We can also describe the growth of the running time as the total length of the radiation and the clone increases. Let $\ell = \log(m^k)$, for some $k \in \mathbb{N}$. Now, we need to consider selections of length $2^\ell = m^k$. The overall running time is $O\big((K + k\log(m))m^{k+1}\big)$. Similarly, $\ell = k + \log(m)$ implies selections of length $m2^k$ and $O\big((K + k + \log(m))m^2 2^k\big)$ running time.

Corollary 11 *Suppose an instance of* IKHO *with m knapsacks, where the total length of clone and radiation parts is* $\log(m^k)$ *($k \in \mathbb{N}$). This instance can be solved in time $O\big((K + k\log(m))m^{k+1}\big)$. If total length is $k + \log(m)$, the running time is $O\big((K + k + \log(m))m^2 2^k\big)$.*

If $\ell = m$, then $k = m/\log(m)$ with the first result of Corollary 11 or $k = m - \log(m)$ with the second result. We set k into the second result and obtain the following asymptotic upper bound for the running time: $\big(K + m - \log(m) + \log(m)\big)m^2 2^{m-\log(m)} = (K+m)m^2 2^{m-\log(m)}$, which, in turn, is $(K+m)m^2 2^m/2^{\log(m)} = (K+m)m2^m$.

To check the running time, we put $k = m/\log(m)$ into the first result and obtain upper bound $\big(K + (m/\log(m))\log(m)\big)m^{1+m/\log(m)} = (K+m)m^{1+m/\log(m)}$. Since $m = 2^{\log(m)}$, this turns out to be $(K+m)(2^{\log(m)})^{m/\log(m)}m = (K+m)m2^m$ and hence, both results give the same running times for an instance with $\ell = m$ (as expected).

Since $K \leq m$, we obtain $O(2^m m^2)$ running time for an arbitrary instance. (Note that we always have $K \leq m/(c+1)$.) Complete enumeration has the same bound. However, if $K = m/(c+1)$, we do not need restriction (5), because we can insert the item as many times into the knapsack array as we want. In these cases we can use the shortest path algorithms designed for directed acyclic graphs. We can compute the shortest paths from single source in directed acyclic graphs in time $O(|V| + |E|)$ [15].

Corollary 12 *If $K = m/(c+1)$ and $\ell = \log(m^k)$, an instance of* IKHO *can be solved in time $O\big(k\log(m)m^{k+1}\big)$. If $K = m/(c+1)$ and $\ell = k + \log(m)$, then $O\big((k + \log(m))2^k m^2\big)$ time is enough.*

Proof. We need $O(m\ell 2^\ell)$ time to construct the graph, and the number of vertices and edges is $O(m2^\ell)$. Thus, in the first result the number of vertices is $O(m^{k+1})$ and the number of edges has the same bound. Constructing the graph takes $O\big(mk\log(m)m^k\big) = O\big(k\log(m)m^{k+1}\big)$ time. The both tasks together take $O\big(k\log(m)m^{k+1}\big) + O\big(m^{k+1} + m^{k+1}\big) = O\big((2 + k\log(m))m^{k+1}\big) = O\big(k\log(m)m^{k+1}\big)$ time.

In the second result we suppose that $\ell = k + \log(m)$. Thus, the number of vertices is $O(m2^k m)$ and the number of edges has again the same bound. Constructing the graph takes $O\big(m(k + \log(m))2^{k+\log(m)}\big) = O\big((k + \log(m)2^k m^2\big)$ time. The construction of the graph and the longest path together take $O\big((k + \log(m)2^k m^2\big) + O\big(2^k m^2 + 2^k m^2\big) = O\big((2 + k + \log(m))2^k m^2\big) = O\big((k + \log(m))2^k m^2\big)$ time. □

Similarly, we can obtain running times for situations, where ℓ is some fixed number ($\ell = O(1)$). The upper bound is now given by $O\big((K + \ell)m2^\ell\big)$. Note that asymptotically, $O\big((K+\ell)m2^\ell\big) = O(Km)$, which is less than the running time of the enumerative solution; the number of different solutions is at least $\sum_{k=0}^{K} \binom{m-kc+c}{k} = \Omega(m^K)$ (see [1], pp. 19 and 24, and [2]).

From the application point of view, the above results describe the running times well enough. For example, in load clipping heuristics making a control plan for one group at time are often approximation algorithms [5, 3]. Typical lengths can be from 6 to 12 for the clone part and from 9 to 12 units for the radiation. Hence, the total lengths range often from 15 to 24 units. Note that there exists controlling utilities (an items) that have much longer total lengths. Moreover, some others use small K (you do not want to freeze because your electricity heaters are controlled too often), while some of the utilities do not need K. For example, street lights or electricity generators have long total lengths, and K is not needed with the generators. In load clipping, the number of time units (the total lengths of items) depends on the granularity of the time interval. A typical value is 5 or 10 minutes, and the optimization interval (m) can be, for instance 24 hours divided into 5 minute (or 10 minute) slots.

Another motivation for the presented methods is that they can be used as a comparison method. If we are designing some other exact or heuristic algorithms, it is good to know, how long it takes to form exact solutions with these methods. Moreover, the presented method can also be used as a starting point for some heuristic approaches. For example, if the presented method is too slow, we can ignore some of the vertices and edges, or states in the path calculations.

3 The IKO problem

3.1 Problem

IKO is otherwise the same as IKHO, but now we have n items. IKO is strongly NP-hard [2], while IKHO is pseudo-polynomial time solvable.

In IKO, we have $x_{ij} \in \{0, 1\}$, profits $p_{ij} \in \mathbb{N}$, weights $w_{ij} \in \mathbb{N}$, lengths of clones c, lengths of radiations u, and interaction functions $I_{ij} : \mathbb{Z} \to \mathbb{Q}$, for knapsacks $i = 1, \ldots, m$ and items $j = 1, \ldots, n$. Moreover, we have a capacity $b_i \in \mathbb{N}$, for each knapsack i, and a positive integer K that gives the upper bound for the times we can insert an item into the knapsack array. The interaction I_{ij} of knapsack i and item j maps the distance from i to a rational number. By using c (clone) and u (radiation) we mean that I_{ij} equals one, for knapsacks $i, \ldots, i+c$, and is arbitrary for knapsacks $i-u, \ldots, i-1$ and $i+c+1, \ldots, i+c+u$. For all other knapsacks, I_{ij} is zero.

An IKO problem is to

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \sum_{k=i-u}^{i+c+u} I_{ij}(k) p_{kj} && (7) \\ \text{subject to} \quad & \sum_{j:\exists i: l \in I_{ij}} x_{ij} w_{ij} I_{ij}(l) \leq b_l, && (8) \\ & x_{kj} = 0, \text{ for } i < k \leq i+c, \text{ when } x_{ij} = 1, && (9) \\ & x_{ij} = 0 \text{ or } 1, \text{ and} && (10) \\ & \sum_{i=1}^{m} x_{ij} \leq K, && (11) \end{aligned}$$

where $l = 1, \ldots, m$ in (8), $i = 1, \ldots, m$ in (9) and (10), and $j = 1, \ldots, n$ in (8)–(11). Restriction (8) checks that the knapsacks are not over filled, (9) checks that the clones are separate, (10) checks that only one item j can be put in a knapsack, and (11) gives the maximum number of times an item can be put in the knapsack array.

We prove new results for IKO by using the longest r-weight-constrained path problem. Let $G = (V, E)$ be a graph, where each edge $e \in E$ has length $l_e \in \mathbb{N}$ and r separate weights $w_{e1}, \ldots, w_{er}$. The longest r-weight constrained path problem is to find the longest path P between given vertices such that the weight constraints are not violated: $\sum_{e \in P} w_{ej} \leq K_j$, where $j = 1, \ldots, r$.

Because our graphs are directed and acyclic, we can use the shortest r-weight constrained path problem, which can be solved in $O(|V||E|d)$ time with dynamic programming [7], where d is the number of distinct resource vectors. The problem handled by Beasley and Christofides in [7] is slightly more general than in our case (they have resource bounds in vertices as well). Further, we have $d = K^n$, or $d = K_1 \cdots K_n$, if we use separate bounds for inclusion times for the items.

Based on folklore, we write down another similar result about the running time, which suits our purposes better, since $|V||E|$ can be larger than $|V|^2$ in the graphs we construct. Further, 0–1 MDKP can be solved by the presented dynamic programming, which is discussed in Section 4.

Lemma 13 *Let $G = (V, E)$ be a graph with weight constraints $K_1, \ldots K_r \in \mathbb{N}$, and with each edge e having length $l_e \in \mathbb{N}$ and weights $w_{e1}, \ldots, w_{er} \in \mathbb{N}$. If G is directed* *and acyclic, the longest r-weight constrained path problem between given vertices can be solved in $O(K_1 \cdots K_r |V|^2)$ time and space.*

Proof. We apply dynamic programming and the proof follows closely the proof given by Bertsekas [8], p. 53.

Let $V = \{1, \ldots, |V|-1, t\}$ be the set of vertices, and let t be the target vertex. If there is no edge e between vertices i and j, then $l_e = w_{ek} = \infty$, where $k = 1, \ldots, r$.

Let $f_p(i, k_1, \ldots, k_r)$ be the minimum length from vertex i to vertex t in $|V| - p$ moves with total weights $k_1, \ldots, k_r$. The optimal path having weights $k_1, \ldots, k_r$ from vertex i to t is $f_1(i, k_1, \ldots, k_r)$. Each edge can have $K_1 K_2 \cdots K_r$ different weight combinations.

Initialize $f_{|V|-1}(i, w_{ek_1}, \ldots, w_{ek_r}) = l_e$, where $i = 1, \ldots, |V| - 1$ and $e = (i, t)$. If there is no edge having weights $k_1, \ldots, k_r$ from i to t, then $f_{|V|-1}(i, k_1, \ldots, k_r) = \infty$. The shortest distance from i to t in $|V| - p$ moves (stages) is

$$f_p(i, k_1, \ldots, k_r) = \min_{j=1,\ldots,|V|-1} \left\{ l_e + f_{p+1}\big(j, k_1 + w_{ek_1}, k_2 + w_{ek_2}, \cdots, k_r + w_{ek_r}\big) \right\},$$

where $p = 1, \ldots, |V| - 2$ and $e = (i, j)$.

The optimal choice when at node i after p moves is to move to a node j^* that minimizes $l_e + f_{p+1}(j, k_1 + w_{ek_1}, k_2 + w_{ek_2}, \cdots, k_r + w_{ek_r})$, where $e = (i, j)$, over all $j = 1, \ldots, |V|-1$. The above dynamic programming algorithm solves the shortest r-weight-constrained path problem. We can ignore the dynamic program tabulations $f_p(i, k_1, \ldots, k_r)$, where $k_q > K_q$, for some $q = 1, \ldots, r$.

Next we consider the running time. Because G is acyclic and directed, the graph has at most $|V|(|V| - 1)/2$ edges. This is also the maximum length of a path in G.

To find the longest r-weight constrained path, we can store the longest path p of each r-weight combination so far found in each vertex. The number of these combinations is $(K_1 + 1) \cdots (K_r + 1)$ and we can store the combinations in a $(K_1 + 1) \cdots (K_r + 1)$ dimensional table contained in each vertex.

If there are two different paths having different lengths but both having the same r-weight combination, we store the longer one into the table. Hence, the space consumption is $O(K_1 \cdots K_r |V|^2)$. We start from the source vertices and proceed towards the sink vertices. In each vertex, we check every incoming edge. There are at most $|V|^2$ edges, and thus, the running time is $O(K_1 \cdots K_r |V|^2)$. □

3.2 IKO with zero length radiation

The next theorem can be seen as a generalization of the Theorem 2.

Theorem 14 IKO $\in$ P *when $n = O(1)$ and $u = 0$.*

Proof. Let c_j, $j = 1, \ldots, n$, be the length of a clone of item j. We construct a graph as follows: let s be the source vertex and t the target vertex. We form m groups of vertices each consisting of at most $(m+1)^n$ vertices. Hence, the total number of vertices is at most $2 + m(m+1)^n$.

Each vertex in a group has label $x_{k_1,\ldots,k_n}$, where $0 \le k_i \le c_i$, for each $i = 1, \ldots, n$. Because $c_i \le m$, we have at most $(m+1)^n$ labels.

If $k_1, \ldots, k_n > 0$, we insert an edge $(x_{k_1,\ldots,k_n}, x_{k_1-1,\ldots,k_n-1})$ from group l to group $l+1$, where $1 \le l \le m-1$.

If an index $k_q = 0$, for some $q = 1, \ldots, n$, we insert two edges from group l to group $l+1$, where $1 \le l \le m-1$. The edges are $(x_{k_1,\ldots,k_q=0,\ldots,k_n}, x_{k_1-1,\ldots,k_q=0,\ldots k_n-1})$ and $(x_{k_1,\ldots,k_q=0,\ldots,k_n}, x_{k_1-1,\ldots,k_q=c_q,\ldots,k_n-1})$. The latter edge corresponds to the inclusion of item $l+1$ into the q^{th} knapsack.

If $r > 0$ indices equal zero, we insert 2^r edges. A vertex group contains at most $(m+1)^n$ vertices, and hence, the total number of edges between two groups is at most $2^n(m+1)^n$. The length is calculated only for these vertices and it corresponds to the profit obtained by including the r items indicated by the vertex label. Other edges have zero length. The n-weight vector has ones in the indices indicated by the r items, other positions have zeros.

We insert an edge only if it complies with constraint (8). If the label of a vertex contains r non-zero indices, we can check (8) in $O(r)$ time, because we do not have to worry about radiations of the items before or after the items corresponding to the edge (that is, to the choice made for the knapsack).

We can insert an item into a knapsack only if some index is zero. Combining this with the edges related to nonzero indices ensures that (9) cannot be violated.

The source vertex s has at most 2^n outgoing edges to the vertices of the first group. Targets of the edges correspond to the different selections of n items we can make in the first knapsack. Every vertex of the last group has ongoing edge to the target vertex t.

We have constructed an acyclic directed graph with $O(2^n + 2^n(m+1)^n(m-1) + (m+1)^n) = O(2^n m^{n+1})$ edges. The term 2^n is for edges from s to the first vertex group. Then there are $m-1$ vertex groups, where each group contains $(m+1)^n$ vertices and each vertex has at most 2^n ongoing edges. Last, we have $(m+1)^n$ edges from the last vertex set to t. The graph has at most $2 + m(m+1)^n$ vertices. We can construct the graph in $O(2^n n + 2^n(m+1)^n(m-1)n + (m+1)^n n) = O(2^n m^{n+1} n)$ time. The n's are for the restriction checks.

Next we seek for the longest n-weight constrained path, where the weight constraint is K for each item (obtained from restriction (11)). Hence, we have to update $(K+1)^n$ different weight combinations in $2 + m(m+1)^n$ vertices. By Lemma 13, this can be done in polynomial time, as $K \le m/(c_i+1)$ (and n is fixed). $\square$

The edges from $x_{0,\dots,0}$ in group l to group $l+1$, where $1 \le l \le m-1$, have the same target vertices as the edges departing from vertex s. This allows, for example, a solution without any selected item; we just go from s to vertex $x_{0,\dots,0}$, and go to the vertex with the same label in every vertex group and finally arrive to t.

Figure 5 depicts the graph construction of Theorem 14. We have drawn the source and target vertices s and t as well as the first three vertex groups, and the last group of vertices. In this example we have three items such that $c_1 = 2$ and $c_2 = c_3 = 1$. The lengths of clones are 3, 2 and 2, respectively. From the source vertex s we can move, say, to the vertex 322 corresponding to the choice $x_{11} = x_{12} = x_{13} = 1$ (all three items are selected into the first knapsack). Other seven vertices to be chosen are 320, 302, 022, 300, 020, 002 and 000. These eight vertices in group l ($2 \le l \le m$) are also the targets of edges departing from vertex v_{000} from the previous vertex group $l-1$.

Consider vertex 010 in vertex group 2. We can insert the first and the third item into knapsack 2 corresponding to group 2. Item two is inserted into knapsack 1, and hence, we can insert it again into knapsack 3. This is shown as four edges departing from 010. The target vertices are 302, 300, 002, and 000. Note that the only feasible path to 010 goes through 020 in the first vertex group. By fine tuning the algorithm, we can take this into account. The longer the clone lengths, the larger the savings in running time.

Similarly, we can go from 320 to the vertices 210 or 212. The first target corresponds to a situation where item 3 is not inserted to the third knapsack. The second target, on the other hand, means that the item is inserted.

The rest of this subsection concerns the complexity of the presented method. If we neglect the number of edges, the construction of the graph and the search of the longest path by Lemma 13 gives $O\big(2^n m^{n+1} n\big) + O\big(K^n (2 + m(m+1)^n)^2\big)$ running time. This is a rough upper bound.

When every index is non-zero, there is only one edge. The number of these vertices in a group is at most $(m+1)^n$. There are 2^n vertices with one or more zeros in the indices. When there are l zeros, we insert 2^l vertices in $O(2^l n)$ time (we check restriction (8) in $O(n)$ time). Hence, the vertices having one or more zero indices imply $\sum_{l=1}^{n} 2^l \binom{n}{l}$ edges in one group in $O\big(\sum_{l=1}^{n} 2^l \binom{n}{l} n\big)$ time.

By the binomial theorem, the number of these edges is 3^n. Hence, we have at most $(m+1)^n + 3^n$ edges between two groups, which we can form in $O\big((m+1)^n n + 3^n n\big)$ time, giving $O\big(2^n n + m((m+1)^n n + 3^n n) + (m+1)^n n\big) = O\big(((m+1)^{n+1} + m3^n)n\big)$ total time to construct the graph. This is much less than $O\big(2^n (m+1)^{n+1} n\big)$ used in Theorem 14.

We can store the n-weight combination tables of the vertices for one vertex set at a time. We update the longest paths as we proceed from vertex set to the next

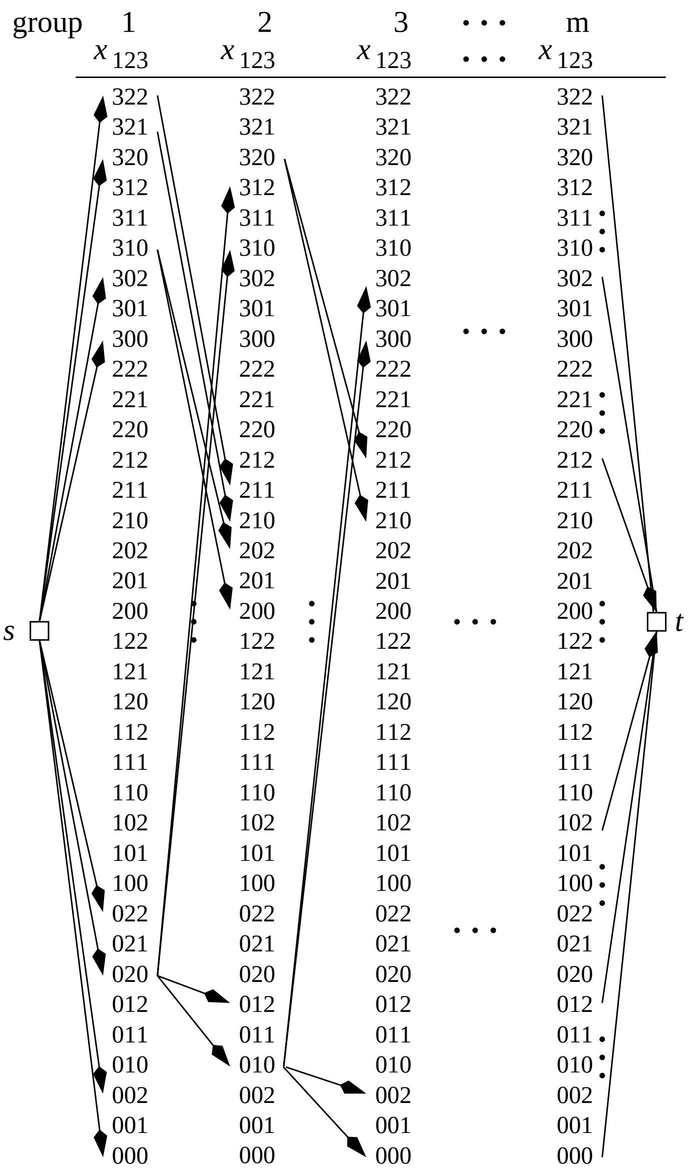

Figure 5: The graph structure for Theorem 14 with three items such that $c_1 = 2$, $c_2 = 1$ and $c_3 = 1$ (and each $u_i = 0$).

one. Each update takes a constant time, as the number of vertices with maximum lengths in the indices is the same as the number of vertices with zeros, 2^n, and they are the targets of 3^n edges departing from the vertices with zeros (and n is fixed). We also update a pointer to the longest path. Finally, in the target vertex t we use the pointer to the longest path in constant time.

Thus, we save some space and time leading to running time $O\big(2^n + m((m+1)^n + 3^n)K^n + (m+1)^n\big) = O\big(K^n((m+1)^{n+1} + m3^n)\big)$ for the longest path. The summands stand for the edges from s, then m times between groups, where we have at most $(m+1)^n + 3^n$ edges, and the edges to t, respectively. Hence, the total time to construct the graph and to search the longest path is $O\big(n((m+1)^{n+1} + m3^n)\big) + O\big(K^n((m+1)^{n+1} + m3^n)\big) = O\big((K^n + n)((m+1)^{n+1} + m3^n)\big)$.

In the following corollary we suppose that each item has its own clone length c_j, $j = 1, \dots, n$. Let c be the longest maximum length. We do not prove the corollary here, as the proof closely resembles that of Theorem 14, in which we used the fact that $c_j \leq m$. However, the interesting instances are such that c_j is much smaller than m: we can insert item at most $m/(c_j + 1)$ times into the knapsack array.

Corollary 15 *When $u = 0$,* IKO *can be solved in time*

$$O\big((K^n + n)m((c_1 + 1) \cdots (c_n + 1) + 3^n)\big).$$

3.3 IKO in polynomial time with non-zero radiation length

The results in the previous subsection have practical implications, at least for the load clipping problem (for example in the case of generators and street lights). Even though the previous results are contained in the following, we can save some time by using the results given in the previous subsection, since we can construct the edges more directly than in the method to be presented next.

The following theorem is analogical to Theorem 10. This time, we have n items, which we take into account in the labels of the vertices. Compared to Theorem 10, we have more edges between vertex sets. Additional edges with the additional restrictions on the weights on edges affect to the running time analysis, too.

We suppose that the items have identical lengths (later we relax this assumption). The graph to be constructed has vertices $v_{x_{ij} \cdots x_{i+\ell,j}}$, where $x_{11,\dots,\ell 1,\dots,1n,\dots,\ell n}$ has 2^ℓ different bit combinations. Each vertex corresponds to a selection of n items into ℓ consecutive knapsacks.

Lemma 16 *Let the maximum lengths of clone and radiation parts be ℓ for each item. Every item selection $v_{11,\dots,\ell 1,\dots,1n,\dots,\ell n}$ (ignoring any other items in other knapsacks) can be checked against restrictions* (8) *and* (9) *in $O(n\ell 2^{n\ell})$ time.*

Proof. Consider a selection $v_{11,\ldots,\ell 1,\ldots,1n,\ldots,\ell n}$. We are examining ℓ consecutive knapsacks: there can at most ℓ ones, and the effects of an item can be calculated in $O(\ell^2)$ time (and at the same time we can check the restrictions). Because we have n items, a selection $v_{11,\ldots,\ell 1,\ldots,1n,\ldots,\ell n}$ can be checked against (8) and (9) in time $O(n\ell^2)$.

Next consider a set of all selections containing $2^{n\ell}$ elements of length $n\ell$. Combining two bit strings (and the corresponding results in the knapsacks) takes $O(2n\ell)$ time.

The rest of the proof closely follows that of Lemma 6 and we conclude that the total time consumption is $O(n\ell 2^{n\ell})$. □

The proof of the following theorem mimics the proofs of Lemmas 7–9.

Theorem 17 IKO $\in$ P *when* $n = O(1)$, $\ell = \log(m)$.

Proof. (A sketch.) Let I_{ij}, p_{ij}, w_{ij}, b_j, K, n, and m form an instance of IKO. We suppose also that the total length of radiation and clone parts is $\ell = \log(m)$ for each item.

We start from the source vertex s and insert at most $2^{n\ell} = m^n$ outgoing edges to the first vertex set. (Note that $m = 2^\ell$.) The vertices in the first vertex set correspond to every combination to select n items into the first ℓ knapsacks. If some combination is impossible, to comply with (8) and (9), the corresponding edge is not inserted. By Lemma 16, the first edges take $O(n\ell 2^{n\ell})$ time.

We insert an edge $(v_{x_{ij}\cdots x_{i+\ell,j}}, v_{x_{i+1,j}\cdots x_{i+\ell+1,j}})$ between consecutive vertex sets, if $v_{x_{i+1,j}\cdots x_{i+\ell+1,j}}$ is a legal item selection and does not violate (8) and (9).

The profit and restriction checks for every vertex between two consecutive vertex sets is done in $O(n\ell 2^{n\ell})$ time. We construct the edges between feasible vertices in $O(2^n 2^{n\ell})$ time. Hence, the total time needed to make edges between two vertex sets is $O((2^n + n\ell)2^{nl})$.

We use radiations and clone lengths as in the one item case. That is, for each item, the bit corresponding to the first position of the clone part (right after u_l part) tells, if we should calculate the effects of including the item (length for the edge).

Also weights are set similarly to the one item case. This time, each item has its own coordinate in the n-weight vector. If item j is included ($x_{i+u_l+1,j} = 1$), the corresponding j^{th} bit in the resource bound of the edge is one, otherwise it is zero.

Now we have a system that contains every selection of length u in the knapsack array with n items. We have linked selections so that every legal selection is attainable. The number of selections is $O(m 2^{n\ell}) = O(m^{n+1})$. This bounds the total number of vertices in the vertex sets. The number of edges between two consecutive vertex sets is $2^n 2^{n\ell} = 2^n m^n$ (we have 2^n decisions for selections, which can be in $2^{n\ell}$ states in ℓ knapsacks). The graph construction takes totally $O\big(n\ell 2^{n\ell} + m(2^n + n\ell)2^{n\ell} + n\ell 2^{n\ell}\big) = O\big((2^n + n\log(m))m^{n+1}\big)$ time.

Next we have to find the best set of selections. Condition (11) restricts, for each item, the number of ones a path can contain. This leads to the r-weight-constrained longest path problem. Recall that our graph is acyclic and directed. As K is polynomially bounded (on m), we can form the longest r-weight-constrained path in polynomial time (on m) by Theorem 13 (recall that n is fixed). $\square$

To obtain the second set of vertices from the first set, we drop the first knapsack and add a new knapsack $\ell+1$. As with Theorem 10, the consecutive vertex sets contain $\ell-1$ common knapsacks. Let the common subselection be $x_{2j},\ldots,x_{\ell j}$, $(j=1,\ldots,n)$. The items in the first knapsack x_{1j} as well as the items in the new knapsack $x_{\ell+1,j}$.

Figure 6 demonstrates the case $n=2$ and $\ell_1=3$ and $\ell_2=2$. Each vertex set has 32 vertices and there are 4 edges departing each vertex. Moreover, we have to consider every 2-weight combination when searching for the longest path. The size of the combination table is $(K+1)^2$. Figure 6 also demonstrates, how to handle the last vertex set, when every item has its own total length ℓ_i.

As an example, consider vertex $x_{111,11}$ at vertex set "123,12", and suppose that $u_l=0$. This corresponds to a situation, where we include items 1 and 2 into the first two knapsack and item 1 also to the third knapsack. The length of edges $(s, v_{111,11})$ corresponds to the inclusions $x_{1,1}=1$ and $x_{1,2}=1$. We also know that we are going to include both items into the second knapsack. This is the common length of the four $(=2^2)$ edges departing from vertex set "123,12" to vertex set "234,23". Vertex $x_{110,10}$ in the second vertex set corresponds to a situation where item 1 is not inserted into the fourth knapsack and item 2 is not inserted into the third knapsack. Vertex $x_{110,11}$ corresponds to a situation where item 1 is not inserted into a fourth knapsack while item 2 will be inserted into the third knapsack.

To analyze the running time, note that the graph construction takes $O(m(2^n+n\ell)2^{n\ell})$ time. We have $O(m2^{n\ell})$ vertices and the number of edges is $O(m2^n2^{n\ell})$.

By Lemma 13, we need $O(m(2^n+n\ell)2^{n\ell}+K^n|V|^2)$ time, which equals to $O(m(2^n+n\ell)2^{n\ell}+K^n(m2^{n\ell})^2) = O(K^n m^2 m^{2n}) = O(K^n m^{2(n+1)})$. The obtained bound can be improved.

Remark, that because our graph has a special structure, the actual time consumption is smaller than given in Lemma 13. We can proceed from one vertex set to the next, and store the n-weight combination tables of dimension $O((K+1)^n)$ in m^n vertices of the vertex set.

A path contains $O(m)$ vertices, and a vertex set contains m^n vertices. Each vertex contains at most $(K+1)^n$ paths. Hence, the space consumption is $O(K^n m^{n+1})$. The longest path algorithm now runs in time $O(2^{n\ell})+O(K^n 2^n 2^{n\ell} m)+O(2^{n\ell}) = O(K^n 2^n 2^{n\ell} m) = O(K^n 2^n m^{n+1})$. The first term stands for the edges from the source, the second for the path checks with every weight combination in each vertex, and the

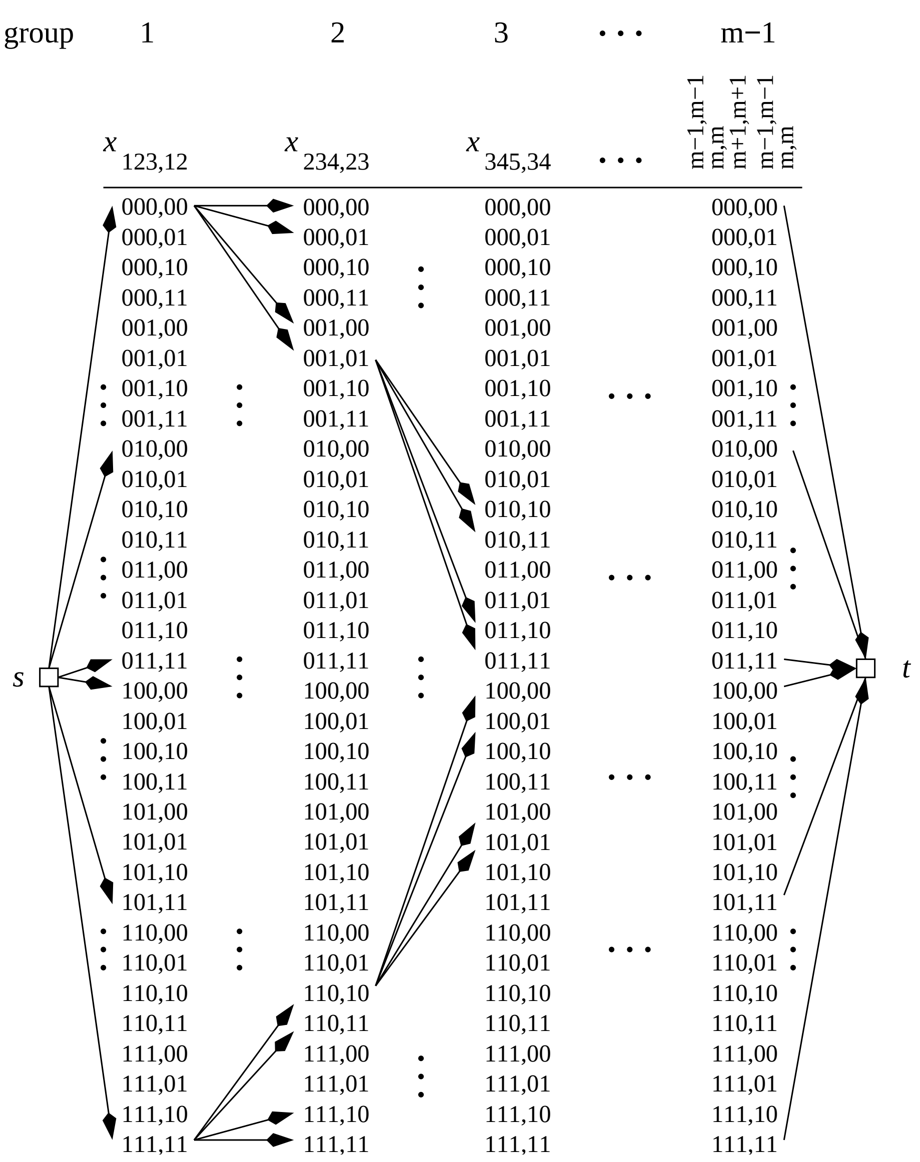

Figure 6: The graph structure for Theorem 17 with two items ($n = 2$), where $\ell_1 = 3$ and $\ell_2 = 2$. We have $m - 1$ vertex groups, because the minimum total length is two units.

last one is for the edges into the target vertex. Note that the paths in the n-weight combination table need to be traversed in $O(m)$ time only at vertex t when we seek the longest path. We can find the longest path in constant time, because we have kept a pointer to it.

Thus, the total time is $O\big(m(2^n + n\ell)2^{n\ell}\big) + O\big(K^n 2^n 2^{n\ell} m\big) = O\big((K^n 2^n + 2^n + n\ell)2^{n\ell} m\big) = O\big((K^n 2^n + 2^n + n\log(m))m^{n+1}\big)$. This is much less than the bound suggested by Lemma 13.

We can relax some of the assumptions in the proof. Each item can have its own maximum number K_i of inclusion times, and the total length ℓ_i (that is, the length of left and right radiation u_{il} and u_{ir}, and the length of clone c_i together). This means that we replace every $n\ell$ with $\ell_1 + \cdots + \ell_n$.

Corollary 18 IKO *can be solved in* $O\big((K_1 \cdots K_n 2^n + 2^n + \ell_1 + \cdots + \ell_n)2^{(\ell_1+\cdots+\ell_n)} m\big)$ *time.*

In the time bound of Corollary 18, $K_1 \cdots K_n$ is for filling up the n-weight combination table, while $2^{\ell_1+\cdots\ell_n}$ is for the number of vertices in one vertex set. The additional m is the number of vertex sets and 2^n is the in and out degree of each vertex in a vertex set. By relaxing $\ell = \log(m)$ to $\ell = \log(m^k)$ or to $\ell = k + \log(m)$, we obtain the following.

Corollary 19 *If* $\ell = \log(m^k)$*,* IKO *can be solved in* $O\big((K^n 2^n + 2^n + nk\log(m))m^{nk+1}\big)$ *time. If* $\ell = k + \log(m)$*,* IKO *can be solved in* $O\big((K^n 2^n + 2^n + n(k + \log(m)))2^{nk} m^{n+1}\big)$ *time.*

Proof. If $\ell = \log(m^k)$, we have $2^{nk\log(m)} = m^{nk}$ and $O\big((K^n 2^n + 2^n + n\ell)2^{n\ell} m\big) = O\big((K^n 2^n + 2^n + n + k\log(m))m^{nk+1}\big)$.

If $\ell = k + \log(m)$, then $2^{n(k+\log(m))} = 2^{nk} m^n$. Now $O\big((K^n 2^n + 2^n + n\ell)2^{n\ell} m\big) = O\big((K^n 2^n + 2^n + n(k + \log(m)))2^{nk} m^{n+1}\big)$. □

If $K = m/(c + 1)$ for each item, we do not use restriction (11). In these cases we can use the shortest path methods (our graph is directed and acyclic), and the result can be formulated as follows:

Corollary 20 *Suppose that* $K = m/(c+1)$ *for each item. If* $\ell = \log(m^k)$*,* IKO *can be solved in* $O\big((2^n + nk\log(m))m^{nk+1}\big)$ *time. If* $\ell = k + \log(m)$*,* IKO *can be solved in* $O\big((2^n + n(k + \log(m)))2^{nk} m^{n+1}\big)$ *time.*

Proof. If $\ell = \log(m^k)$, the number of vertices is $2 + m^{nk} m = 2 + m^{nk+1}$ and the number of edges is $m^{nk} + 2^n m^{nk} O(m) + m^{nk} = O(2m^{nk} + 2^n m^{nk+1})$. The first term is for the edges from s, the next is for the edges between vertex groups, and the last term is for the edges incident with t. Thus, the running time for the shortest path method is

$O(|V|+|E|) = O(2+m^{nk+1}+2m^{nk}+2^n m^{nk+1}) = O(2^n m^{nk+1})$. The construction time together with the path search time gives $O\big(m(2^n+nk)\log(m)m^{nk}\big)+O\big(2^n m^{nk+1}\big) = O\big((2^n+2^n+nk\log(m))m^{nk+1}\big)$.

If $\ell = k+\log(m)$, the number of vertices is $2+2^{nk}m^n m$. Thus, the shortest path running time is $O(2+2^{nk}m^{n+1}+2^{nk}m^n+2^n 2^{nk}m^n(m-1)+2^{nk}m^n) = O((2^n+1)2^{nk}m^{n+1}) = O(2^{nk+n}m^{n+1})$. The terms are obtained as in the previous paragraph. The total time is $O\big(m(2^n+n(k+\log(m)))2^{nk}m^n\big)+O\big(2^{nk+n}m^{n+1}\big) = O\big((2^n+2^n+n(k+\log(m)))2^{nk}m^{n+1}\big)$. □

4 New instances of MDKP and IP solvable in polynomial time

In this section we present new instances of MDKP and IP that are polynomial time solvable. We first argue that IKHO and 0–1 MDKP are equally hard on the approximability sense. It demonstrates the close relationship between the problems. Thereafter, it is easy to see how to apply the earlier methods to 0–1 MDKP. We also extend the methods to 0–n MDKP, where each decision variable ranges from 0 to 2^n-1.

We have earlier shown that 0–1 MDKP can be polynomially reduced to IKHO [4]. Let W' be the integer weight matrix and p' integer profit vector of 0–1 MDKP of n items and m knapsacks (we are to maximize $p'x$ such that $W'x \leq b'$). In IKHO, set $I_i(k) = w'_{ik}$ and $w_i = 1$. Since the knapsack capacities can be the same for both problems, we have $b_i = b'_i$.

Now, we should find profits p_i such that $p'_i = \sum_{k=i-u}^{i+c+u} I_i(k)p_k$. If we could set $p = W'^{-1}p'$ we would have profits p. If W were invertible square matrix, we would obtain p as above. In this case we should set $u = m$ in order to calculate profits p' correctly. If W is not a square matrix, the use of a matrix pseudo-inverse (see, for example [23]) is difficult, or maybe even impossible. To avoid the difficulties caused by non-square weight matrices we could add some dummy knapsacks or items to the instance, so that the weight matrix is square and that a solution for the new instance implies a solution for the original instance of 0–1 MDKP. To avoid the difficulties caused by the non-invertible matrices, we can remove some of the knapsacks and store the dependency information so that the solution to the original case would be obtainable.

The above reasoning vaguely argues that IKHO is as hard as 0–1 MDKP, also in general, and not only in the case of invertible square weight matrices. On the other hand, we can think that selecting an item in 0–1 MDKP means that we add p'_i to the objective. Similarly, we could think that p'_i will be added after inserting an item in IKHO. The profit p'_i can be taken care in the graph construction although we do not have separate profit vector p that works with the radiation. We can think that in a way "$p' = W'p = W'(W'^{-1}p')$", even though the $W'^{-1}p'$ cannot be constructed for every instance of 0–1 MDKP.

Consider next the instances of 0–1 MDKP having m items and m knapsacks b'_i, with profits p'_i, and where the restriction matrix W' is a band matrix with bandwidth $O(\log(m))$. (In a band matrix W' we have $w'_{ij} = 0$, for $|i - j| \geq w$, where w is the width.)

We form the "selection graph" used in Section 2.2 from the given 0–1 MDKP instance and use the restrictions normally. As above, we set interactions $I_i(k) = w'_{ik}$, weights $w_i = 1$, and knapsack sizes $b_i = b'_i$. The profits for the lengths of edges are obtained directly from the profits of the present 0–1 MDKP case. Hence, restriction (3) is fulfilled, because $c = 0$, and (5) is not needed as $K = m/(c + 1)$ (we may include item into all knapsacks, if other restrictions allow it).

Bandwidth of w is seen as a clone (with clone and radiation part) of length $\ell = 1 + 2w$. Analogically to Corollary 12, we are able to estimate the running times as a function of the bandwidth.

Corollary 21 0–1 MDKP *having band matrix restriction of width* $w = k + \log(m)$ *can be solved in time* $O\big((k + \log(m))2^{2k+1}m^3\big)$. *Width* $w = \log(m^k)$ *implies time* $O\big(k\log(m)m^{2k+1}\big)$.

Proof. As $K = m/(c+1)$, we do not need (5). Hence, we use Corollary 12, which implies $O\big((v + t\log(m))2^v m^{t+1}\big)$ running time with total length $\ell = v + t\log(m)$. Now, set $\ell = 1 + 2k + 2\log(m)$, (thus $t = 1 + 2k$ and $v = 2$), giving $O\big((1 + 2k + 2\log(m))2^{1+2k}m^{2+1}\big) = O\big((k + \log(m))2^{2k+1}m^3\big)$. Further, $\ell = 1 + 2k\log(m)$ gives $O\big((1 + 2k\log(m))2^1 m^{2k+1}\big) = O\big(k\log(m)m^{2k+1}\big)$. □

Recall Theorem 13. It can be used to solve 0–1 MDKP instances. The above case needs $O(K^m m)$ time with a graph having m pairs of edges each corresponding to either selecting an item or not, each edge having m restrictions, and K being the largest number (or the largest size of the m knapsacks). (See also [33].) Even though we have neglected to consider the lengths of elements of instances, we dare to say that Corollary 21 gives much more efficient algorithm for the instances considered.

As an example, consider the following 0–1 MDKP instance. Let $p'_i = i$ ($i = 1, \ldots, 10$), and let the width of the band matrix be 1. Suppose also that $w'_{i,i-1} = w'_{ii} = w_{i,i+1} = i$. Last, let $b'_i = 30$ (hence, every edge will be shown). Figure 7 shows the edges, all directing from left to right. In this example, radiation has length one to the left and length one to the right from the clone part, whose length is one. Thus, a selection is three knapsacks wide and the middle knapsack gives the profit

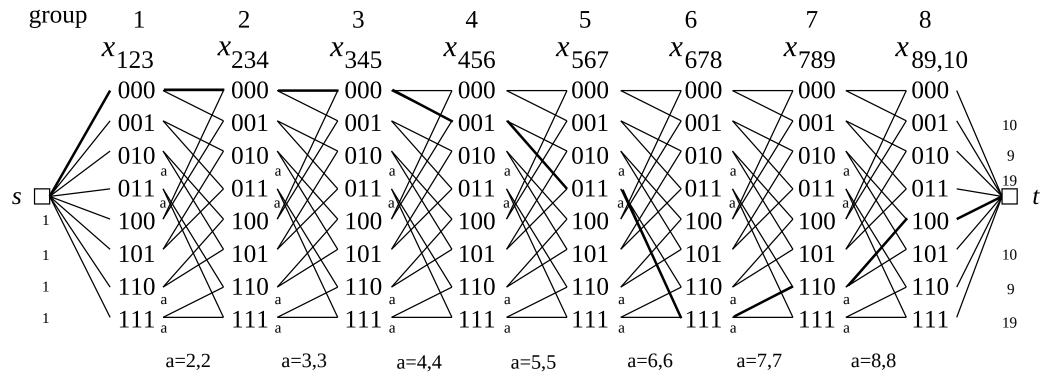

Figure 7: The graph obtained for 0–1 MDKP example. We have 10×10 matrix, whose band-width is 1. The thicker line shows a solution path corresponding to $x_i = 0$, $i = 1, \ldots, 5, 9, 10$, and $x_6 = x_7 = x_8 = 1$.

for the edges, except for the edges from s (the leftmost knapsack gives the profit) and to t (the second and third knapsack).

Figure 7 shows the length (the profit), if it is not zero. For example, the edge from s to 110 has length 1, because we take into account the profit of the inclusion of the first item (u_l part). Note that we do not use the weights here, as we are allowed to make as many inclusions as we want.

We have marked with a small "a" the edges that have non-zero length. For example, the edge from 011 in the first vertex set to 110 in the second vertex set has length 2, because the edges from the first to the second vertex set take into account the profits of the inclusion of the second item. There are two numbers for each "a", as there are two edges from each vertex in the vertex sets, for example, edges (011,110) and (011,111) between each vertex set.

Edges going to the target vertex t have different weights. There we have to add also the profits of inclusions of the tenth item in addition to the ninth item.

When we construct the graph, we check restriction (2) against every selection. For example, vertex 111 in the sixth vertex set corresponds to $x_6 = x_7 = x_8 = 1$. If some of the restrictions (2) were violated, some of the edges would be missing.

If we want to model the solution where only items 6, 7, and 8 are included, our path starts from s and goes to the upper most vertex of the first vertex set. The path continues to 000 in the third set, 001 in the fourth, and 011 in the fifth. The seventh knapsack has weight 6+7+8=21. The path continues from sixth set to 110 and to 100, from which it goes to t. Figure 7 draws this path in thicker line.

The profits corresponding to the selection $x_6 = x_7 = x_8 = 1$ (and $x_i = 0$, where $i \neq 6, 7, 8$) are given by the edges $(011, 111)$ with profit six (from fifth to sixth set), $(111, 110)$ with profit seven (from sixth to seventh set), and $(110, 100)$ with profit eight (from seventh to eighth set).

After applying the method designed for IKHO to 0–1 MDKP, we would like to apply the proof of Corollary 3 accordingly. The counterparts of m restrictions (3) of IKHO in 0–1 MDKP are knapsack restrictions $x_i + \cdots + x_{i+c_i} \leq 1$, for $i = 1, \ldots, m$, when inserting an item into the knapsack i in IKHO. If we set $x_i = 1$, then $x_{i+1} = \cdots = x_{i+c_i} = 0$. Knapsack $i + c_i + 1$ can take the next item. When we construct the graph, we draw an edge from (A_i, B_{i+c_i+1}) with length p_i. As there are m rows, we draw m edges corresponding to the restriction rows among other edges. Restriction (5) of IKHO can be kept unchanged in 0–1 MDKP. If 0–1 MDKP contains at least one such restriction, then we use the longest weight-constrained path algorithm. Otherwise, we use the longest path algorithm. As the graph is directed and acyclic, we use the shortest path algorithm.

The above application of Corollary 3 restricts 0–1 MDKP instances considerable and therefore, it is not as interesting as the result given in Corollary 21.

The above construction resembles the linear programming problem, where the restrictions are difference constraints [15]. That is, the restrictions are of form $x_j - x_i \leq b_k$. Difference constraints can be used to model, for example, relative timing constraints. Shortest paths can be applied to find a feasible solution for difference constraints [15]. However, if we are to account profits p_i, when $x_i = 1$ (or when $x_i \geq 1$), the application of shortest paths for difference constraints is not so obvious.

The above results raise another natural question at extending the algorithms of IKO instances. Let an instance of 0–1 MDKP consist of nm items and m knapsacks, where the restriction matrix W' is a concatenation of n band matrices: $W' = [W'_1\ W'_2\ \cdots\ W'_n]$, where W_i $(i = 1, \ldots, n)$ is an $m \times m$ band matrix with a bandwidth of size $O(\log(m))$.

Let w^k_{ij} be the weight of position i, j of W'_k. Now we set parameters for IKO: $w_{ij} = 1$, interactions $I_{ij}(k) = w^j_{ik}$, and $c = 0$, $K = m/(c+1)$ and $b_i = b'_i$. Again, we do not map profits from an 0–1 MDKP instance directly to an IKO instance, but rather use them as lengths in the constructed graph.

Corollary 22 0–1 MDKP *with n "concatenated" band matrix restrictions of width $\ell = \log(m^k)$ can be solved in time $O\big((2^n + n + 2kn\log(m))2^n m^{2kn+1}\big)$. Width $\ell = k + \log(m)$ implies time $O\big((2^n + n + 2nk + 2n\log(m))2^{n+2k} m^{2n+1}\big)$.*

Proof. Because $K = m/(c+1)$, we do not need restriction (11). By combining the results of Corollary 20, we obtain that $\ell = v + t\log(m)$ implies $O\big((2^n + n(v + t\log(m)))2^{nv} m^{nt+1}\big)$ time. We map ℓ to $1 + 2\ell$, that is, set $v = 1$ and $t = 2k$ for the first result and obtain $O\big((2^n + n(1 + 2k\log(m)))2^{n\cdot 1} m^{n\cdot 2k+1}\big)$.

Set $v = 1 + 2k$ and $t = 2$ for the second result and now $O\big((2^n + n(1 + 2k + 2\log(m)))2^{n(1+2k)}m^{2n+1}\big) = O\big((2^n + n + 2nk + 2n\log(m))2^{n+2k}m^{2n+1}\big)$. $\square$

Corollary 22 solves in polynomial time also a class of instances of 0–n MDKP, where the weight matrix is an $m \times m$ square matrix and has bandwidth $u = O(\log(m))$, and where each variable can range between 0 and $2^n - 1$ and n is fixed. (In 0–n MDKP we are to maximize pz such that $Wx \le b$ and $x_i \in \{0, \ldots, n\}$.)

We form a "concatenated" matrix from n matrices. Each submatrix has a factor 2^i, $i = 0, \ldots, n-1$, which is used to multiply the profits and weights. Now we have a situation in which Corollary 22 can be applied. The concatenated band matrix restrictions are $W'' = [2^0W' \; 2^1W' \; \cdots \; 2^{n-1}W']$. Similarly, the profit vector is $p'' = [2^0p' \; 2^1p' \; \cdots \; 2^{n-1}p']$. Knapsack capacities are the same. The value of the variable j is obtained by summing appropriately over the values at each index j of n parts. Because our graphs are directed and acyclic, the above discussion and results hold also for the 0–1 and 0–n IP problems.

For example, suppose an instance of 0–n IP with one band matrix of width $u = 1$ and the parameters given as in the 0–1 MDKP example: $p'_i = i$ $(i = 1, \ldots, 10)$, $w'_{i,i-1} = w'_{ii} = w_{i,i+1} = i$, and $b'_i = 30$. Suppose that each variable can range between 0 and $7 = 2^3 - 1$.

Hence, the concatenated band matrix is $W'' = [1W' \; 2W'' \; 4W'']$ and the profit vector is $p'' = [1p' \; 2p' \; 4p']$. Here W'' is a 10×30 matrix and p'' is a 30×1 vector.

The graph has 10 vertex groups, each containing $2^{3 \cdot 3} = 512$ vertices. Moreover, we have $2^3 \cdot 512$ edges between two consecutive vertex groups. Vertex s has 512 outgoing vertices and vertex t has as many incoming edges. The profits for the edges from s are given by the u_l part, that is by the left most knapsack.

For example, vertex $100, 110, 100$ in the first vertex group corresponds the decision $x_1 = 7$, $x_2 = 2$, and $x_3 = 0$. The length of the edge $(s, v_{100,110,100})$ is $1 \cdot 1 + 2 \cdot 1 + 4 \cdot 1 = 7$. The first knapsack has weight $1 \cdot 1 + 2 \cdot 1 + 4 \cdot 1 = 7$ from the decision $x_1 = 7$ and $2 \cdot 2$ from the decision $x_2 = 2$, totalling 9.

As an another example, consider vertex $011, 001, 001$ in the second vertex set. This vertex corresponds to the decision $x_2 = 0$, $x_3 = 1$, and $x_4 = 7$. Weight of the third knapsack can be calculated: $1 \cdot 3 + 7 \cdot 4 = 31$, which is more than $b_3 = 30$. Hence, there is no incoming edges to this vertex. Note that we can give some estimations about the weights in knapsacks 2 and 4, but the exact amount can only be calculated in the vertex sets 1 and 3, respectively. If the knapsack size of the third knapsack were larger, $011, 001, 001$ would have incoming edges of length 0, since the edges have to depart from vertices such that the middle bits are zeros (we see this from the vertex label, the first positions are zeros).

5 Conclusion

We have constructed polynomial time algorithms for IKHO, IKO, 0–1 MDKP (and 0–1 IP) and 0–n MDKP (and 0–n IP), n fixed, which have special but easily recognizable weight structure on the item weights. The structures have a size parameter that characterizes the running times of different instances. Eventually, we can even characterize the classes of instances, where the size parameter is not fixed. Similarly, the growth in n in the several item problems is described as well.

We have left some problems open. For instance the running time analysis handling the sizes of instance parameters concerning individual weights, profits, and knapsack sizes. In some applications this is not crucial and we may assume that parameters fit into a single memory word and can be read and written in constant time. Obviously, this assumption does not hold in all applications.

Next we list some fruitful research directions.

A question related to load clipping application is the variable clone length c. Some of the controllable utilities have minimum and maximum control lengths and the optimization method should determine the number of controls, starting times and lengths. The problems (1)–(5) and (7)–(11) can be easily modified to allow the decision of the clone length. The proof of Theorem 2 describes a special case, where the utilization of the variable clone length is easy and efficient.

We leave it open to incorporate the variable clone lengths to the other methods presented. Especially, we would like either to redesign the method given in the proof of Lemma 8, or to design a completely new method, or if neither of the approaches works, to establish some hardness results showing the inefficiency of the case handled in Theorem 10.

We can also pose some other natural questions related to the methods. Now the weight matrices have special structure that contains shape (band matrix, length of radiation and clone). The band matrix restrictions can be easily generalized to u_l and u_r lower and upper diagonals, but we would like to establish that the shape is not essential as long as the number of affected knapsacks is not too great.

Another line to improve the methods is connected to the clone length c: in the method of Theorem 2 we are able to jump several knapsack at a time while the other methods do not allow that. The reason is radiation. Even though we do not use radiation in Theorem 14, the method does not use long jumps. Further, it may be possible to replace the method behind Theorem 10 so that longer jumps based on c are possible and at the same time the radiation is handled correctly.

0–n MDKP application uses the same bounds for each decision variable. To enhance the usability of the presented approach, we would like to have separate ranges for each decision variable.

The presented methods may work well with some other models and methods. For example, difference constraints, which are restrictions of the form $x_j - x_i \leq b_k$, can solve problems containing relative timing constraints [15] (in polynomial time by using shortest path methods, see also [19]). In a way, we can interpret difference constrains as giving a minimum and maximum lengths for controls, which are pairs of consecutive variables: x_1 gives end time and x_2 the start time. We can also state that two consecutive controls cannot overlap by making restrictions $x_3 - x_2 \geq 0$.

If we want to use some objective function, the application of difference constraints is not straightforward. Moreover, the use of some extra constraints (knapsack capacities, for example) is not easy to model with difference constraints. However, it would be interesting to see, if one can efficiently add difference constraints somehow to the presented methods and thus broaden the class of problems that can be solved in polynomial time.

By choosing appropriate interaction functions I, profits p, weights w and knapsack capacities b, we can imitate some scheduling problems [1], like "sequencing with release times and deadlines on one processor" or "single machine scheduling" in the sense of [27]. Can we utilize the methods presented in this paper to those problems? And of course, we want to find out other problems, in which the presented methods work well.

Contrary to the above line of thought, we may also ask, how general is the applied approach of coding restrictions into the vertex sets and then search the path. It may be directly applicable to a large class of problems without using the IK-model or MDKP in between. We may try to obtain restriction matrices with desired properties, or to code restrictions directly to a similar graph we used.

In the above list, the proposal to design heuristics can use two approaches: we may try to reduce either the size of the vertex sets or the number of sets (and somehow ensure that the we comply with the problem restrictions). Other approach is connected to the applications: we may decrease their solution space somehow and then apply the presented methods to give exact solutions for the reduced instance.

Sometimes the application may imply a natural way to reduce the number of vertex sets. For example, in load clipping there may be time intervals, when there is no need to optimize. Even though we can recognize those intervals easily in load clipping, we have to be prepared to optimize the whole time interval. The running time analyses can be utilized here. If the actual optimization interval can be reduced enough, we may use the method presented for IKHO. Otherwise we may use larger set of controlling utilities (that is, use larger n in IKO). The decision is dynamic and cannot be tested beforehand, as opposite to comparing the running times beforehand to help the choice of an appropriate method.

Last, we can also turn our attention to the methods we utilized to achieve the given results. The shortest paths are much studied subject and we have efficient solutions for the directed acyclic graphs. Since our graph constructions are highly symmetric, it may be possible to find out the shortest paths more efficiently. Furthermore, we can save space by updating the paths at the same time as constructing graphs so that only a vertex set is kept in memory. It may give faster solutions (which is the case, for example, in the discussion preceding Corollary 18). Other possibility is to form paths depth first (see, e.g., [8], pp. 62–81) and apply, for example, branch and bound.

References

[1] Isto Aho. Interactive knapsacks. Technical Report A-1999-5, University of Tampere, Dept. of Computer Science, March 1999. [Online]. Available: http://www.cs.uta.fi/reports/r1999.html.

[2] Isto Aho. Interactive knapsacks. *Fundamenta Informaticae*, 44:1–23, 2000.

[3] Isto Aho. Notes on the properties of dynamic programming used in direct load control scheduling. Technical Report A-2000-12, University of Tampere, Dept. of Computer and Information Sciences, August 2000. [Online]. Available: http://www.cs.uta.fi/reports/r2000.html.

[4] Isto Aho. On the approximability of interactive knapsack problems. In *SOFSEM 2001: Theory and Practice of Informatics, 28th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 2234, pages 152–159, Piestany, November 2001.

[5] Isto Aho, Harri Klapuri, Jukka Saarinen, and Erkki Mäkinen. Optimal load clipping with time of use rates. *International Journal of Electrical Power & Energy Systems*, 20(4):269–280, 1998.

[6] J.E. Beasley. Population heuristics. Technical report, Imperial College, The Management School, 1999.

[7] J.E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[8] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts, 1996.

[9] Paola Cappanera. *Discrete Facility Location and Routing of Obnoxious Activities*. PhD thesis, Dip. di Matematica, Univ. di Milano, 1999. [Online.] Available: http://www.di.unipi.it/∼cappaner.

[10] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

[11] Ashok K. Chandra, D. S. Hirschberg, and C. K. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3:293–304, 1976.

[12] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, Baltimore, January 1999.

[13] Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *Proc. of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 213–222, San Francisco, January 2000.

[14] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

[15] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[16] Irina Dumitrescu and Natashia Boland. Algorithms for the weight constrained shortest path problem. *International Transactions in Operational Research*, 8:15–29, 2001.

[17] Martin Dyer, Alan Frieze, Ravi Kannan, Ajai Kapoor, Ljubomir Perkovic, and Umesh Vazirani. A sub-exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. *Combinatorics, Probability and Computing*, 2:271–284, 1993.

[18] Henning Fernau. Parameterized maximization. Technical Report WSI-2001-22, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2001.

[19] John P. Fishburn. Solving a system of difference constraints with variables restricted to a finite set. *Information Processing Letters*, 82:143–144, 2002.

[20] A.M. Frieze and M.R.B. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses. *European Journal of Operational Research*, 15:100–109, 1984.

[21] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[22] Minos N. Garofalakis, Banu Özden, and Avi Silberschatz. Resource scheduling in enhanced pay-per-view continues media databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 516–526, 1997.

[23] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, London, 1981.

[24] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17:36–42, 1992.

[25] Arild Hoff, Arne Løkketangen, and Ingvar Mittet. Genetic algorithms for 0/1 multidimensional knapsack problems. In *Proceedings Norsk Informatikk Konferanse, NIK '96*, 1996.

[26] Robert C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings of AI'2001 (the Canadian conference on Artificial Intelligence)*, Lecture Notes in Artificial Intelligence 2056, pages 57–66, 2001.

[27] Toshihide Ibaraki and Yuichi Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, July 1994.

[28] Edward Yu-Hsien Lin. A bibliographical survey on some well-known non-standard knapsack problems. *Information Systems and Operations Research*, 36(4):274–317, November 1998.

[29] Udi Manber. *Introduction to Algorithms, A Creative Approach.* Addison-Wesley, Reading, 1989.

[30] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, New York, 1990.

[31] Silvano Martello and Paolo Toth. Generalized assignment problems. *Lecture Notes in Computer Science*, 650:351–369, 1992.

[32] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1999.

[33] G.L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[34] Maria A. Osorio, Fred Glover, and Peter Hammer. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. Technical

Report HCES-08-00, The University of Mississippi, Hearin Center for Enterprise Science, 2000.

[35] Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28:765–768, 1981.

[36] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience Publication, Chichester, 1998.

[37] Aravind Srinivasan. Improved approximations of packing and covering problems. In *Proc. of the 27th ACM Symposium on the Theory of Computing*, pages 268–276, 1995.

[38] H. Martin Weingartner and David N. Ness. Methods for the solutions of the multidimensional 0/1 knapsack problem. *Operations Research*, 15:83–103, 1967.

[39] L. A. Wolsey. A view of shortest route methods in integer programming. *Cahiers du Centre d'Études de Recherche Opérationelle*, 16:317–335, 1974.

[40] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems.* PhD thesis, Universität des Saarlandes, 2001. [Online.] Available: http://www.mpi-sb.mpg.de/∼mark/.