A fast algorithm for strongly correlated knapsack problems

David Pisinger

Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

Received 27 February 1996; received in revised form 16 December 1997; accepted 28 July 1998

Abstract

We consider a variant of the 0–1 Knapsack Problem, where the profit of each item corresponds to its weight plus a fixed constant. These so-called Strongly Correlated Knapsack Problems have attained much interest due to their apparent hardness and wide applicability in several fixed-charge problems.

A specialized algorithm for the problem is presented, where the main approach is to derive an additional constraint from an extended cover. By surrogate relaxation with optimal multipliers, we obtain a Subset-sum Problem defined in the profits of the items. It is proved that an optimal solution to the Subset-sum Problem is also an optimal solution to the original problem provided that the largest possible number of items is chosen. Based on this observation, a 2-optimal heuristic is derived which solves the problem to optimality for several large-sized problems. In those cases where the heuristic fails, we solve the problem to optimality by restricting the problem to a fixed number of chosen items β. For each value of β the problem is solved through dynamic programming.

Extensive computational experiments are provided showing that we are able to solve strongly correlated instances faster than other algorithms solve uncorrelated instances. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Knapsack Problem; Fixed-charge Problem; Dynamic Programming; Memorization; Exact Algorithm

1. Introduction

We consider so-called Strongly Correlated Knapsack Problems, which may be characterized as ordinary 0–1 Knapsack Problems where the profit of each item corresponds to its weight plus (or minus) a fixed constant. These problems appear in the formulation of several fixed-charge problems (shipment fares, cutting-stock problems, investment planning), as well as by surrogate relaxation of Subset-sum Problems with additional constraints. Despite the relatively moderate coefficient sizes, Strongly Correlated Knapsack Problems seem to be very hard to solve. This has made them popular as benchmarks for new knapsack algorithms.
Recent techniques have made it possible to solve these problems even for large-sized instances, but the solution times are still hundreds of times larger than those for uncorrelated instances. Pandit and Ravi Kumar [5] presented a specialized algorithm based on lexicographical search, while Martello and Toth [4] gave a general knapsack algorithm which also was able to solve these instances. Finally Pisinger [8] obtained similar results for general Knapsack Problems by using dynamic programming.

This paper presents an algorithm which solves large-sized Strongly Correlated Knapsack Problems faster than state-of-art algorithms solve uncorrelated Knapsack Problems. In Section 2 it is shown that tight upper bounds may be derived by surrogate relaxation of the original weight constraint with a Balas cut [1]. The relaxed problem becomes a Subset-sum Problem when optimal surrogate multipliers are applied, and it is proved that an optimal solution to the relaxed problem also will be an optimal solution to the original problem, provided that it selects as many items as possible. Thus in Section 3 we use a 2-optimal heuristic to solve the Subset sum Problem, maintaining the maximal amount of items in all the solutions considered. It is proved that this 2-optimal heuristic will find an exact solution to the original problem with overwhelming probability for large problems. In cases where the 2-optimal heuristic does not find an exact solution, Section 4 presents a dynamic programming algorithm to solve the problem to optimality. The algorithm repeatedly solves a Subset-sum Problem with a fixed number of items, making it possible to tighten the upper bound considerably after each iteration. Global bounding rules are presented in Section 5 together with the main algorithm.

Extensive computational results are presented in Section 6, showing that all problems found in the literature may be solved hundreds of times faster than by competing algorithms. Finally Section 7 shows that the derived results are equally valid for Inverse Strongly Correlated Knapsack Problem, and the conclusion in Section 8 discusses how surrogate relaxation in connection with a Balas cut may be applied to obtain tight and quickly derived upper bounds for more general problems.

2. Strongly correlated problems

Strongly Correlated Knapsack Problems (SCKP) may be formulated as

\[
\begin{align*}
\text{maximize} & \quad z_1 = \sum_{j=1}^{n} (w_j + k)x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} w_j x_j \leq c, \\
& \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n,
\end{align*}
\]

where the weights \( w_j, \ j = 1, \ldots, n, \) are positive integers, and \( k \) is a positive constant. The problem is NP-hard, since for \( k = 0 \) we obtain the Subset-sum Problem. Assume that the items are ordered according to nondecreasing weights

\[
w_j \leq w_{j+1}, \quad \text{for } j = 1, \ldots, n - 1,
\]
and let the break item $b$ be defined by $b = \min \{ j : \sum_{i=1}^{j} w_i > c \}$. An LP-optimal solution is found by using the greedy algorithm: Since the profit-to-weight ratio of the items is nonincreasing, we simply set $x_j = 1$ for $j = 1, \ldots, b-1$ and $x_j = 0$ for $j = b+1, \ldots, n$ while $x_b = (c - \sum_{j=1}^{b-1} w_j)/w_b$.

The IP-solution which chooses the first $b - 1$ items will be denoted the break solution $x'$, thus $x'_j = 1$, $j = 1, \ldots, b - 1$ and $x'_j = 0$, $j = b, \ldots, n$. The weight of the break solution is $v' = \sum_{j=1}^{b-1} w_j$ and the residual capacity of the knapsack is $r = c - v'$. As the first $b$ items define a cover, we may impose the following additional constraint

$$\sum_{j=1}^{n} x_j \leq b - 1,$$

(3)

to problem (1) without excluding any optimal solution [1]. Since this two-constrained knapsack problem is very difficult to solve, we will surrogate relax (1) and (3) using multipliers $1$ and $S$, respectively. There is no loss of generality in this assumption, since the first multiplier always can be chosen as $1$ by appropriate scaling. Thus we get the relaxed problem

$$\maximize \sum_{j=1}^{n} (w_j + k)x_j$$

subject to $\sum_{j=1}^{n} (w_j + S)x_j \leq c + S(b - 1)$,

$x_j \in \{0, 1\}$, $j = 1, \ldots, n$.

(4)

An optimal solution to this problem with a given multiplier $S_1$ will also be a feasible solution for the problem with a different multiplier $S_2$ if $S_2 > S_1$, thus the tightest IP upper bound is obtained with $S = 0$. However, for the LP-relaxed problem we have:

**Theorem 1.** If the weights satisfy that

$$\sum_{j=1}^{b} w_j \leq \sum_{j=n-b+2}^{n} w_j$$

(5)

then the surrogate multiplier which yields the smallest objective value in the LP-relaxed version of problem (4) is $S = k$.

**Proof.** First notice that constraint (5) says that the weight sum of the first $b$ items is not greater than the weight sum of the last $b - 1$ items. Due to the ordering (2) this will always be the case when the weights have a reasonable variation: For instance if $w_1 + w_2 \leq w_n$ then the constraint is already satisfied.

The LP-relaxed problem is solved by the greedy algorithm, thus we must distinguish between three cases depending on which items have largest profit-to-weight ratios in each situation.

**$S < k$:** With the ordering (2) profit-to-weight ratios in (4) will be nonincreasing, thus an optimal solution to the LP-relaxed problem is: $x_j = 1$ for $j = 1, \ldots, b - 1$ and
$x_b = (c - \sum_{j=1}^{b-1} w_j)/(w_b + S)$. This yields the objective value

$$z_b = \sum_{j=1}^{b-1} (w_j + k) + \left( c - \sum_{j=1}^{b-1} w_j \right) \frac{w_b + k}{w_b + S}. \tag{6}$$

Thus the LP-solution will be decreasing for increasing values of $S$.

$S = k$: In this situation all items have the same profit-to-weight ratio, so we construct an alternative solution by setting $b' = \max\{i: \sum_{j=i}^{n} (w_j + k) > c + k(b - 1)\}$, in which case we have the optimal solution

$$x_j = 0, \ j = 1, \ldots, b' - 1, \quad x_j = 1, \ j = b' + 1, \ldots, n,$$

$$x_{b'} = \frac{c + k(b - 1) - \sum_{j=b'}^{n} (w_j + k)}{w_{b'} + k} < 1. \tag{7}$$

This solution has the objective value $z_\ell = c + k(b - 1)$.

$S > k$: First note that due to assumption (5) we have

$$\sum_{j=n-b+2}^{n} (w_j + k) = \sum_{j=n-b+2}^{n} w_j + k(b - 1) > \sum_{j=1}^{b} w_j + k(b - 1) > c + k(b - 1), \tag{8}$$

thus $b' > n - b + 2$, implying $n - b' < b - 2$. We want to prove that for $S > k$ Solution (7) is still valid for the LP-relaxed problem. We have

$$\sum_{j=1}^{n} (w_j + S)x_j = \sum_{j=b'+1}^{n} (w_j + k) + x_{b'}(w_j + k) + \sum_{j=b'+1}^{n} (S - k) + x_{b'}(S - k)$$

$$\leq c + k(b - 1) + (n - b')(S - k) + x_{b'}(S - k) < c + S(b - 1).$$

Thus for $S > k$ the LP-relaxed solution $z_\ell$ to (4) will satisfy $z_\ell \geq z_\ell$.

Notice that constraint (5) is necessary, since if, e.g. $w_j = d$ for all $j$, then $\sum_{j=1}^{b} w_j > \sum_{j=n-b+2}^{n} w_j$. But in this case all items have the same profit-to-weight ratio, and the bound (6) shows that the larger value of $S$ we choose, the tighter LP-solution we get in (4). Thus in this case we should choose $S \geq k$. \hfill \Box

Using the LP-optimal multiplier $S = k$, an upper bound $z_2$ to SCKP may be derived by solving the following Subset-sum Problem (SSP):

\begin{equation}
\begin{aligned}
\text{maximize} \quad & z_2 = \sum_{j=1}^{n} (w_j + k)x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} (w_j + k)x_j \leq c + k(b - 1), \tag{9} \\
& x_j \in \{0, 1\}, \ j = 1, \ldots, n.
\end{aligned}
\end{equation}
We have the obvious relation
\[ z_1 \leq z_2 \leq c + k(b - 1), \tag{10} \]
saying that we may use \( u = c + k(b - 1) \) as an upper bound for SCKP.

The following theorems show that a solution to SSP also is a solution to SCKP provided that some special constraints are satisfied:

**Theorem 2.** The two problems SCKP and SSP have the same break item.

**Proof.** The break item of SSP is defined by \( b' = \min\{j: \sum_{i=1}^{j} (w_i + k) > c + k(b - 1)\} \).
We have \( \sum_{j=1}^{b} (w_j + k) = \sum_{j=1}^{b} w_j + kb > c + kb > c + k(b - 1) \), meaning that \( b' \leq b \).
On the other hand \( b' \geq b \) since we have \( \sum_{j=1}^{b-1} (w_j + k) = \sum_{j=1}^{b-1} w_j + k(b - 1) \leq c + k(b - 1) \).

**Theorem 3.** An optimal solution \( x^* \) to SSP is also an optimal solution to SCKP provided that \( \sum_{j=1}^{n} x_j = b - 1 \).

**Proof.** An optimal solution \( x^* \) to SSP satisfies that \( \sum_{j=1}^{n} (w_j + k)x_j^* \leq c + k(b - 1) \), so from the assumption we have \( \sum_{j=1}^{n} w_j x_j^* = \sum_{j=1}^{n} (w_j + k)x_j^* - k \sum_{j=1}^{n} x_j^* \leq c + k(b - 1) - k(b - 1) = c \). In other words, \( x^* \) is a feasible solution to SCKP and it has the same objective value as the upper bound given by (10).

Thus solving SSP not only gives us an upper bound on the original problem, but also an optimal solution in those cases where the obtained solution \( x \) satisfies \( \sum_{j=1}^{n} x_j = b - 1 \).

3. **A two-optimal heuristic**

The previous results indicate that tight lower bounds may be obtained by fixing the number of items at \( \sum_{j=1}^{n} x_j = b - 1 \) in SCKP, as this collects the largest number of fixed-charges \( k \) in the objective function. The problem may thus be formulated as

\[
\begin{align*}
\text{maximize} & \quad z_3 = \sum_{j=1}^{n} w_j x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} w_j x_j \leq c, \\
& \quad \sum_{j=1}^{n} x_j = b - 1, \\
& \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n.
\end{align*}
\tag{11}
\]

The corresponding solution value of (1) is given by \( z_1 = z_3 + k(b - 1) \). Due to the two constraints, this problem (11) is considerably harder to solve than an ordinary
Subset-sum Problem, but a 2-optimal heuristic has shown to be very efficient for large-sized instances.

The 2-optimal heuristic restricts the solution space to those solution vectors which may be obtained from the break solution \( x' \) by interchanging any item present in the break solution with an item not present in the break solution. But although we should expect that this will demand considering \( O(n^2) \) pairs of items, the following propositions show that less than \( n \) pairs need to be considered when the items are ordered according to nondecreasing weights (2).

**Theorem 4.** For each item \( i \geq b \) there is one item \( h < b \) which yields the largest objective value in (11) when items \( i \) and \( h \) are interchanged. If the residual capacity is \( r \) then for each item \( i \) the optimal item is \( h = \min \{ j : w_j \geq w_i - r \} \).

**Proof.** By interchanging item \( h \) with item \( i \) the largest objective value in (11) is obtained for the smallest \( w_h \) such that \( r + w_h - w_i > 0 \). \( \Box \)

**Theorem 5.** An item \( i \geq b \) with \( w_i > w_{b-1} + r \) cannot be interchanged with any item \( h < b \) without violating feasibility of the solution. Similarly, an item \( h < b \) with \( w_h < w_b - r \) cannot be interchanged with any item \( i \geq b \) without violating feasibility.

**Proof.** Immediate from (2). \( \Box \)

We may assume that \( v' \neq c \) as otherwise the break solution is optimal. With the items ordered according to (2) the 2-optimal algorithm may be sketched as:

0 **Algorithm** twogoal
1 \( m \leftarrow \max \{ j : w_j \leq w_b - 1 + r \} \); \( h \leftarrow \min \{ j : w_j \geq w_b - r \} \); \( z_3 \leftarrow v' \);
2 for \( i \leftarrow b \) to \( m \) do
3 \hspace{1em} while \( w_h < w_i - r \) do \( h \leftarrow h + 1 \);
4 \hspace{1em} if \( v' - w_h + w_i > z_3 \) then \( z_3 \leftarrow v' - w_h + w_i \);
5 \hspace{1em} if \( z_3 = c \) then optimal solution found, stop.

An example of the 2-optimal heuristic is found in Fig. 1. The algorithm runs in \( O(n) \) time, thus the main computational effort is to order the items according to nondecreasing weights. But if we perform the ordering “by need” as described in [7] then several large-sized instances may be solved without ordering more than a couple of items, meaning that these instances are solved in linear time.

**Theorem 6.** The probability for finding a 2-optimal solution \( x \) which satisfies \( \sum_{j=1}^{n} w_j x_j = c \) and \( \sum_{j=1}^{n} x_j = b - 1 \) is

\[
P = 1 - \frac{1}{w_b - 1} \sum_{r=1}^{w_b-1} \left( 1 - \frac{\min\{r,R - w_b\} + 1}{w_b(R - w_b + 1)} \right)^{(b-1)(n-b+1)}
\]

if we assume that the weights are randomly distributed integers in \([1,R]\).
Fig. 1. An example of the 2-optimal heuristic. Initially we find $m = \max\{j : w_j \leq 11\} = 7$ and $h = \min\{j : w_j \geq 4\} = 2$. Now, for $i = 5$ we use $h = 2$ getting the objective value $z_5 = 23$. For $i = 6$ we use $h = 4$ getting the same objective, and finally for $i = 7$ we use $h = 4$ obtaining the optimal value $z_7 = 24 = C$.

Fig. 2. Pairs $h,i$ where optimal solutions are found on the line $w_i = w_h + r$.

**Proof.** It is obvious that a 2-optimal solution satisfies $\sum_{j=1}^{n} x_j = b - 1$ as we maintain this invariant throughout the algorithm. The 2-optimal solution is optimal if we can find two indices $h < b$ and $i > b$ such that $w_i - w_h = r$. All weights $w_j$ are randomly distributed in $[1, R]$, thus the situation looks like in Fig. 2.

We assume that the residual capacity $r = c - v'$ is randomly distributed in $[1, w_h - 1]$. There are totally $(b - 1)(n - b + 1)$ pairs of weights, which fall within a rectangle of size $w_h(R - w_h + 1)$, as $1 \leq w_h \leq w_h$ and $w_h \leq w_i \leq R$ due to the ordering of the weights. For a given value of $r$, the probability that one pair is optimal may be expressed as $P'_r = (\min\{r, R - w_h\} + 1)/(w_h(R - w_h + 1))$ since $\min\{r + 1, R - w_h + 1\}$ points fall on the line $w_i = w_h + r$ out of $w_h(R - w_h + 1)$ points.

The probability that a pair is not optimal is $1 - P'_r$, thus the probability that none of the $(b - 1)(n - b + 1)$ pairs are optimal may be found as $Q'_r = (1 - P'_r)^{(b - 1)(n - b + 1)}$, and the probability that we find at least one optimal pair is $P_r = 1 - Q'_r$. The average probability for all different values of $r \in [1, w_h - 1]$ may be found as $P = (1/w_h - 1) \sum_{r=1}^{w_h - 1} P_r$, which by reduction gives the stated. \[\square\]

Some probabilities (12) are given in Table 1 as average values of 1000 randomly generated instances. Nearly all large-sized instances will be solved by the 2-optimal heuristic.

4. Dynamic programming

In those cases where optimality of the heuristic solution cannot be proven we solve problem SCKP to optimality by considering each possible value of $\sum_{j=1}^{n} x_j = \beta$ for descending values of $\beta < b$. This leads to the following two-constrained Subset-sum
Table 1
Probability for finding a 2-optimal solution in per cent, incl. cases where \( r = 0 \)

<table>
<thead>
<tr>
<th>( n \backslash R )</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>70.5</td>
<td>14.3</td>
<td>1.6</td>
</tr>
<tr>
<td>100</td>
<td>98.4</td>
<td>69.1</td>
<td>15.0</td>
</tr>
<tr>
<td>300</td>
<td>100.0</td>
<td>96.3</td>
<td>65.6</td>
</tr>
<tr>
<td>1000</td>
<td>100.0</td>
<td>99.9</td>
<td>96.3</td>
</tr>
<tr>
<td>3000</td>
<td>100.0</td>
<td>100.0</td>
<td>99.7</td>
</tr>
<tr>
<td>10000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>30000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>100000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Problem

\[
\text{maximize } z_4 = \sum_{j=1}^{n} w_j x_j \\
\text{subject to } \sum_{j=1}^{n} w_j x_j \leq c, \\
\sum_{j=1}^{n} x_j = \beta, \\
x_j \in \{0,1\}, \quad j = 1, \ldots, n.
\] (13)

The corresponding objective value of (1) is given by \( z_1 = z_4 + k\beta \). Assuming that the items are ordered according to nondecreasing weights as given in (2), then the beta solution \( \hat{x} \) is the solution where the \( \beta \) lightest items are chosen. The weight of the beta solution is given by \( v = \sum_{j=1}^{\beta} w_j \), and to ensure feasibility we must have \( v \leq c \).

**Theorem 7.** An optimal solution to (13) may be obtained from the beta solution \( \hat{x} \) by repeatedly interchanging two items \((h, i)\) where \( h < \beta \) and \( i > \beta \).

**Proof.** Assume that the optimal solution is given by \( x^* \). Let \( h_1, \ldots, h_x \) be the indices \( h_i \leq \beta \) where \( x_{h_i}^* = 0 \), and \( i_1, \ldots, i_x \) be the indices \( i_j > \beta \) where \( x_{i_j}^* = 1 \). Obviously we have the same amount of indices \( \{h_j\} \) and \( \{i_j\} \) as we must maintain \( \sum_{j=1}^{n} x_j = \beta \). Order the indices such that \( h_x < \cdots < h_1 \leq \beta < i_1 \cdots < i_x \). Starting from the beta solution \( x = \hat{x} \) we repeatedly interchange items \( h_j \) and \( i_j \) for \( j = 1, \ldots, x \) obtaining \( x^* \). \( \Box \)

**Corollary 8.** The weight sum corresponding to the solution vector \( x \) will be nondecreasing for each interchange of items \( h_j \) and \( i_j \) as \( w_{h_j} \leq w_{i_j} \).

**Corollary 9.** Assume that all interchanges are made such that we first insert item \( i_j \) and then remove item \( h_j \). Then any solution \( x \) considered in the process of transforming \( \hat{x} \) to \( x^* \) satisfies the following bound on the corresponding weight sum:
\[ v \leq \sum_{j=1}^{n} w_j x_j \leq c + w_{p'}, \] where the first inequality follows from Corollary 8 and the second inequality follows from the fact that if we insert an item \( i > \beta \) getting a solution \( x \) with \( \sum_{j=1}^{n} w_j x_j > c + w_{p'} \), then \( x \) cannot become feasible by removing an item \( h \leq \beta \) as \( w_h \leq w_{p'} \).

**Theorem 10.** If an item \( i > \beta \) has \( w_i > w_{p'} + c - v \) then no solution with \( x_i = 1 \) is feasible.

**Proof.** As we maintain the same amount of items in all solutions, setting \( x_i = 1 \) means that another \( x_h = 0 \). The heaviest item \( h \leq \beta \) which can be removed in order to obtain feasibility is \( w_{p'} \), but when interchanging the two items in the beta solution we get the weight sum \( v - w_{p'} + w_i > c \). Thus \( x \) is not a feasible solution, and feasibility cannot be obtained by further interchanges due to Corollary 8.

**Theorem 11.** Assume that an item \( i > \beta \) was inserted obtaining a solution \( x \) with \( \sum_{j=1}^{n} x_j = \beta + 1 \) and weight sum \( W = \sum_{j=1}^{n} w_j \). Then interchanging \( i \) with an item \( h < \max \{ j : w_j < W - c \} \) will not lead to a feasible solution.

**Proof.** Removing an item \( h \) with \( w_h < W - c \) leads to the weight sum \( W - w_h > c \).

The previous observations may be combined to obtain an efficient dynamic programming algorithm as follows. Let \( f_{s,t}(\tilde{c}) \) for \( s \leq \beta, \ t > \beta, \ v \leq \tilde{c} \leq c + w_{p'} \) be an optimal solution to the subproblem of (13), which is defined on the variables \( i = s, \ldots, t \) of the problem:

\[
f_{s,t}(\tilde{c}) = \max \left\{ \sum_{j=1}^{s-1} w_j + \sum_{j=s}^{t} w_j x_j + \sum_{j=s-1}^{t-1} w_j x_j + \sum_{j=s}^{t} w_j x_j \leq \tilde{c}, \right. \\
\left. (s-1) + \sum_{j=s}^{t} x_j = \beta', \ x_j \in \{0,1\} \text{ for } j = s, \ldots, t \right\}, \tag{14}
\]

where \( \beta' = \beta \) for \( \tilde{c} \leq c \) and \( \beta' = \beta + 1 \) for \( \tilde{c} > c \). We will only consider those states \((s,t,\mu)\) where \( \mu = f_{s,t}(\mu) \), i.e. those weight sums \( \mu \) which can be obtained by interchanging items among \( s, \ldots, t \). The following dominance relation is used:

**Theorem 12.** Given two states \((s,t,\mu)\) and \((s',t',\mu')\). If \( \mu = \mu', \ s \geq s' \) and \( t \leq t' \), then state \((s,t,\mu)\) dominates state \((s',t',\mu')\) and we may fathom the latter.

**Proof.** Assume that an optimal solution \( x' \) is found from state \((s',t',\mu')\). We will show that a solution \( x^* \) with same objective value may be found from state \((s,t,\mu)\). If the solution vector corresponding to \((s,t,\mu)\) is \( x \), then we set \( x_j^* = x_j \) for \( j = s, \ldots, t \), while \( x_j^* = x_j' \) for \( j = 1, \ldots, s' - 1 \) and for \( j = t' + 1, \ldots, n \). Finally we set \( x_j^* = 1 \) for \( j = s', \ldots, s - 1 \), and \( x_j^* = 0 \) for \( j = t + 1, \ldots, t' \). In this way we have constructed a solution vector which corresponds to \( x \) at \( j = s, \ldots, t \) and which has the same objective values as that of \( x' \).

Using the dominance rule, we will enumerate the states for \( t \) running from \( b - 1 \) to \( n \). Thus at each stage \( t \) and for each value of \( \mu \) we will have only one index \( s \),
Fig. 3: The items and tables $s_\beta(\mu), \sigma(\mu)$ for a given instance.

which actually is the largest $s$ such that a feasible solution with weight sum $\mu$ can be obtained at the variables $x_1, \ldots, x_t$. Therefore let $s_t(\mu)$ for $t = \beta, \ldots, n$ and $v \leq \mu \leq c$ be defined as

\[
\begin{align*}
\text{there exists a solution } x \text{ which satisfies} \\
\sum_{j=1}^{s-1} w_j + \sum_{j=s}^{t} w_j x_j = \mu, \\
(s-1) + \sum_{j=s}^{t} x_j = \beta, \\
\end{align*}
\]

where we set $s_t(\mu) = 0$ if no feasible solution exists for the restricted problem.

At each iteration we either interchange item $t$ with an item $h \leq \beta$ or we omit item $t$. Thus after each iteration of $t$ all states considered will satisfy $\sum_{j=1}^{n} x_j = \beta$ and $\sum_{j=1}^{n} w_j x_j \leq c$. But to improve the complexity we will use a table $\sigma(\mu)$, for $\mu = v + w_{\beta+1}, \ldots, c + w_{\beta}$ to memorize which items previously have been removed for the corresponding weight $\mu$, such that we do not have to repeat the same operations. This leads to the following algorithm, which is exemplified in Fig. 3:

0 Algorithm \text{betadyn}

1. $m \leftarrow \max\{j : w_j \leq w_{\beta} + c - v\}$;
2. for $\mu \leftarrow v + 1$ to $c$ do $s_\beta(\mu) \leftarrow 0$;
3. $s_\beta(v) \leftarrow \beta + 1$;
4. for $\mu \leftarrow v + w_{\beta+1}$ to $c$ do $\sigma(\mu) \leftarrow 1$;
5. for $\mu \leftarrow c + 1$ to $c + w_{\beta}$ do $\sigma(\mu) \leftarrow \max\{j : w_j < \mu - c\} + 1$;
6. for $t \leftarrow \beta + 1$ to $m$ do
7. for $\mu \leftarrow v$ to $c$ do $s_t(\mu) \leftarrow s_{t-1}(\mu)$;
8. for $\mu \leftarrow v$ to $c + w_{\beta} - w_t$ do
9. $\mu' \leftarrow \mu + w_t$;
10. if $s_{t-1}(\mu) > \sigma(\mu')$ then
11. for $h \leftarrow \sigma(\mu')$ to $s_{t-1}(\mu) - 1$ do $\mu'' \leftarrow \mu' - w_h$; $s_t(\mu'') \leftarrow \max\{s_t(\mu''), h\}$;
12. $\sigma(\mu') \leftarrow s_{t-1}(\mu)$;
Algorithm \textit{betadyn} does the following: In line 1 we derive $m$ according to Proposition 8, and thus do not have to consider items $t > m$ in lines 6–12. In lines 2–3 we initialize $s_t(\mu)$ for $t = \beta$. At this stage only the beta solution satisfies $\sum_{j=1}^n x_j = \beta$, thus the table is initialized according to this fact. In lines 4–5 we initialize the $\sigma(\mu)$ table. As we have not tried to remove any items yet, we set $\sigma(\mu) = 1$ to indicate that we have not removed any items before item $h = 1$. For $\mu > c$ we may improve the initialization slightly, since, due to Proposition 9, we will never have to consider the removal of items $h$ with $w_h < \mu - c$, thus such items are marked in table $\sigma(\mu)$ as if they already have been considered.

Now items $t$ are considered for $t = \beta + 1$ to $m$ in line 6, alternately inserting item $t$ or omitting item $t$. Line 7 corresponds to the case where $t$ is omitted meaning that table $s_t(\mu)$ is copied without changes. In line 8–12 we insert item $t$ and update $s_t(\mu)$ accordingly. In order to maintain the equality $\sum_{j=1}^n x_j = \beta$, we have to remove an item $h \leq \beta$ in line 11. Thus the new weight sum $\mu'' = \mu + w_t - w_h$ is derived, and we update $s_t(\mu'')$ according to (15). However, we do not have to consider all items $h < s_t(\mu)$ when removing weights $w_h$, since for a given weight sum $\mu'$ subtracting an item $h$ will identify a unique weight sum $\mu'' = \mu' - w_h$. Repeating such an operation more than once will not lead to an increase in $s_t(\mu'')$. Thus table $\sigma(\mu)$ is used for this memorization in lines 10–12.

The optimal objective value is found as $z_4 = \max\{\mu: s_m(\mu) \neq 0\}$, while the corresponding solution vector is derived by backtracking through the states that lead to the optimal objective value.

**Theorem 13.** Algorithm \textit{betadyn} finds the optimal solution $x^*$ with $\sum_{j=1}^n x_j^* = \beta$.

**Proof.** According to Proposition 3 we need to show that all interchanges of pairs $(s, t)$ are considered. But the algorithm is constructed such that after each insertion of an item $t$ we remove all possible items $s$ before the last removed item $s_t(\mu)$.

The only restriction to the interchanges of pairs $(s, t)$ is in line 10 where we memorize, for each value of $\mu$, which items previously have been removed. But if $\sigma(\mu') = d > 1$ then we know that $s_t(\mu'') \geq h$ for $\mu'' = \mu' - w_h, h = 1, \ldots, \sigma(\mu')$, and thus we do not have to consider the removal of these items again. \(\square\)

**Theorem 14.** The complexity of Algorithm \textit{betadyn} is $O(nr)$ in time and space, where $r = c - v + 1$.

**Proof.** Space complexity: Table $s_t(\mu)$ is defined for $v \leq \mu \leq c$ thus it holds $r$ elements for each value of $t = \beta, \ldots, m$. Table $\sigma(\mu)$ is defined for $v + w_{\beta+1} \leq \mu \leq c + w_\beta$, i.e., $c - v + 1 + (w_\beta - w_{\beta+1}) \leq r$ elements.

Time complexity: The initialization of $s_t(\mu)$ in lines 2–3 takes $r$ operations, while initializing table $\sigma(\mu)$ in lines 4–5 takes $\beta + r$ operations as the different values of $\max\{j: w_j < \mu - c\} + 1$ may be derived in $\beta$ steps totally. The inner part of line 7 is executed at most $nr$ times, while lines 8–10 and 12 for each value of $t$ are executed
Finally, due to the memorization, the inner part of line 11 will remove at most \( \beta \) items for each value of \( \mu' = v + w_{p+1}, \ldots, c + w_\beta \), thus this part demands at most \( nr \) steps.

5. Main algorithm

As several large-sized strongly correlated problems may be solved faster than a complete sorting of the items, it is worthwhile to avoid as much of the sorting as possible [2]. Thus we apply the partitioning algorithm presented in [7] to find the break item \( b \), deriving the bound (10) in linear time.

The 2-optimal heuristic is then applied, sorting the items as they are considered. A 2-optimal solution \( x \) is an optimal solution to SCKP if \( \sum_{j=1}^n w_j x_j = c \). If optimality cannot be proved, we solve problem (13) for decreasing values of \( \beta \). The following bound is used for terminating the process:

**Theorem 15.** Let \( V = \sum_{j=n-\beta+1}^{n} w_j \) be the weight sum of the \( \beta \) largest items. Then an upper bound for SCKP with the additional constraint \( \sum_{j=1}^{n} x_j \leq \beta \) is \( u = \min \{c, V\} + k\beta \).

**Proof.** Assume that \( x^* \) is an optimal solution to the problem. Then obviously \( \sum_{j=1}^n w_j x_j^* \leq c \) and \( \sum_{j=1}^n w_j x_j^* \leq V \), thus \( z_1 = \sum_{j=1}^{\beta} (w_j + k) x_j^* = \sum_{j=1}^{\beta} w_j x_j^* + k \sum_{j=1}^{\beta} x_j^* \leq \min \{c, V\} + k\beta \).

**Theorem 16.** For a given \( \beta \) consider problem (13). If \( w_{\beta+1} - w_{\beta} \geq c - v \) then the only feasible solution to (13) is the beta solution.

**Proof.** In order to obtain a different solution from the beta solution, one must interchange at least two items \( h \leq \beta \) and \( i > \beta \). But then the weight sum is \( v - w_h + w_i \geq v - w_\beta + w_{\beta+1} > c \).

This leads to the following main algorithm:

**Algorithm scknap**

1. Find the break item \( b \) in linear time.
2. Find 2-optimal solution \( z \), sorting items by need. if \( z = c + k(b-1) \) then stop.
3. Sort remaining items.
4. for \( \beta \leftarrow b-1 \) downto 1 do
5. \( v \leftarrow \sum_{j=1}^{\beta} w_j; \ V \leftarrow \sum_{j=n-\beta+1}^{n} w_j; \ u \leftarrow \min \{c, V\} + k\beta; \)
6. if \( u \leq z \) then stop.
7. if \( w_{\beta+1} - w_{\beta} \leq c - v \) then solve problem (13) obtaining \( z_4 \). \( z \leftarrow \max \{z, z_4 + k\beta\} \).

For large-sized instances, the algorithm typically runs in linear time, as it is easy to find a 2-optimal solution without sorting more than a couple of items. For small-sized problems, the observed computational effort however becomes \( O(n^2c) \), as the inner
loop is performed at most \( n \) times, each demanding \( O(n(c - \nu + 1)) \) operations. If only one iteration of \( \beta - b - 1 \) is necessary to prove optimality, then the complexity of algorithm \( \text{betaDyn} \) is \( O(nw_\beta) \), and thus \( \text{betaDyn} \) runs in linear time when the weights are bounded by a constant \( R \).

We may derive an upper bound on the number of iterations in line 4. Choosing the trivial lower bound \( z = \sum_{j=1}^{b-1} w_j + k(b - 1) \) and upper bound \( u = c + k\beta \), we have that \( u > z \) if and only if \( c + k\beta > \sum_{j=1}^{b-1} w_j + k(b - 1) \) which means that \( (b - 1) - \beta < (c - \sum_{j=1}^{b-1} w_j)/k \leq R/k \). Thus the larger \( k \) is, the fewer iterations will be needed.

6. Computational experiments

The \textsc{knapsack} algorithm has been implemented in C and the following tests were performed on a HP9000/735. In each instance, the weights are randomly distributed in \([1, R]\), where the range \( R \) is tested with \( R = 100, 1000 \) and 10 000. The fixed-charge \( k \) has been tested with \( k = 10, 100, 1000 \) and 10 000. For each value of \( n \) and \( R \) we generate \( S = 1000 \) instances such that the capacity of instance \( i \) is given as \( c = (i/(S + 1)) \sum_{j=1}^{R} w_j \).

First Table 2 shows how many problems out of 1000 actually are solved by the 2-optimal heuristic. Due to formulation (11) the efficiency of the 2-optimal heuristic does not depend on the value of \( k \), thus we do not distinguish between the different classes of \( k \). All large-sized instances are solved by the heuristic, while the small sized problems are difficult to solve for large values of \( R \). Notice the conformity with the expected probabilities given in Table 1.

Table 3 shows the average solution times for the \textsc{knapsack} algorithm in milliseconds. All problems considered are solved within fractions of a second, and generally the strongly correlated problems are solved faster than uncorrelated \textsc{Knapsack} Problems [9] (i.e. problems where profits and weights are random distributed in \([1, R]\)). The experiments also showed that up to 5 different values of \( \beta \) had to be considered in (13) to solve a single problem to optimality. The number of dynamic programming

<table>
<thead>
<tr>
<th>( n \times R )</th>
<th>100</th>
<th>1000</th>
<th>10 000</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>70.0</td>
<td>16.4</td>
<td>2.0</td>
</tr>
<tr>
<td>100</td>
<td>96.6</td>
<td>68.9</td>
<td>13.9</td>
</tr>
<tr>
<td>300</td>
<td>99.9</td>
<td>95.8</td>
<td>64.7</td>
</tr>
<tr>
<td>1000</td>
<td>100.0</td>
<td>99.6</td>
<td>96.7</td>
</tr>
<tr>
<td>3000</td>
<td>100.0</td>
<td>100.0</td>
<td>99.6</td>
</tr>
<tr>
<td>10 000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>30 000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>100 000</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Table 3
Solution times in milliseconds, average of 1000 instances

<table>
<thead>
<tr>
<th>k = 10</th>
<th>k = 100</th>
<th>k = 1000</th>
<th>k = 10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>n\R</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
</tr>
<tr>
<td>30</td>
<td>0.1</td>
<td>0.4</td>
<td>8.1</td>
</tr>
<tr>
<td>100</td>
<td>0.0</td>
<td>0.1</td>
<td>3.3</td>
</tr>
<tr>
<td>300</td>
<td>0.1</td>
<td>0.2</td>
<td>1.5</td>
</tr>
<tr>
<td>1000</td>
<td>0.6</td>
<td>0.5</td>
<td>0.8</td>
</tr>
<tr>
<td>3000</td>
<td>1.2</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>10000</td>
<td>4.4</td>
<td>4.1</td>
<td>4.0</td>
</tr>
<tr>
<td>30000</td>
<td>12.4</td>
<td>12.4</td>
<td>12.6</td>
</tr>
<tr>
<td>100000</td>
<td>47.3</td>
<td>46.4</td>
<td>46.3</td>
</tr>
</tbody>
</table>

Finally Table 4 shows the solution times of scknap compared to several competing algorithms from the literature. The pr algorithm was presented by Pandit and Ravi-Kumar [5], while mth is due to Martello and Toth [4]. Finally the minknap algorithm is due to [8]. The solution times of pr and mth are taken from [4], where the tests were run on a VAXstation 3100. The minknap and scknap algorithm were run on a HP9000/735. Based on the solution times from [9] one may assume that the latter computer is about 10 times faster than the first, thus the solution times of pr and mth have been divided by 10. The instances are generated as described in [4]: Weights are randomly distributed in \([1, R]\), while the fixed-charge is chosen as \(k = R/10\), and the capacity is \(c = \frac{1}{2} \sum_{j=1}^{n} w_j\). For large sized instances the scknap algorithm is thousands of times faster than the competing algorithms. It should be emphasized that the considered instances are relatively easy for the competing algorithms since \(k\) is large, making it possible to derive tight upper bounds by surrogate or Lagrangian relaxation.
7. Inverse strongly correlated problems

All the previous results may be generalized to Inverse Strongly Correlated Knapsack Problems (ISCKP) which are defined as

\[
\begin{align*}
\text{maximize} & \quad z_1 = \sum_{j=1}^{n} (w_j - k)x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} w_j x_j \leq c, \\
& \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n.
\end{align*}
\]  

where \(k > 0\) is a positive constant, and the weights satisfy \(w_j > k\).

The main Propositions from Section 2 are shown as follows: Assume that the items are ordered according to nonincreasing weights, and let the break item \(b\) be defined by \(b = \min\{j : \sum_{j=1}^{j} w_i > c\}\). Assume that the initial solution \(z_1 = \sum_{j=1}^{b-1} (w_j - k)\), has been saved, such that we in the remaining section only are considering improved solutions.

Any better solution must consist of at least \(b\) items due to the ordering, thus we may impose the additional constraint \(\sum_{j=1}^{n} x_j \geq b\) to problem ISCKP in the search. By surrogate relaxation of this inequality to ISCKP using multipliers \(S_1 = 1, S_2 = -k\) we get \(\sum_{j=1}^{n} (w_j - k)x_j \leq c - kb\), meaning that an upper bound \(z_2\) to ISCKP may be derived by solving the following Subset-sum Problem (ISSP):

\[
\begin{align*}
\text{maximize} & \quad z_2 = \sum_{j=1}^{n} (w_j - k)x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} (w_j - k)x_j \leq c - kb, \\
& \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n.
\end{align*}
\]  

We have the obvious relation \(z_1 \leq z_2 \leq c - kb\), saying that \(u - c - kb\) is an upper bound for ISCKP. In a similar way as in Proposition 3 we may prove that an optimal solution \(x^*\) to ISSP is also an optimal solution to ISCKP provided that \(\sum_{j=1}^{n} x_j^* = b\).

Thus the 2-optimal heuristic and the betadyn algorithm may be applied to the Inverse Strongly Correlated Knapsacks Problems as presented earlier.

8. Conclusion

The present paper has uncovered special properties of Strongly Correlated Knapsack Problems that make these problems relatively easy to solve. The results are interesting in themselves, as we now are able to solve several fixed-charge problems in reasonable time, but due to the novel techniques, we may expect that the presented results also
may be applicable to algorithms for general Knapsack Problems. This has recently been demonstrated in Martello, Pisinger and Toth [3].

Fixing the number of items at $\sum_{j=1}^{n} x_j = \beta$ in (13) may be seen as a generalization of the core problem presented in Balas and Zemel [2] to Strongly Correlated Knapsack Problems. The search is in both cases focused on those solutions where we expect best objective values to be found. A similar approach may be used for other fixed-charge problems.

The betadyn dynamic programming algorithm may be used in several nonlinear Knapsack Problems, where the constraints depend on the number of items chosen. Among these should be mentioned the Collapsing Knapsack Problem and the Expanding Knapsack Problem considered in [6]. By fixing the number of items to $\sum_{j=1}^{n} x_j = \beta$ these problems become ordinary Knapsack Problems which can be solved by a modified version of betadyn for each value of $\beta$. Also Multiple-choice Knapsack Problems with a fixed number of items chosen in each class may be solved by betadyn.

Acknowledgements

The author wishes to thank the referees for helpful comments which improved the presentation of the paper.

References