Computers and Mathematics with Applications 62 (2011) 2913–2930

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

A two state reduction based dynamic programming algorithm for the bi-objective 0–1 knapsack problem

Aiying Rong [a,*], José Rui Figueira [b,1], Margarida Vaz Pato [c]

[a] Cemapre (Center of Applied Mathematics and Economics), ISEG -Technical University of Lisbon, Rua do Quelhas 6, 1200-781 Lisboa, Portugal

[b] INPL, Ecole des Mines de Nancy, Laboratoire LORIA, Parc de Saurupt- CS 14 234, 54 052 Nancy Cedex, France

[c] Centro de Investigação Operacional, FC - University of Lisbon, ISEG - Technical University of Lisbon, Rua do Quelhas 6, 1200-781 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 14 March 2011
Received in revised form 26 July 2011
Accepted 27 July 2011

Keywords:
Bi-objective knapsack instances
Multi-objective optimization
Dynamic programming
State reduction

ABSTRACT

In this paper, we present a dynamic programming (DP) algorithm for the multi-objective 0–1 knapsack problem (MKP) by combining two state reduction techniques. One generates a backward reduced-state DP space (BRDS) by discarding some states systematically and the other reduces further the number of states to be calculated in the BRDS using a property governing the objective relations between states. We derive the condition under which the BRDS is effective to the MKP based on the analysis of solution time and memory requirements. To the authors' knowledge, the BRDS is applied for the first time for developing a DP algorithm. The numerical results obtained with different types of bi-objective instances show that the algorithm can find the Pareto frontier faster than the benchmark algorithm for the large size instances, for three of the four types of instances conducted in the computational experiments. The larger the size of the problem, the larger improvement over the benchmark algorithm. Also, the algorithm is more efficient for the harder types of bi-objective instances as compared with the benchmark algorithm.

1. Introduction

The 0–1 knapsack problem (KP) is one of the most intensively studied NP-hard combinatorial optimization problems [1]. The multi-objective 0–1 KP (MKP) is a generalization and a natural extension of the single objective 0–1 KP by considering two or more objectives. The MKPs are frequently encountered in practice since multiple conflicting objectives are more appropriate to model real-world situations. Examples can be found in capital budgeting [2], selection of transportation investment alternatives [3], relocation issues arising in nature conservation, biology [4], selection of building renovation methods [5], environmental investments [6], and facility location [7].

The MKP is described below. Given n items and r profit objectives for each item, with the kth profit objective $c_j^k (k = 1, \ldots, r)$ and weight w_j for item j $(j = 1, \ldots, n)$ and a knapsack of capacity W, the problem is to select a subset of items whose total weight does not exceed W and whose total profit objectives are maximized in the Pareto sense. The MKP can be formulated as the following multi-objective integer linear programming model:

$$\text{“max”} \left(\sum_{j=1}^{n} c_j^1 x_j, \ \ldots, \ \sum_{j=1}^{n} c_j^r x_j \right) \tag{1}$$

* Corresponding author. Tel.: +351 213922747; fax: +351 213922782.
E-mail addresses: arong@iseg.utl.pt (A. Rong), Jose.Figueira@mines.inpl-nancy.fr (J.R. Figueira), mpato@iseg.utl.pt (M.V. Pato).
1 Associate member at CEG-IST, Lisboa, Portugal.

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.07.067

subject to

$$\sum_{j=1}^{n} w_j x_j \le W, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{3}$$

where x_j are decision variables indicating whether the jth item is selected to place in the knapsack or not. Here, we follow the common assumptions in most literature: W, w_j, $c_j^k (j = 1, \ldots, n; k = 1, \ldots, r)$ are positive integers. To avoid trivial solutions, it is assumed that $w_j \le W (j = 1, \ldots, n)$ and $\sum_{j=1}^{n} w_j > W$.

For the single objective 0–1 KP, decades of algorithmic improvements have made it possible to solve in a reasonable time limit nearly all standard instances from the literature. But some types of instances are still hard to solve because of its NP-hard nature. Ref. [8] pointed out that the strongly correlated instances (weight coefficients and profit coefficients are strongly correlated) and some other types of instances challenged the existing algorithms. Ref. [9] compared the solution times of all recent algorithms using classical and new benchmark test instances. The new benchmark set includes instances with large ($\ge 10^5$) or moderate (10^3) weight coefficients and all the algorithms based on currently used upper bound techniques showed bad performance on these instances. This study pointed out that dynamic programming (DP) is one of the best approaches for solving the hard types of the 0–1 KP.

In the multi-objective optimization context, the solution process consists of finding the Pareto frontier (PF) with a number of non-dominated objective vectors in the objective space which corresponds to efficient solutions in the decision space. Hence, compared with the single objective KP, the MKP poses more challenges. On the one hand, there are intractable instances of multi-objective combinatorial optimization problems, for which the number of efficient solutions is not polynomial in the size of their instances [10]. On the other hand, for most multi-objective combinatorial optimization problems, deciding whether a given objective vector is dominated or not is an NP-hard problem [11], even if the underlying single objective version can be solved in polynomial time.

However, these difficulties do not prevent the research effort from developing efficient algorithms able to find the PF quickly from the practical viewpoints. In the following, we review the main accurate approaches for solving the MKP. Ref. [12] presented the theoretical DP framework for the multi-objective integer KP. Ref. [13] implemented an exact algorithm for the bi-objective 0–1 KP (BKP) by exploring developments for the multi-objective linear programming problem. The above two research contributions can be considered as theoretical developments because the authors did not present extensive experimental results. Other researchers have developed specific algorithms based on extensive numerical tests. Most of this research work focused on calculating the PF with the exception of [14]. Ref. [14] presented a generic labeling algorithm, which calculated both the PF in the objective space and the corresponding efficient solutions in the decision space, for the multi-objective integer KP. Ref. [15] presented a two-phase branch and bound algorithm for the BKP. Ref. [16] presented a labeling algorithm for the BKP by transforming a KP into a shortest path problem. It is in essence a DP algorithm. Ref. [17] studied a dominance based on the DP (DDP) algorithm for the MKP and numerical tests were conducted for the BKP and tri-objective KP. Ref. [18] applied bound sets in the DP algorithm for the MKP and numerical tests were conducted for the BKP. Both [17,18] are hybrid DP algorithms.

Similar to the single-objective 0–1 KP, DP algorithms [16–18] are among the best approaches to solve the MKP. Ref. [16] attempted to reduce the number of states to be calculated by generating a forward reduced-state DP space (FRDS) but the computational effort for the final stage is very heavy. The DP space consists of all of the states and related transitions between states in the DP process. Ref. [17] relied on several dominance relations to discard partial solutions that cannot lead to new non-dominated objective vectors and Ref. [18] applied elaborate bounding techniques to reduce the number of states to handle in the DP process. This can significantly reduce the computational effort for the algorithm. However, applying dominance relations and bounding techniques still needs a heavy computational effort.

In this paper, we focus our attention on finding the PF for the MKP, i.e., the algorithm is designed to address the multi-objective case. But our implementation and numerical tests focus on the bi-objective case. We follow DP approaches for dealing with the MKP and use a backward reduced-state DP space (BRDS) to avoid heavy computational effort for the FRDS in the final stage. The major contributions of the paper are summarized as follows. First, we identify a BRDS by exploring the network of the basic sequential DP (BDP) process. Second, we derive the condition under which the BRDS is effective to the MKP based on the analysis of its impact on the solution time and memory requirements. Finally, we develop a new DP algorithm by applying the BRDS in conjunction with a property governing the objective relations between states, which can help to reduce further the number of states to be calculated in the BRDS. To our knowledge, it is the first time that the BRDS is used in the DP algorithm.

Some states, which have been discarded in our approach, may coincide with those discarded by the dominance relations in [17] or by bounding techniques [18]. However, the techniques used for discarding the states are completely different. Moreover, our techniques are especially efficient for the hard types of KP instances as compared with [17,18]. The hard types of instances include conflicting instances where the profit objectives are negatively correlated. The hardest instances are those where conflicting objectives are positively correlated with the weight coefficients. Usually the cardinality of the PF for these instances increases rapidly as the problem size increases. Very few techniques can handle these instances efficiently.

The paper is broken down as follows. In Section 2, we give the network representation of the multi-objective BDP process to provide the foundation for generating the BRDS. In Section 3, we outline the main components of a two state reduction

based DP (TDP) algorithm. We present a procedure for obtaining the BRDS and a procedure for generating the pseudo critical states, between which the profit values are the same, in the DP space, used to reduce the number of states to be calculated. Then we present a TDP algorithm by combining these components. In Section 4, computational results are reported for the BKP instances. A comparison with the DDP algorithm [17] is presented. To the authors' knowledge, the DDP algorithm is the best algorithm for handling the hard type of instances efficiently. For the non-hard type of instances, the algorithm presented in [18] is more efficient than the DDP algorithm.

2. Network representation of basic sequential DP process

Dynamic programming (DP) is a general optimization methodology developed by Bellman [19]. It can be considered as a recursive process, which interprets an optimization problem as a multi-stage decision process. Each stage consists of many states. A state is a way to describe a solution for the sub-problem and contains enough information to determine the optimal solution of the future state. In Bellman's optimization principle, a recursive equation is set up to describe the optimal solution at a given state in terms of the optimal solution of the previously considered states. There are different representations of the states within a DP framework (see [12]). Below we describe one of these ways for solving the 0–1 KP, called basic sequential DP (BDP) procedure, corresponding to mode III in [12].

The BDP process includes n stages, namely $\alpha = 1, \ldots, n$. Each stage α corresponds to one variable whose value is determined and consists of $(W + 1)$ states. It means that each stage is associated with a sub-problem and the sub-problems are sequentially solved stage by stage. Here we represent the underlying recursive equations of the BDP algorithm based on network optimization terminology.

2.1. Network presentation

The network for the BDP process is a direct connected network. A state in the DP procedure is represented by a node and the transition from state to state by a directed arc in the network. We use the same layer technique as [16] to represent the stages, i.e., a stage in the DP process is associated with a layer in the network.

Let $G = (N, A, p)$ be a direct connected network, where N is the set of nodes and $A \subseteq N \times N$ is the set of arcs. The arc from node i to node j is denoted by (i, j) and the values associated with the arc (i, j) are represented by an r dimensional vector $p(i, j) = (p^1(i, j), \ldots, p^r(i, j))$. There are n layers in the network, namely, $\alpha = 1, \ldots, n$, and each layer consists of W+1 nodes, counting from 0 to W. Let α^t, corresponding to state t at stage α, denote the node at the tth position in layer $\alpha(\alpha = 1, \ldots, n; t = 0, \ldots, W)$. In the BDP process, t represents the knapsack capacity of the sub-problem of node α^t. Consequently, the total weight of items for the solution at node α^t is not larger than t. For two non-negative integer $t1$ and $t2$ in the same layer, if $t1 > t2$, then the position of the node α^{t1} is higher than that of α^{t2}. The arc values concerning the r objectives from layers $\alpha - 1$ to α are given by $(c_\alpha^1, \ldots, c_\alpha^r)$ if item α is included in the knapsack in layer α or $(0, \ldots, 0)$, otherwise. Let $S(\alpha^t)$ denote the set of non-dominated objective vectors at node α^t. Then recursive equations of the BDP algorithm can be represented as follows.

For $\alpha = 1$, we get:

$$S(1^t) = \begin{cases} \{(0, \ldots, 0)\} & 0 \le t < w_1 \\ \{(c_1^1, \ldots, c_1^r)\} & w_1 \le t \le W. \end{cases} \tag{4}$$

and for $\alpha = 2, \ldots, n$, the recursive equations can be represented as follows:

$$S(\alpha^t) = \begin{cases} S((\alpha - 1)^t) & 0 \le t < w_a \\ \text{non-dominated } \left(S((\alpha - 1)^t) \cup \{(c_\alpha^1, \ldots, c_\alpha^r) \oplus S((\alpha - 1)^{t - w_\alpha})\}\right) & w_\alpha \le t \le W \end{cases} \tag{5}$$

where "non-dominated" in Eq. (5) refers to all the non-dominated objective vectors for the operating set and the "$\oplus$" operator means that the addition operation is applied over all the elements of the set $S((\alpha - 1)^{t - w_\alpha})$, i.e., $(c_\alpha^1, \ldots, c_\alpha^r)$ should be added to each element of the set. The parameters W, w_α,$c_\alpha^k(\alpha = 1, \ldots, n, k = 1, \ldots, r)$ are the same as those used in the problem formulation (1) and (2).

At each stage $\alpha = 1, \ldots, n$, a node at a higher position will obtain the non-dominated objective vectors of a node at a lower position if these vectors remain non-dominated at a higher position. Similarly, each node at stage $\alpha(\alpha = 2, \ldots, n)$ will obtain the non-dominated objective vectors of a related node at stage $\alpha - 1$ if these vectors remain non-dominated at stage α. These results are guaranteed by Eq. (5). Finally, $S(n^W)$ collects all the non-dominated objective vectors for the problem.

2.2. A numerical example

Next we use the following BKP example to illustrate the network underlying the BDP process and all the subsequent networks.

"max" $(7x_1 + 9x_2 + 3x_3 + 7x_4 + 6x_5, 2x_1 + 2x_2 + 10x_3 + 6x_4 + 9x_5)$

subject to: $3x_1 + 2x_2 + 2x_3 + 4x_4 + 3x_5 \le 9$,

$x_j \in \{0, 1\}, \quad j = 1, \ldots, 5.$

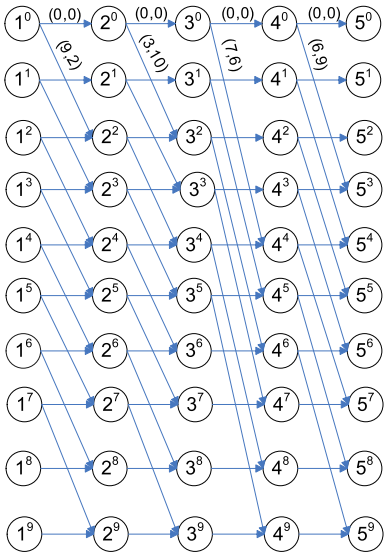

Fig. 1. Illustration for the network underlying the basic sequential DP process.

Fig. 1 shows the network underlying the BDP process. All the horizontal arcs from layers $\alpha - 1$ to $\alpha (\alpha = 2, \ldots, 5)$ have the same profit vector $(0, 0)$. All the remaining arcs from layers $\alpha - 1$ to α $(\alpha = 2, \ldots, 5)$ also have the same profit vector (c_α^1, c_α^2). Note that we only label the profit vectors of two arcs in each layer of the network.

3. Outline of the new DP algorithm

In our new DP algorithm, we improve the BDP algorithm by introducing two reductions. The first reduction is concerned with generating a backward reduced-state DP space (BRDS). The second reduction is concerned with generating pseudo sparse (critical) nodes used to further reduce the number of states to actually calculate the BRDS. The following notation is used in all the procedures related to the new algorithm:

A	set of arcs,
QN	set of necessary nodes,
QS	set of (pseudo) sparse nodes,
TN	set of node positions for necessary nodes,
TN_α	set of node positions for the necessary nodes in layer α based on decreasing order,
TS	set of node positions for (pseudo) sparse nodes,
TS_α	set of node positions for (pseudo) sparse nodes in layer α based on decreasing order,
V	set of auxiliary node positions,
b_α	the lowest position of the necessary nodes in layer α.

3.1. Generating a backward reduced-state DP space

As mentioned in Section 2, the BDP procedure in fact solves all-capacity KPs [1]. In the last layer n, all the non-dominated objective vectors of the sub-problem with knapsack capacity t $(t = 0, \ldots, W)$ can be obtained at the corresponding node n^t. Particularly, all the non-dominated objective vectors of the problem with knapsack capacity W are obtained at node n^W. As illustrated in Fig. 1, since there are no connections between the nodes in the same layer, the non-dominated objective vectors of the sub-problems with different knapsack capacities are independent of each other. Hence, in the last layer n, only node n^W is necessary for solving the problem with capacity W and all the remaining nodes are unnecessary. Using this observation as a starting point, we can generate a reduced-state DP space using a backtracking procedure.

Starting from node n^W in the last layer $\alpha = n$, the nodes in the previous layer $\alpha - 1$, which have direct connection with node n^W can be identified. There are two nodes connecting to node n^W, one is represented by $(n-1)^{W-w_n}$ and the other is by $(n-1)^W$. This process can be repeated for all the layers α $(\alpha = n-1, \ldots, 2)$ to identify all the connecting nodes. For a node $\alpha^t (\alpha = 2, \ldots, n)$, if $w_\alpha \le t \le W$, there are two nodes in the previous layer $\alpha - 1$ connected to it, one is $(\alpha-1)^t$ and the other is $(\alpha-1)^{t-w_\alpha}$; if $0 \le t < w_\alpha$, there is only one node $(\alpha-1)^t$ connected to it. We call these related nodes identified by the backtracking process necessary nodes. All the necessary nodes and the corresponding arcs form a backward

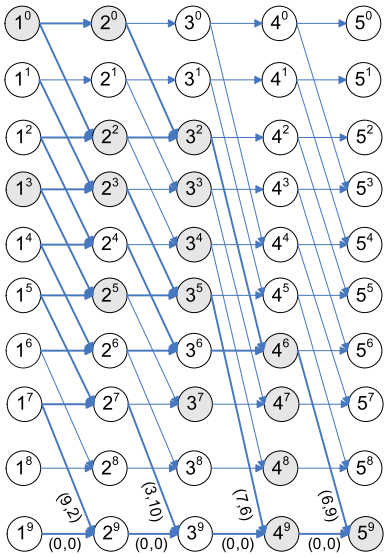

Fig. 2. Illustration for the necessary nodes and related arcs (thick) as well as (pseudo) sparse nodes in the DP process.

Algorithm 1

begin
  $QN \leftarrow \{n^W\}$, $TN \leftarrow \{W\}$, and $V \leftarrow \{\}$;
  $A \leftarrow \{\}$;
  for ($j = n$-1 to 1) do
   begin
    while ($TN \neq \{\}$) do
     begin
      Let a be the first element in TN and remove it from TN;
      if ($a \notin V$) then
      begin
        $V \leftarrow V \cup \{a\}$;
        $QN \leftarrow QN \cup \{j^a\}$;
      end
      $A \leftarrow A \cup \{(j^a, (j+1)^a)\}$;
      $p(j^a, (j+1)^a) \leftarrow (0,\ldots,0)$;
      if ($a - w_{j+1} \geq 0$) then
      begin
        if ($a - w_{j+1} \notin V$)
        begin
          $V \leftarrow V \cup \{a - w_{j+1}\}$
          $QN \leftarrow QN \cup \{j^{a-w_{j+1}}\}$;
        end
        $A \leftarrow A \cup \{(j^{a-w_{j+1}}, (j+1)^a)\}$;
        $p(j^{a-w_{j+1}}, (j+1)^a) \leftarrow (c^1_{j+1},\ldots,c^r_{j+1})$;
      end
    end
    $TN \leftarrow V$ and $V \leftarrow \{\}$;
  end
end

Fig. 3. Procedure for generating a backward reduced-state DP space.

reduced-state DP space (BRDS). Fig. 2 shows the necessary nodes and corresponding arcs (thick lines) as well as (pseudo) sparse nodes (see later description) in the BDP space using the example in Section 2. The procedure for generating the BRDS is given in Fig. 3.

Next, we analyze the impact of the BRDS on the single objective KP and MKP in terms of solution time efficiency and memory requirements.

The BRDS is generated following the network of the BDP process in a straightforward backtracking procedure. The observation is valid for both the single objective KP and MKP. However, to our best knowledge, there are no published

reports about how to generate the BRDS and use it to develop an algorithm for the single objective KP or the MKP. The comprehensive monograph such as [1] does not mention the BRDS either. The major reason for ignoring the BRDS in the literature is that it has little benefit in terms of solution time but it introduces additional memory requirements for the single objective case as compared with the BDP algorithm. The arguments are given below.

The BRDS must be generated beforehand. The number of nodes assumes a non-increasing pattern as shown in Fig. 2. The generation takes $O(nN_1)$ time and $O(nN_1)$ memory space (where N_1 is the number of necessary nodes in the first stage of the BRDS). Moreover, the algorithm based on it also takes $O(nN_1)$ time to obtain the optimal solution and $O(nN_1)$ memory space for recording the values of the decision variables. The BDP algorithm needs $O(nW)$ time and $O(nW)$ memory space [1]. In the worst case, $N_1 = W + 1$. Here it is worth mentioning that the computational effort for generating the BRDS within the DP algorithm should be larger than that for the DP algorithm based on it to compute the optimal solution because the necessary nodes should be implicitly ordered in each stage during the generating process (the operation is not explicitly shown in the procedure in Fig. 3).

Since the time for generating the BRDS is comparable to the time for solving the single objective 0–1 KP based on it or the BDP algorithm, there is not much benefit for the solution time even though the number of nodes is reduced. Moreover, since the memory space for storing the BRDS is the same as the space for recording the values of decision variables for the algorithm based on it, the BRDS has potential to increase the memory requirements as compared with the BDP algorithm. When $N_1 > W/2$, the memory requirements are larger than those for the BDP algorithm.

Similar to the single objective case, whether the BRDS is effective to the MKP or not should be determined by both the solution time efficiency and memory requirements. The BRDS can be effective against the MKP if its generation time is smaller than the time for calculating the PF and its memory requirements are smaller than those for storing the objective vectors during the DP process for the BRDS based algorithm.

The condition of the computational time can be satisfied easily in the MKP context. The time for calculating the PF is determined by the number of objective vectors calculated during the DP process. In the extreme case, if each node only contains one objective vector in each stage of the DP process, then the time for calculating the PF for the BRDS based algorithm is comparable to the time for generating the BRDS. Otherwise, it should be larger. The larger the number of objective vectors of a node, the larger the computational time for obtaining the PF for the BRDS based algorithm. It means that the computational time is not a barrier for applying the BRDS.

In the following, we focus on the memory requirement analysis of the BDP algorithm and the BRDS based algorithm and derive the conditions under which the BRDS is favorable for the MKP, i.e., the sum of the memory requirements for storing the objective vectors for the BRDS based algorithm and for storing the BRDS itself are smaller than those for the BDP algorithm and the memory requirements for storing the BRDS are smaller than those for storing the objective vectors.

For both the BDP and BRDS based algorithms, only one stage of objective vectors need to be kept if the calculation process proceeds from the node with the highest position to the node with the lowest position in each stage, as for the single objective case [1].

Let N_α denote the number of nodes, M_α the maximum number of non-dominated objective vectors of a node at stage α ($\alpha = 1, \ldots, n$) in the DP process, M the maximum number of non-dominated objective vectors of a node for the entire DP process and U the memory requirements for storing the objective vectors of a DP based algorithm (either BDP or BRDS-based).

For determining the memory requirements, we use the memory requirements for storing an integer as a basic measuring unit. Hence, the memory requirements for storing an r dimensional integer objective vector are r times the basic unit and for storing one stage of objective vectors are $r \max_\alpha(N_\alpha M_\alpha)$. Then the memory requirements for storing the objective vectors for the BDP or BRDS based algorithms should be

$$U = r \max_{\alpha}(N_\alpha M_\alpha). \tag{6}$$

For both the BDP and BRDS based algorithms, the PF is obtained at a single node at the final stage, thus, $M \geq ND$ (where ND is cardinality of the PF for a given knapsack instance). Based on (6), the memory requirements for the BDP algorithm (U^{BDP}) are given by

$$U^{\mathrm{BDP}} = rM(W + 1). \tag{7}$$

Since the node profile of the BRDS assumes a non-increasing pattern and M_α has tendency to increase with α, for the BRDS based algorithm, the memory requirements for node calculation are small at both the early and late stages due to the fact that the number of non-dominated objective vectors of a node is small at the early stages and the number of nodes is small at the late stages. The maximum memory requirement will occur somewhere around the middle stage θ where both the number of nodes and the number of non-dominated objective vectors are moderately large. Then $M_\theta = \rho_1 M, N_\theta = \rho_2(W + 1)$, and $N_1 = \rho_3(W + 1)$ (where $0 < \rho_1 < 1, 0 < \rho_2 < 1, 0 < \rho_3 \leq 1$ and $\rho_3 > \rho_2$). It is known that memory requirements for the BRDS based algorithm (U^{BRDS}) are the sum of the memory requirements for storing the objective vectors during the DP process and for storing the BRDS, i.e.,

$$U^{\mathrm{BRDS}} = \rho_2\rho_1 rM(W + 1) + \rho_3(W + 1)n. \tag{8}$$

Combining (7) and (8), if $M > \rho_3 n/((1-\rho_1\rho_2)r)$, then the memory requirements for the BRDS based algorithm are smaller than those for the BDP algorithm. At the same time, it is required that $\rho_3(W+1)n < \rho_2\rho_1 rM(W+1)$, i.e., $M > \rho_3 n/\rho_2\rho_1 r$.

Since $M \geq ND$ and based on numerical results of [17], ND is the function of n and r, the BRDS based algorithm will be effective for the MKP if

$$ND > \max\{\rho_3 n/((1-\rho_1\rho_2)r), \rho_3 n/(\rho_2\rho_1 r)\}. \tag{9}$$

The right-hand side of (9) is determined by the second term when $\rho_1\rho_2 < 0.5$, and the first term, otherwise. It means that the right-hand side of (9) is a linear function of n for a given r, i.e., $\lambda n/r$ (λ is a constant, $\lambda > 0$). This condition holds for some types of instances, especially for hard type of instances where ND increases fast ($ND \gg \lambda n/r$) as n increases for a given r and for types of instances where the number of efficient solutions increases exponentially with respect to n [20]. Based on [17], ND increases much faster with respect to n for the tri-objective case than for the bi-objective case for the same type of instances.

Now, we briefly discuss the time complexity of the BRDS based algorithm. As commented by Klamroth and Wiecek [12], the structure of the DP space, including the total number of nodes, the number of final nodes and the number of transition arcs affects the time complexity of the related DP algorithm. More importantly, the time complexity should be related to the number of objective vectors of a node, which is associated with the ND of an instance. The BRDS contributes to reducing all the above factors that affect the time complexity. The BRDS helps to reduce the final nodes of the BDP space from $(W+1)$ to 1, as well as the number of nodes with a large number of objective vectors in the late stages and related transition arcs. It means that the BRDS based algorithm can improve the solution time efficiency greatly as compared with the BDP algorithm.

3.2. Generating (pseudo) sparse nodes

Ref. [21] mentioned another type of reduced-state DP space to improve the efficiency of the BDP algorithm for the single objective 0–1 KP. Refs. [22,23] applied the same type of reduced-state DP space to solve the single objective integer KP. This type of reduced-state DP space was generated based on the state dominance relation by discarding the dominated states through a forward process. We call it a forward reduced-state DP space (FRDS). A dominated state (node) is characterized as follows: its position is not lower than that of the other node and its profit values are not better than that of the other node. The non-dominated nodes, called sparse (or critical) nodes, form the FRDS. The sparse nodes are nodes with distinct profit values in the DP space. A property governing sparse nodes states that the profit values of the nodes between two consecutive sparse nodes in each stage are the same. It means that the DP space can be fully characterized by sparse nodes. The sparse node DP (SDP) algorithms [21–23] have been developed based on the FRDS by calculating the profit values of the sparse nodes.

The labeling algorithm [16], an intermediate version of the SDP algorithm for the BKP also processes the nodes in the FRDS. It generates nodes directly based on the weight of items without state dominance checking and the number of nodes is also reduced as compared with the BDP algorithm. However, some additional nodes may be generated between two (true) sparse nodes. We call these nodes pseudo sparse nodes. A pseudo sparse node is contiguous to either another pseudo sparse node or a true sparse node. The set of objective vectors for the nodes between two consecutive (pseudo) sparse nodes is the same.

In our DP algorithm, we calculate the necessary nodes of the BRDS and attempt to reduce further the number of nodes to calculate by using the property of sparse nodes. It means that the sparse nodes should be known before the state calculation starts in each stage. However, this is not possible because sparse nodes can only be obtained during the solution process of the SDP algorithm with state dominance checking. Instead, in our DP algorithm we generate sparse nodes including pseudo ones based on the labeling algorithm [16] because pseudo sparse nodes maintain the property of sparse nodes as mentioned above and these nodes can be generated beforehand based on the weight of items.

The calculation principle of the necessary nodes is the same regardless of the fact that we use the true and pseudo sparse nodes. We calculate only one necessary node between two consecutive (pseudo) sparse nodes based on the property of (pseudo) sparse nodes. However, the calculated node is not necessarily a (pseudo) sparse node. Sometimes, it may be some intermediate node between two consecutive (pseudo) sparse nodes and its objective vectors are the same as one of these two (pseudo) sparse nodes. In this way, the number of nodes to actually calculate can be smaller than the necessary nodes considered in Section 3.1.

Next, we discuss some considerations for generating (pseudo) sparse nodes in our algorithm. First, it is not necessary to generate the (pseudo) sparse nodes whose position is lower than the lowest position of the necessary node in each stage. Second, it is neither necessary to generate the arcs connecting to (pseudo) sparse nodes nor necessary to generate the dummy source and sink nodes in the labeling algorithm [16]. The remaining steps are similar to the network converting algorithm in [16]. Fig. 4 gives a procedure for generating (pseudo) sparse nodes.

The (pseudo) sparse nodes are illustrated together with the necessary nodes in the DP space of Fig. 2 above. Here it is worth mentioning that the algorithm in Fig. 4 does some relaxation and simplifies the generation process. Some generated (pseudo) sparse nodes are not necessary nodes in the DP space, for example nodes 4^7 and 4^8 in Fig. 2, because (pseudo) sparse nodes are generated by a forward process while necessary nodes are generated by a backtracking process. But this

```
Algorithm 2
1:  begin
2:      QS ← {1^0, 1^{w_1}}, TS ← {0, w_1}, and V ← {};
3:      for (j = 2 to n) do
4:       begin
5:          Let b_j be the lowest position of the necessary node in layer j;
6:          while (TS ≠ {}) do
7:          begin
8:             Let a be the first element in TS and remove it from TS;
9:             if (a ≥ b_j) then
10:             begin
11:                if (a ∉ V) then
12:                begin
13:                   V ← V ∪ {a};
14:                   QS ← QS ∪ {j^a};
15:                end
16:             end
17:             if (a + w_j ≤ W) then
18:             begin
19:                if (a + w_j ≥ b_j) then
20:                begin
21:                   if (a + w_j ∉ V) then
22:                   begin
23:                      V ← V ∪ {a + w_j};
24:                      QS ← QS ∪ { j^{a+w_j} };
25:                   end
26:                end
27:             end
28:       end
29:       TS ← V and V ← {};
30:    end
31: end
```

Fig. 4. Procedure for generating (pseudo) sparse nodes.

does not affect the function of the (pseudo) sparse nodes. These nodes can be skipped automatically when we process the calculation of the necessary nodes in the BRDS (see Steps 14 and 27 of the DP algorithm in Fig. 5).

3.3. Two state reduction based DP (TDP) algorithm

By combining the procedure for generating the BRDS (Fig. 3) with the procedure for generating (pseudo) sparse nodes (Fig. 4), at the same time, using the property of (pseudo) sparse nodes to determine which nodes should be calculated in the BRDS, we obtain a two state reduction based DP (TDP) algorithm. We arrange both necessary and (pseudo) sparse nodes at each stage based on the decreasing order of node positions. This means that we search the BRDS from the node with the highest position to the node with the lowest position in each stage of the DP process. We present the TDP algorithm in Fig. 5.

Based on Eq. (5), a node with the position lower than w_α at stage $\alpha (\alpha = 2, \ldots, n)$ directly takes the value of the node with the same position at the previous stage $\alpha - 1$ (Step 30); no calculation is needed. It means that only the nodes with position not lower than w_α at stage α may need to be calculated (from Steps 19 to 28). The first necessary node at each stage has position W. It must be calculated (Steps 9 and 10). Then the first (pseudo) sparse node with the position not higher than that of this calculated node is found (Step 14). Then, the necessary nodes with the positions not lower than the position of this (pseudo) sparse node should have the same set of objective vectors as the node calculated on the basis of the property of the (pseudo) sparse nodes. Thus, we do not need to calculate these nodes (Steps 20 and 21). It means that the next necessary node to calculate is one whose position is lower than that of this (pseudo) sparse node (Step 23). Immediately after that, the new (pseudo) sparse node with the position not higher than the newly calculated node is found (Step 27).

The above procedure can be repeated till all the necessary nodes are checked, either by calculating, or by taking the objective vectors from a calculated node (Steps 6–32). If the calculated node is the last node (the lowest position node) in each stage, it is not necessary to find the new (pseudo) sparse node. Some (pseudo) sparse nodes generated by the procedure in Fig. 4, which are not necessary nodes, are automatically skipped during the process of finding the next (pseudo) sparse nodes with the position not higher than the newly calculated necessary nodes (Steps 14 and 27). Fig. 6 illustrates the nodes (gray) to actually calculate in the BRDS space for the TDP algorithm, using the example in Section 2.

It is worth mentioning that the DP procedure in Fig. 5 can change slightly to reduce memory requirements in implementation. The BRDS must be generated beforehand as shown in Fig. 5, because a backtracking process is used for generating the necessary nodes as shown in Fig. 3. The memory requirements for storing the BRDS are $N_1 n$, where N_1 is the number of necessary nodes in the first stage as mentioned in Section 3.1. In the worst case, $N_1 = W + 1$. However, not

Two state reduction based DP (TDP) algorithm

1: begin
2: Call Algorithm 1 for generating the backward reduced-state DP space (Figure 3)
3: Call Algorithm 2 for generating (pseudo) sparse nodes (Figure 4);
4: Assign the objective vectors of necessary nodes corresponding to the position in set TN_1 based on equation (4);
5: Let p, t, s be the node positions;
6: for ($\alpha = 2$ to n) do
7: begin
8: Let b_α be the lowest position in TN_α;
9: Let t be the first element in TN_α and remove it from TN_α;
10: $S(\alpha^t) \leftarrow$ non- dominated$\left(S((\alpha-1)^t) \cup \{(c_\alpha^1,...,c_\alpha^r) \oplus S((\alpha-1)^{t-w_\alpha})\}\right)$;
11: if ($|TN_\alpha| > 0$) then
12: begin
13: $p \leftarrow t$;
14: $s \leftarrow$ the first element in TS_α not larger than t;
15: end
16: while ($TN_\alpha \neq \{\}$) do
17: begin
18: Let t be the first element in TN_α and remove it from TN_α;
19: if ($t \geq w_\alpha$) then
20: if ($s \leq t$) then
21: $S(\alpha^t) \leftarrow S(\alpha^p)$; // copy node
22: else
23: $S(\alpha^t) \leftarrow$ non-dominated$\left(S((\alpha-1)^t) \cup \{(c_\alpha^1,...,c_\alpha^r) \oplus S((\alpha-1)^{t-w_\alpha})\}\right)$;
24: if ($t > b_\alpha$)
25: begin
26: $p \leftarrow t$;
27: $s \leftarrow$ the first element in TS_α not larger than t;
28: end
29: else
30: $S(\alpha^t) \leftarrow S((\alpha-1)^t)$; // copy node
31: end
32: end
33: end

Fig. 5. Procedure for the two state reduction based DP (TDP) algorithm.

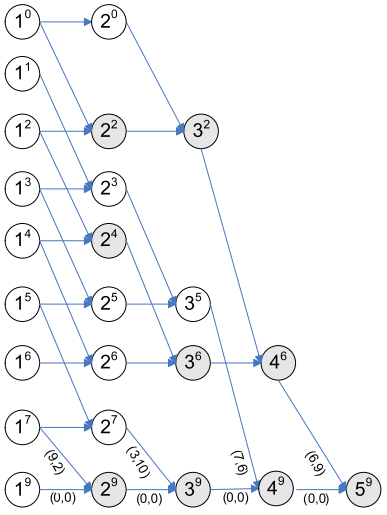

Fig. 6. Illustration for the states (grey) to actually calculate in the backward reduced-state DP space (BRDS) for the TDP algorithm.

all (pseudo) sparse nodes are needed beforehand because they are generated using a forward process as shown in Fig. 4. These nodes can be identified at the beginning of each stage (after Step 5 of Fig. 5) before the necessary nodes are calculated as the DP process goes on. Hence, only one stage of (pseudo) sparse nodes needs to be kept during the DP process. To get the (pseudo) sparse nodes of the next stage based on the nodes of the current stage, twice the memory requirements of

Table 1
Illustration of the calculation process for the basic sequential DP (BDP) procedure.

Node position	Stage				
	1	2	3	4	5
0	{(0, 0)}	{(0, 0)}	{(0, 0)}	{(0, 0)}	{(0, 0)}
1	{(0, 0)}	{(0, 0)}	{(0, 0)}	{(0, 0)}	{(0, 0)}
2	{(0, 0)}	{(9, 2)}	{(9, 2), (3, 10)}	{(9, 2), (3, 10)}	{(9, 2), (3, 10)}
3	{(7, 2)}	{(9, 2)}	{(9, 2), (3, 10)}	{(9, 2), (3, 10)}	{(9, 2), (6, 9),(3, 10)}
4	{(7, 2)}	{(9, 2)}	{(12, 12)}	{(12, 12)}	{(12, 12)}
5	{(7, 2)}	{(16, 4)}	{(16, 4), (12, 12)}	{(16, 4), (12, 12)}	{(16, 4), (15, 11), (12, 12), (9, 19)}
6	{(7, 2)}	{(16, 4)}	{(16, 4), (12, 12)}	{(16, 8), (12, 12), (10, 16)}	{(16, 8), (15, 11), (12, 12), (10, 16), (9, 19)}
7	{(7, 2)}	{(16, 4)}	{(19, 14)}	{(19, 14), (10, 16)}	{(19, 14), (18, 21)}
8	{(7, 2)}	{(16, 4)}	{(19, 14)}	{(19, 18)}	{(22, 13), (19, 18), (18, 21)}
9	{(7, 2)}	{(16, 4)}	{(19, 14)}	{(23, 10), (19, 18)}	{(23, 10), (22, 17), (19, 18), (18, 21), (16, 25)}

storing the nodes of the current stage are needed for storing both the nodes at the current stage and the new generated nodes. If all the (pseudo) sparse nodes of a stage are generated, the memory requirements for (pseudo) sparse nodes are $2(W + 1)$ because the number of (pseudo) sparse nodes at a stage is $(W + 1)$ in the worst case. However, for the TDP algorithm, only the (pseudo) sparse nodes with position not lower than the lowest position of the necessary nodes at each stage need to be generated (Step 9 of Fig. 4). The lowest position of the necessary nodes will increase as the stage increases (Fig. 6). Therefore, the number of (pseudo) sparse nodes can be reduced in the TDP algorithm. Consequently, the memory requirements for (pseudo) sparse nodes in the TDP algorithm should be a fraction of $2(W + 1)$. It means that memory requirements for pseudo sparse nodes are much smaller than those for the BRDS.

3.4. Algorithm illustration

We illustrate the results of the BDP algorithm in Table 1 using the example in Section 2. Next, we apply the TDP algorithm of Fig. 5 to the same instance to show how it works.

The core of the TDP algorithm is to determine which nodes are needed to calculate and which nodes take the objective vectors of the calculated nodes based on the property of (pseudo) sparse nodes. We roughly follow the steps of the algorithm in Fig. 5.

The preparation work for the entire algorithm is done from Steps 1 to 4. The necessary nodes in the BRDS and the (pseudo) sparse nodes are generated based on the procedures of Figs. 3 and 4 in Steps 2 and 3, respectively, and the initial values of the nodes are assigned in Step 4 of Fig. 5.

The positions of the necessary nodes (connected by thick arcs in Fig. 2) at each stage $\alpha(\alpha = 1, 2, 3, 4, 5)$, denoted by TN_α, are given below: $TN_1 = (9, 7, 6, 5, 4, 3, 2, 1, 0)$, $TN_2 = (9, 7, 6, 5, 4, 3, 2, 0)$, $TN_3 = (9, 6, 5, 2)$, $TN_4 = (9, 6)$, and $TN_5 = (9)$. The lowest positions of the necessary nodes at each stage α, denoted by b_α, are given by $b_1 = 0$, $b_2 = 0$, $b_3 = 2$, $b_4 = 6$, and $b_5 = 9$. These positions are used to restrict the lowest position of (pseudo) sparse nodes (refer to Step 9 of Fig. 4). The positions of the (pseudo) sparse nodes (the gray nodes in Fig. 2) at each stage α, denoted by TS_α, are given below: $TS_1 = (3, 0)$, $TS_2 = (5, 3, 2, 0)$, $TS_3 = (7, 5, 4, 3, 2)$, $TS_4 = (9, 8, 7, 6)$, and $TS_5 = (9)$. Table 1 shows that nodes 2^3 and 3^3 are pseudo sparse nodes.

In Step 4, the initial objective vectors of necessary nodes are assigned based on Eq. (4):

$$S(1^9) = S(1^7) = S(1^6) = S(1^5) = S(1^4) = S(1^3) = \{(7, 2)\}; S(1^2) = S(1^0) = \{(0, 0)\}.$$

From Steps 6 to 32, the objective vectors of the necessary nodes in the subsequent stages α ($\alpha = 2, 3, 4, 5$) are checked (either by calculating, or by taking the objective vectors of the calculated nodes). For the TDP algorithm, the calculation is processed from the node with the highest position to the node with the lowest position in each stage. Therefore, the results of the nodes are given based on the decreasing order of the position following the procedure.

At stage 2, the first node $2^9(t = 9)$ must be calculated: $S(2^9) =$ non-dominated $(S(1^9) \cup \{(9, 2) \oplus S(1^7)\}) =$ non-dominated $(\{(7, 2)\} \cup \{(9, 2) + (7, 2)\}) = \{(16, 4)\}$. After 9 is taken from TN_2, we check the condition in Step 11, $|TN_2| = 7 > 0$ is satisfied, then $p \leftarrow t = 9$, $s \leftarrow 5$ (the first element in TS_2 not higher than $t = 9$). Next, from Steps 17 to 31, we use the property of the (pseudo) sparse nodes to check which nodes should be calculated. In Step 18, we take the position of node 2^7, $t = 7$, from TN_2, then we check the condition in Step 19. Since $t \geq w_2(= 2)$ is satisfied, then we need to check the condition in Step 20, $s = 5 \leq t = 7$ is satisfied, $S(2^7) \leftarrow S(2^9) = \{(16, 4)\}$. The situations of nodes 2^6 and 2^5 are the same as that of 2^7, $S(2^6) \leftarrow S(2^9) = \{(16, 4)\}$; and $S(2^5) \leftarrow S(2^9) = \{(16, 4)\}$. For the next node $2^4(t = 4)$, the condition in Step 20 is not satisfied, we go to Step 23 to determine $S(2^4) =$ non-dominated $(S(1^4) \cup \{(9, 2) \oplus S(1^2)\}) =$ non-dominated $(\{(7, 2)\} \cup \{(9, 2) + (0, 0)\}) = \{(9, 2)\}$. After this, we check the condition in Step 24, $t = 4 > b_2(= 0)$ is satisfied, then p and s are updated based on Steps 26 and 27, $p \leftarrow t = 4$; $s \leftarrow 3$ (the first element in TS_2 not higher than $t = 4$). For the next node $2^3(t = 3)$, the condition in Step 20 is satisfied, $S(2^3) \leftarrow S(2^4) = \{(9, 2)\}$. For node $2^2(t = 2)$, the condition in Step 20 is not satisfied, we go to Step 23 to determine $S(2^2) =$ non-dominated $(S(1^2) \cup \{(9, 2) \oplus S(1^0)\}) =$ non-dominated$(\{(0, 0)\} \cup \{(9, 2) + (0, 0)\}) = \{(9, 2)\}$. The condition in Step 24, $t = 2 > b_2(= 0)$ is satisfied and p and s are updated based on Steps 26 and 27, $p \leftarrow t = 2$; $s \leftarrow 2$. For the final node $2^0(t = 0)$, the condition in Step 19 is not satisfied, we go to Step 30, $S(2^0) \leftarrow S(1^0) = \{(0, 0)\}$.

For the remaining stages $\alpha(\alpha = 3, 4, 5)$, we can repeat the similar process as for stage 2.

At stage 3, $S(3^9)$, $S(3^6)$ and $S(3^2)$ must be calculated and $S(3^5) \leftarrow S(3^6)$: $S(3^9)$ = non-dominated $(S(2^9) \cup \{(3, 10) \oplus S(2^7)\})$ = non-dominated $(\{(16, 4)\} \cup \{(3, 10) + (16, 4)\})$ = $\{(19, 14)\}$; $S(3^6)$ = non-dominated $(S(2^6) \cup \{(3, 10) \oplus S(2^4)\})$ = non-dominated $(\{(16, 4)\} \cup \{(3, 10) + (9, 2)\})$ = $\{(16, 4), (12, 12)\}$; $S(3^5) \leftarrow S(3^6)$ = $\{(16, 4), (12, 12)\}$; $S(3^2)$ = non-dominated $(S(2^2) \cup \{(3, 10) \oplus S(2^0)\})$ = non-dominated $(\{(9, 2)\} \cup \{(3, 10) + (0, 0)\})$ = $\{(9, 2), (3, 10)\}$.

At stage 4, all the nodes need to be calculated: $S(4^9)$ = non-dominated $(S(3^9) \cup \{(7, 6) \oplus S(3^5)\})$ = non-dominated $(\{(19, 14)\} \cup \{\{(7, 6)\} \oplus \{(16, 4), (12, 12)\}\})$ = $\{(23, 10), (19, 18)\}$; $S(4^6)$ = non-dominated $(S(3^6) \cup \{(7, 6) \oplus S(3^2)\})$ = non-dominated $(\{(16, 4), (12, 12)\} \cup \{\{(7, 6)\} \oplus \{(9, 2), (3, 10)\}\})$ = $\{(16, 8), (12, 12), (10, 16)\}$.

At the last stage 5, only one node must be calculated: $S(5^9)$ = non-dominated $(S(4^9) \cup \{(6, 9) \oplus S(4^6)\})$ = non-dominated $(\{(23, 10), (19, 18)\} \cup \{\{(6, 9)\} \oplus \{(16, 8), (12, 12), (10, 16)\}\})$ = $\{(23, 10), (22, 17), (19, 18), (18, 21), (16, 25)\}$.

From the above illustration and Table 1, it can be seen that the TDP algorithm works properly for the example in Section 2. The BRDS avoids calculating some nodes with a heavy computational effort at the late stages, e.g. nodes in stages 4 and 5. Applying (pseudo) sparse nodes further helps to reduce the number of nodes to calculate the BRDS (see Fig. 6) at the early stages, e.g., nodes at stages 2 and 3.

4. Numerical results

To test the performance of the TDP algorithm and the contributions of the two reduction techniques to the solution time efficiency, we implemented the TDP algorithm and the DP algorithm using only the BRDS (RDP) in C++ in the Microsoft visual studio 2003 environment. All the experiments were carried out on a 2.49 GHz Pentium PC with 2.98 GB RAM under Windows XP operating systems. We used the DDP algorithm provided by an author of [17] as the benchmark. To the authors' knowledge, the DDP algorithm is the best algorithm for dealing with the hard type of instances efficiently. The algorithm presented in [18] can handle the non-hard type of instances better than the DDP algorithm. In this study, we considered only randomly generated BKP instances and used the same types of instances as [17,18].

4.1. Test instances

The numerical experiments were referred with the following types of instances.

(A) Uncorrelated instances: $c_j^1 \in_R [1, 1000]$, $c_j^2 \in_R [1, 1000]$, and $w_j \in_R [1, 1000]$.

(B) Unconflicting instances, where c_j^1 is positively correlated with c_j^2: $c_j^1 \in_R [101, 1000]$, $c_j^2 \in_R [c_j^1 - 100, c_j^1 + 100]$ and $w_j \in_R [1, 1000]$.

(C) Conflicting instances, where c_j^1 is negatively correlated with c_j^2: $c_j^1 \in_R [1, 1000]$, $c_j^2 \in_R [\max\{900 - c_j^1, 1\}, \min\{1100 - c_j^1, 1000\}]$ and $w_j \in_R [1, 1000]$.

(D) Conflicting instances with correlated weight, where c_j^1 is negatively correlated with c_j^2 and w_j is positively correlated with c_j^1 and c_j^2: $c_j^1 \in_R [1, 1000]$, $c_j^2 \in_R [\max\{900 - c_j^1, 1\}, \min\{1100 - c_j^1, 1000\}]$ and $w_j \in_R [c_j^1 + c_j^2 - 200, c_j^1 + c_j^2 + 200]$.

Here $u \in_R [a, b]$ denotes that u is a uniform random number in $[a, b]$. For all the instances, we set $W = \lfloor 1/2 \sum_{j=1}^{n} w_j \rfloor$.

As commented by Bazgan et al. [17], most experiments were made only with instances of Type A. Instances of Type B were introduced in [16], which should be viewed as quasi-single objective instances since two "unconflicting" objectives are involved. Instances of Types C and D were newly defined in [17] because conflicting objectives are more appropriate to model real-world situations. These are hard instances. Instances of Type D are harder than those of Type C because the weight coefficients in Type D are correlated with the profit objectives. Similar to the single objective case, correlated instances are more difficult to solve than uncorrelated instances [8]. In fact, very few techniques are effective to correlated instances [1].

We have tested the TDP algorithm for all the above four types of instances. We generated 30 instances for each problem size n (number of items in the knapsack) and the average results were obtained over the 30 instances.

4.2. Computational results

Before we report the results for the algorithm, we investigate the impact of item orders. The order of items plays an important role in implementing our DP algorithm for the MKP because solution time efficiency of the algorithm is affected by the order of items. Ref. [17] proposed the MKP three orders, O^{sum}, $O^{\max}$ and $O^{\min}$ motivated by the profit-to-weight ratio for the single objective 0–1 KP. The related three measures are derived from the profit-to-weight ratios c_j^k/w_j for each objective k of item j. Let h_j^k be the rank of item j based on c_j^k/w_j, then the higher the c_j^k/w_j, the higher h_j^k. O^{sum} represents the order based on the sum of the ranks $(\sum_{k=1}^{r} h_j^k)$, $O^{\max}$ the order based on maximum rank $(\max_{k=1,\ldots,r}\{h_j^k\} + 1/r \sum_{k=1}^{r} h_j^k)$ and $O^{\min}$ the order based on the minimum rank $(\min_{k=1,\ldots,r}\{h_j^k\} + 1/r \sum_{k=1}^{r} h_j^k)$. The measures for the maximum and the minimum rank are slightly different from those defined in [17]. We do not consider the number of items n as a divisor in the second term of the measure. Here we introduce a fourth order O^{weight}, based on the weight of items. We have implemented all these orders in our DP algorithm.

Table 2 shows that instances of Types A and B respond better to the order O^{weight} and instances of Types C and D respond better to the order $O^{\min}$. The increase (expressed in percentage) of the solution time compared with the time

Table 2
Average solution time (s) for different item orders.

Type	n	O^{weight}	O^{sum}	$O^{\max}$	$O^{\min}$	Random
A	200	45.34 (−14%)	61.52 (+17%)	63.36 (+20%)	50.49 (−4%)	52.61
B	300	11.71 (−21%)	17.27 (+15%)	17.08 (+14%)	16.30 (+9%)	14.96
C	120	20.93 (−12%)	20.39 (−14%)	23.03 (−3%)	15.91 (−33%)	23.83
D	60	8.13 (−7%)	9.23 (−7%)	7.20 (−17%)	6.17 (−29%)	8.65

The increase (expressed in percentage) of the solution time compared with the time obtained when items are randomly selected is given in brackets.

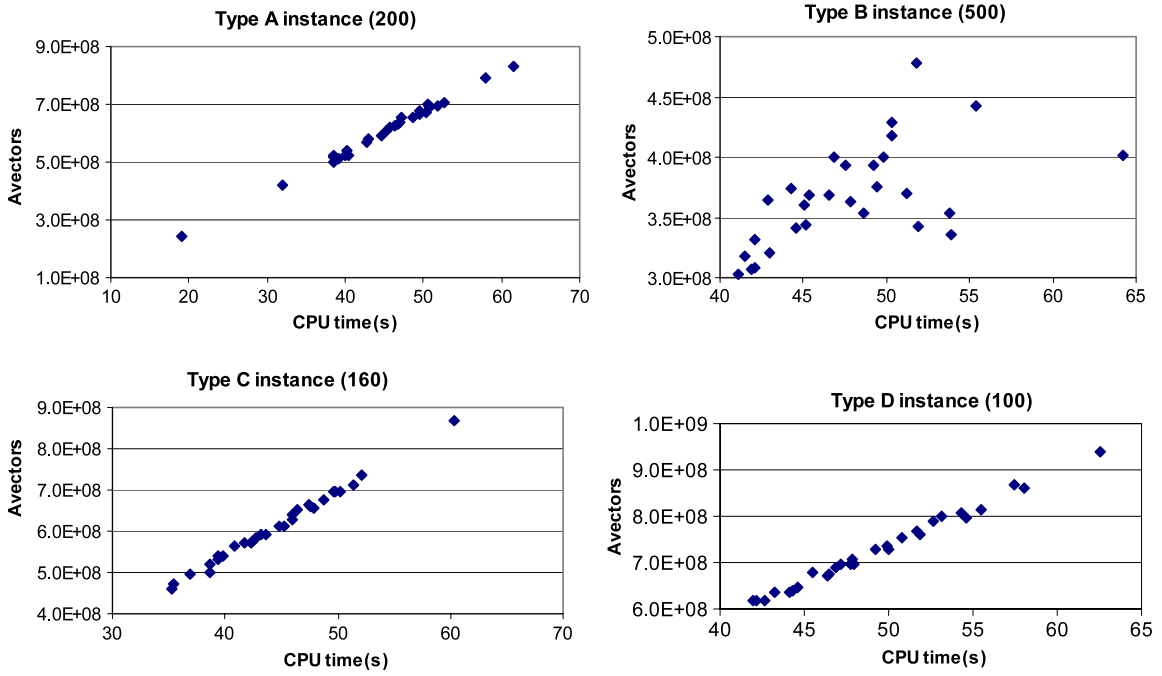

Fig. 7. Relations between CPU time and Avectors for different types of instances.

obtained when items are selected randomly is given in brackets. The best orders of the RDP algorithm are the same as those of the TDP algorithm. In the subsequent reports, we give the results for different types of instances based on their best orders.

Next, we investigate the relations between the solution time (CPU time) and the number of actually calculated objective vectors (Avectors) during the process for the TDP algorithm based on instances of Type A with size 200, Type B with size 500, Type C with size 160 and Type D with size 100 (See Fig. 7). The average solution times of these instances are close (refer to Table 3).

Based on Fig. 7, for instances of Types A, C, and D, the number of actually calculated objective vectors (Avectors) is positively correlated with the solution time. In fact, Avectors determines the solution time because the time for calculating the objective vectors in the DP process is much larger than the generation time for the BRDS and for pseudo sparse nodes. However, for instances of Type B, it seems that the solution time is not solely determined by Avectors. It means that generation time for the BRDS and for pseudo sparse nodes cannot be ignored for instances of Type B due to the fact that this generation time is comparable to the time for calculating the objective vectors in the DP process.

Table 3 gives the average results related to the solution time of the TDP algorithm for all the four types of instances. For determining the problem sizes for the different types of instances, on the one hand, the largest problem size is limited by the memory capacity of the computer. On the other hand, the smallest problem size is determined by three factors: the solution time and the comparative performance with the benchmark algorithm, as well as the special requirements for comparison. The solution time should be large enough to record and the chosen size can identify the problem size where the relative performance of the TDP algorithm and the benchmark DDP algorithm change. In terms of special requirements for comparison, we give an example to illustrate it. For the problem size 100 of Type B, the solution time for the DDP algorithm is too small to record accurately. However, for the TDP algorithm, the solution time is large enough to record. We need to compare the number of non-dominated objective vectors of this size for all the four types of instances as demonstrated below.

Table 3 introduces three ratios (RT, AR and AT): RT is the ratio of necessary nodes and total nodes (refer to Fig. 2); AR is the ratio of actually calculated nodes and necessary nodes (refer to Fig. 6); AT is the ratio of actually calculated nodes and total nodes (refer to Figs. 2 and 6).

Table 3
Average results related to solution time for all the four types of instances.

Type	n	Avectors					Solution	TNode
		(10^6)	*ND*	RT	AR	AT	Time (s)	(10^5)
A	100	46.43	163.2	0.525	0.594	0.312	3.70	25.38
	150	209.09	328.9	0.526	0.637	0.337	15.21	56.94
	200	609.52	522.8	0.527	0.662	0.349	45.34	99.86
	220	855.80	625.2	0.528	0.668	0.351	60.00	120.75
	240	1188.42	728.7	0.528	0.673	0.355	85.93	143.71
	260	1610.07	841.1	0.528	0.678	0.358	112.82	168.99
	280	2076.30	931.7	0.528	0.681	0.359	144.69	196.21
B	100	2.21	5.8	0.525	0.594	0.312	0.71	25.38
	250	37.40	17.4	0.527	0.676	0.357	7.21	156.15
	300	67.47	23.4	0.527	0.685	0.361	11.71	224.90
	350	105.51	26.0	0.528	0.691	0.364	17.53	304.90
	400	169.93	32.0	0.528	0.696	0.367	24.07	498.18
	450	254.37	40.9	0.528	0.700	0.369	34.35	502.22
	500	365.03	48.6	0.528	0.703	0.371	47.58	624.29
C	60	12.39	217.8	0.692	0.415	0.287	1.16	9.04
	80	40.64	350.7	0.738	0.438	0.323	3.46	16.08
	100	101.80	511.6	0.763	0.451	0.344	8.53	25.38
	120	202.34	678.8	0.780	0.456	0.356	15.91	36.54
	140	362.47	867.7	0.792	0.461	0.365	27.82	49.63
	160	610.41	1080.3	0.800	0.467	0.374	44.46	64.58
	180	934.66	1291.0	0.807	0.467	0.377	67.83	81.27
	200	1371.52	1495.5	0.812	0.468	0.380	98.81	99.86
D	30	1.77	240.0	0.429	0.202	0.087	0.21	4.50
	40	11.18	392.1	0.528	0.362	0.191	1.02	8.00
	50	36.18	569.1	0.579	0.441	0.256	2.88	12.51
	60	82.59	776.9	0.617	0.490	0.303	6.17	18.05
	70	164.51	990.8	0.641	0.521	0.334	11.91	24.55
	80	289.00	1248.0	0.660	0.546	0.361	20.08	32.08
	90	462.00	1525.8	0.673	0.561	0.377	31.53	40.60
	100	725.47	1765.0	0.684	0.573	0.392	49.29	50.03
	110	1050.67	2128.0	0.693	0.581	0.403	69.41	60.52

Avectors (10^6): number of actually calculated objective vectors (the value in brackets is the order of the quantity); *ND*: number of non-dominated objective vectors; RT: ratio of necessary nodes and total nodes (refer to Fig. 2); AR: ratio of actually calculated nodes and necessary nodes (refer to Fig. 6); AT: ratio of actually calculated nodes and total nodes (refer to Figs. 2 and 6); TNode (10^5): number of nodes in the BDP process (refer to Fig. 1) (the value in brackets is the order of the quantity).

The ratios RT, AR and AT reflect the contribution of the reduction techniques in terms of node reduction. Table 3 shows that the three ratios increase as the problem size increases for all the four types of instances. Instances of Type D increase more significantly than those of Types A, B and C. This means that instances of Type D are more difficult to solve than those of Types A, B and C.

Moreover, the tractability of the instances is reflected by the average number of non-dominated objective vectors (*ND*). As Table 3 shows, the *ND* is approximately 6, 163, 431 and 1765 for instances of Types B, A, C and D, respectively, with the same size, i.e., 100 items, considered in the test. It means that instances of Type D are harder than those of Type C, which in turn are harder than those of Type A, which in turn are harder than those of Type B because the computational effort for obtaining the PF increases with *ND*.

In the TDP algorithm, the computational effort can be measured by Avectors. Based on Fig. 7, for instances of Types A, C, and D, the computational effort is proportional to Avectors while for instances of Type B, the computational effort is not solely determined by Avectors. It is worth mentioning that the computational effort is not proportional to the number of states (nodes) to calculate because the number of calculated objective vectors is different from node to node. However, reducing the number of calculated nodes can reduce the computational effort when the problem size is reasonably large.

Next, we give the memory requirements of the TDP algorithm. For the current implementation, both the nodes and arcs of the BRDS are stored in the memory before the DP process starts. The memory requirements for the arcs are twice the memory requirements for the nodes because each node has at most two incoming arcs. However, the TDP algorithm can also be implemented without storing arcs beforehand since the arcs can be determined dynamically when the nodes are known. The solution time should be a little larger than the current solution time because dynamical determination of arc links will consume some amount of time. The memory requirements for the current TDP algorithm are the sum of the following components: nodes (BRDS_N) and arcs (BRDS_A) of the BRDS, two stages of pseudo sparse node (Sparse) and one stage of the objective vectors (Vectors) during the DP process. Table 4 gives the memory requirements for these components and the memory requirements for the BDP algorithm in mega bytes (MB) as well as some indicators related to the memory

Table 4
Average results related to memory requirements for all the four type of instances.

Type	n	Memory usage (MB)					NC	RD
		BDP	TDP					
			BRDS_N	BRDS_A	Sparse	Vectors		
A	100	38.4	10.1	20.2	0.18	14.5	0.997	1.36
	150	110.8	22.8	45.5	0.29	39.2	0.999	1.23
	200	233.6	39.9	79.9	0.38	82.5	0.999	1.22
	220	299.1	48.3	96.6	0.42	102.1	0.999	1.17
	240	381.4	57.5	114.9	0.46	128.3	0.999	1.18
	260	471.0	67.6	135.2	0.51	157.7	0.999	1.19
	280	563.4	78.5	156.9	0.55	189.0	0.999	1.15
B	100	2.8	10.1	20.2	0.18	1.8	0.997	2.91
	250	16.6	62.4	124.9	0.48	10.3	0.999	2.01
	300	25.2	89.9	179.9	0.58	15.0	0.999	1.96
	350	34.5	121.9	243.9	0.68	19.8	0.999	2.14
	400	46.6	159.3	318.5	0.78	24.8	0.999	2.02
	450	62.2	202.1	404.2	0.88	32.2	0.999	1.86
	500	79.8	249.7	499.4	0.98	41.0	0.999	1.81
C	60	29.2	3.6	7.1	0.12	12.5	0.985	1.12
	80	59.7	6.4	12.8	0.16	25.7	0.993	1.06
	100	111.3	10.1	20.2	0.20	46.2	0.996	1.07
	120	173.3	14.6	29.2	0.24	71.1	0.997	1.05
	140	257.6	19.8	39.6	0.28	103.1	0.998	1.04
	160	360.8	25.8	51.6	0.32	144.8	0.999	1.03
	180	480.7	32.5	65.0	0.36	191.6	0.999	1.03
	200	614.6	39.9	79.8	0.40	247.2	0.999	1.03
D	30	30.4	1.5	3.1	0.09	4.8	0.849	1.06
	40	67.4	2.9	5.8	0.14	15.0	0.901	1.09
	50	117.7	4.6	9.3	0.18	31.8	0.926	1.04
	60	192.8	6.8	13.6	0.22	52.7	0.943	1.03
	70	285.0	9.4	18.7	0.26	81.7	0.953	1.03
	80	410.7	12.3	24.7	0.30	113.7	0.961	1.03
	90	561.8	15.7	31.4	0.34	152.9	0.966	1.02
	100	725.5	19.4	38.8	0.38	209.0	0.970	1.03
	110	957.3	23.6	47.1	0.42	268.8	0.973	1.02

NC: ratio of the number of nodes in the first stage of the BRDS and $(W+1)$ (refer to Figs. 1 and 2); RD: ratio of maximum number of objective vectors of a state for the TDP algorithm and *ND* (refer to Table 3).

requirements. The memory requirements for the BDP algorithm are estimated based on $(W+1)$ multiplied by the maximum number of objective vectors of a node for the TDP algorithm.

Table 4 introduces two ratios (NC and RD): NC is the ratio of the number of nodes in the first stage of the BRDS and $(W+1)$ (the number of nodes in each stage of the BDP algorithm); RD is the ratio of maximum number of objective vectors of a node for the TDP algorithm and *ND* (refer to Table 3).

The ratio NC gives the relative memory requirements for storing the BRDS (node) with respect to those for the BDP algorithm for the single objective 0–1 KP (see Section 3.1). Based on NC, it can be seen that the BRDS based algorithm is not attractive to the single objective 0–1 KP because the memory requirements for storing both the BRDS and the results of the algorithm are nearly doubled (refer to Section 3.1) as compared with those for the BDP algorithm for the instances in the experiment.

The ratio RD reflects the relative number of objective vectors of a node for the TDP algorithm with respect to *ND*. It can be seen that RD $>$ 1. The larger the *ND*, the larger the memory requirements for storing the objective vectors for both the TDP and BDP algorithms.

For the MKP, the effectiveness of the BRDS should be determined by the relative memory requirements for the TDP algorithm (the sum of memory requirements for all the four components as mentioned above and shown in Table 4 for the current TDP algorithm) against those for the BDP algorithm as well as the relative memory requirements for the BRDS (node) with respect to those for storing the objective vectors as discussed in Section 3.1. It can be seen that the memory requirements for the TDP algorithm are smaller than those for the BDP algorithm (with exception of small size instances of Type A) and the memory requirements for the BRDS are smaller than those for the objective vectors for instances of Types A, C and D. However, for instances of Type B, the results are the opposite. It means that the BRDS is effective to instances of Types A, C and D, especially for instances of Type D but not effective to instances of Type B at all. For instances of Type D, the memory requirements are about one third of those for the BDP algorithm.

Table 5
Memory requirements and solution times for the RDP, TDP and DDP algorithms.

Type	n	Memory usage (MB)			Solution time (s)			GAP (%)	
		RDP[a]	TDP[a]	DDP	RDP	TDP	DDP	RDP	TDP
A	100	44.9	45.1	0.53	3.92	3.70	2.02	93.8	82.8
	150	107.5	107.8	2.13	15.82	15.21	13.29	19.0	14.4
	200	202.3	202.7	5.42	46.23	45.34	47.37	−2.4	−4.3
	220	246.9	247.4	7.75	62.57	61.00	75.56	−17.2	−19.3
	240	300.7	301.1	10.69	86.36	85.93	114.90	−24.8	−25.2
	260	360.5	361.0	13.98	114.34	112.82	165.70	−31.0	−31.9
	280	424.4	425.0	17.80	146.86	144.69	226.66	−35.2	−36.2
B	250	197.6	198.1	0.04	8.19	7.21	0.09	9537	8380
	300	284.8	285.4	0.06	13.23	11.71	0.17	7685	6788
	350	385.6	386.3	0.10	18.97	17.53	0.30	6222	5742
	400	502.6	503.4	0.14	25.12	24.07	0.48	5133	4915
	450	638.4	639.3	0.22	34.98	34.35	0.85	4015	3941
	500	790.1	791.1	0.33	47.51	47.58	1.51	3046	3051
C	60	23.2	23.3	0.41	1.25	1.16	0.80	55.7	45.1
	80	44.9	45.0	0.88	3.53	3.46	2.90	21.7	19.4
	100	76.6	76.8	1.88	8.81	8.53	7.84	12.4	8.8
	120	114.8	115.0	3.50	16.52	15.91	17.55	−5.8	−9.4
	140	162.5	162.8	5.90	28.56	27.82	35.48	−19.5	−21.6
	160	222.2	222.5	9.51	45.20	44.46	66.99	−32.5	−33.6
	180	289.0	289.4	13.63	68.80	67.83	109.41	−37.1	−38.0
	200	367.0	367.4	18.84	99.82	98.81	167.91	−40.5	−41.2
D	30	9.4	9.5	0.29	0.24	0.21	0.38	−36.0	−44.9
	40	23.6	23.8	0.93	1.06	1.02	1.73	−38.4	−41.0
	50	45.7	45.9	2.02	3.06	2.88	5.33	−42.5	−46.0
	60	73.1	73.3	3.97	6.44	6.17	12.32	−47.8	−49.9
	70	109.8	110.0	10.07	12.35	11.91	27.95	−55.8	−57.4
	80	150.7	151.0	12.33	20.55	20.08	53.04	−61.2	−62.1
	90	200.0	200.3	18.51	32.49	31.53	89.94	−63.9	−64.9
	100	267.2	267.6	33.61	50.53	49.29	146.94	−65.6	−66.5
	110	339.5	339.9	38.81	70.59	69.41	230.11	−69.3	−69.8

[a] The memory requirements for the TDP and RDP algorithms include those for storing the arcs of the BRDS (refer to Table 4), which are not necessary as discussed in the text. GAP (%) is the relative solution time of the RDP and TDP algorithms against the DDP algorithm expressed in percentage.

Next, the memory requirements and solution time of the TDP, RDP and DDP algorithms are compared based on Table 5.

In terms of memory requirements, the TDP algorithm is roughly the same as the RDP algorithm because additional memory requirements of the TDP algorithm are those for storing (pseudo) sparse nodes, which are small as compared with those for storing the BRDS as discussed at the end of Section 3.3 and shown in Table 4. Here we stress again that the memory requirements for the TDP and RDP algorithms should be smaller than those shown in Table 5 if the arcs of the BRDS are not stored. It can be seen that the relative memory requirements for the RDP and TDP algorithms with respect to those for the DDP algorithm have tendency to decrease as the problem size increases even though the memory requirements for both the TDP and RDP algorithms are much larger than those for the DDP algorithm. There are two implications for this. On the one hand, it means that dominance relations used in the DDP algorithm can help to reduce the memory requirements significantly. On the other hand, it means that the effectiveness of dominance relations decreases as the problem size increases. In addition, it should be realized that applying dominance relations is time consuming.

In terms of solution (CPU) time, the RDP and TDP algorithms are compared against the DDP algorithm based on GAP (%) given by $100(T_s - T_b)/T_b$ (%), where T_s and T_b represent the solution time of the subject and benchmark algorithms, respectively. The negative value means that the subject algorithm performs better.

Based on Table 5, the TDP algorithm shows a small improvement over the RDP algorithm for all the four types of instances (with the exception of size 500 of Type B) considered in the test. We will discuss the relative advantage of the TDP algorithm over the RDP algorithm later. Now, we discuss the relative solution time of the TDP with respect to that of the DDP algorithm. The TDP algorithm is much worse than the DDP algorithm for instances of Type B. This means that the BRDS based algorithm is also not favorable to instances of Type B in terms of solution time as compared with the DDP algorithm. The DDP algorithm used the bounding dominance relation to discard partial solutions. In fact, for dealing with instances of Type B, the DP algorithm based on more elaborate bounding techniques [18] performs better than the DDP algorithm. It means that the bounding techniques based algorithms are favorable to instances of Type B. It is well known that the bounding technique is very effective to deal with some types of single objective 0–1 KPs (see [1,8]). Therefore, it is not surprising that both the

Table 6
Comparison of the TDP and RDP algorithms using solution time indicators.

Type	n	Avectors (10^6)		AR		AT	
		RDP	TDP	RDP	TDP	RDP	TDP
A	100	48.12	46.43	0.963	0.594	0.506	0.312
	150	212.5	209.09	0.975	0.637	0.515	0.337
	200	615.1	609.52	0.981	0.662	0.517	0.349
	220	862.5	855.8	0.983	0.668	0.517	0.351
	240	1196	1188.4	0.984	0.673	0.519	0.355
	260	1619	1610.1	0.986	0.678	0.521	0.358
	280	2087	2076.3	0.987	0.681	0.521	0.359
B	250	40.51	37.4	0.985	0.676	0.520	0.357
	300	71.88	67.47	0.987	0.685	0.521	0.361
	350	115.4	109.51	0.989	0.691	0.521	0.364
	400	177.4	169.93	0.991	0.696	0.522	0.367
	450	263.8	254.37	0.992	0.700	0.523	0.369
	500	376.5	365.03	0.992	0.703	0.523	0.371
C	60	12.79	12.39	0.955	0.415	0.660	0.287
	80	41.33	40.64	0.967	0.438	0.714	0.323
	100	102.9	101.8	0.975	0.451	0.744	0.344
	120	203.9	202.34	0.979	0.456	0.764	0.356
	140	364.6	362.47	0.982	0.461	0.778	0.365
	160	613.2	610.41	0.985	0.467	0.788	0.374
	180	938.1	934.66	0.986	0.467	0.796	0.377
	200	1376	1371.5	0.988	0.468	0.802	0.380
D	30	2.69	1.77	0.877	0.202	0.377	0.087
	40	12.58	11.18	0.918	0.362	0.485	0.191
	50	38.06	36.18	0.938	0.441	0.543	0.256
	60	84.91	82.59	0.950	0.490	0.587	0.303
	70	167.4	164.51	0.958	0.521	0.615	0.334
	80	292.1	289	0.964	0.546	0.637	0.361
	90	465.7	462	0.969	0.561	0.652	0.377
	100	729.7	725.47	0.972	0.573	0.665	0.392
	110	1055	1050.7	0.975	0.581	0.676	0.403

Avectors (10^6): number of actually calculated objective vectors (the value in brackets is the order of the quantity); AR: ratio of actually calculated nodes and necessary nodes (refer to Fig. 6); AT: ratio of actually calculated nodes and total nodes (refer to Figs. 2 and 6).

DDP algorithm and the DP algorithm presented in [18] give better results for instances of Type B, which are quasi-single objective instances. Ref. [9] explicitly excluded the instances for which the bounding techniques are effective from hard instances.

In terms of instances of Types A and C, the TDP algorithm is better than the DDP algorithm for large sizes while worse for small sizes. In terms of instances of Type D, the TDP algorithm performs better than the DDP algorithm for all sizes. However, the solution time of the TDP algorithm is too small to record accurately for small size instances. Therefore, we do not include these results in the table. Based on the numerical tests, the smallest recorded non-zero time is 0.015 s. It means that the recorded time is 0 if it is less than 0.015 s. In other words, the recorded time is not accurate if it is 0. The overall tendency is that the TDP algorithm performs better as the problem size increases, especially good for hard type instances (Type D) as compared with the DDP algorithm. This is a positive asset of the BRDS based algorithm.

Based on the above comparisons, both the relative memory requirements and the relative solution time of the TDP algorithm with respect to the DDP algorithm has tendency to decrease as the problem size increases. Even though the DDP algorithm has advantage over the TDP algorithm in terms of memory requirements (see Table 5), it can be concluded that the reduction techniques introduced in the TDP algorithm are more effective than the dominance relations introduced in [17] to deal with the hard type instances. Especially, the BRDS is valuable for the MKP, which is also more effective than the FRDS used in the labeling algorithm [16] because the DDP algorithm performs better than the labeling algorithm [17]. The condition (9) for determining the effectiveness of the BRDS is not difficult to satisfy for the typical instances and the types of instances where the number of efficient solutions is exponential with respect to n [20].

Finally, we discuss the contributions of two reduction techniques in the TDP algorithm. Table 6 gives the performance indicators related to the solution time for the TDP and RDP algorithms. Based on the column of solution time and GAP measure in Table 5, it seems that the advantage of the TDP algorithm over the RDP algorithm (the GAP difference between the TDP and the RDP algorithms) has tendency to decrease as the problem size increases for instances of Types B, C and D. For instances of Type B with size 500, the TDP algorithm performs worse than the RDP algorithm. For instances of Type A,

this tendency is not clear from the current results. It means that the BRDS is a primary reduction technique and applying (pseudo) sparse nodes is a secondary (or an auxiliary) reduction technique to achieve further reduction of the solution time when the problem size is reasonably large.

Next, we explore the reason behind this phenomenon. When the columns AR and AT are compared with the column Avectors in Table 6, we see that the reduction percentage of Avectors is much smaller than that of nodes (AR and AT). Hence, applying (pseudo) sparse nodes mainly helps to reduce the nodes with a smaller number of objective vectors (thus a light computational effort) at the early stages while the BRDS can reduce the nodes with a larger number of objective vectors (thus a heavy computational effort) at the late stages. The two reduction techniques complement each other. If the computational effort for generating (pseudo) sparse nodes and for applying the (pseudo) sparse nodes to determine which nodes should be calculated (refer to Fig. 5) is larger than that for calculating the objective vectors of the nodes, then applying (pseudo) sparse nodes can worsen the solution time for the TDP algorithm.

5. Conclusion

In this paper, we have developed a new DP algorithm for the multi-objective 0–1 knapsack problem (MKP) by using two state reduction techniques. The reductions are carried out by exploring the network underlying the basic sequential DP (BDP) process. One reduction is concerned with generating a backward reduced-state DP space (BRDS) by discarding some states in the BDP space. The other one is concerned with achieving further state calculation reduction by using a property governing the objective relations between states. The above two reduction techniques complement each other. The former one reduces the computational effort for the states at the late stages for the DP algorithm while the latter one contributes to achieving reduction for the states at the early stages. We analyze the impact of the BRDS on the single-objective KP and MKP in terms of solution efficiency and memory requirements and find out that it has negative effect on the single objective knapsack problem (KP). We derive the condition under which the BRDS is effective for the MKP, which is not difficult to satisfy for the typical instances, and conclude that the BRDS is a valuable asset for the MKP. The BRDS is particularly favorable to the MKP where the cardinality of the PF increases fast as the problem size increases.

The novelty of the algorithm lies in combining the above two complementary reduction techniques to achieve the goal of reducing the number of objective vectors to calculate during the DP process, which determines the solution time efficiency of the algorithm. To the authors' knowledge, it is the first time that the BRDS is applied to develop a DP algorithm for the MKP.

The numerical experiment with bi-objective instances showed that the new algorithm is faster than the benchmark algorithm [17] for three of the four types of instances though the memory requirements for the new algorithm are larger than those for the benchmark algorithm. Furthermore, the TDP algorithm shows advantage over the BDP algorithm in terms of memory requirements for three of the four types of instances, which lays the foundation for introducing the BRDS for the MKP. In terms of solution time, the larger the problem size, the larger the improvement over the benchmark. The new algorithm is especially effective to hard type knapsack instances with conflicting objectives, which model real-world applications, as compared with the benchmark. Finally, the reduction techniques are general techniques for the KP and the new algorithm is applicable to the multi-objective knapsack optimization with more than two objectives.

Acknowledgments

The authors are grateful to Hadrien Hugot for providing a copy of their dynamic programming algorithm in Ref. [17]. The first author would like to thank FCT (Fundação para a Ciência e a Tecnologia, Portuguese Science and Technology foundation) support through program POCTI 2010 for funding this research. The third author was partially supported by the FCT research project POCTI/ISFL/152.

References

[1] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Berlin, 2004.
[2] M.J. Rosenblatt, Z. Sinuany-Stern, Generating the discrete efficient frontier to capital budgeting problem, Oper. Res. 37 (1989) 384–394.
[3] J. Teng, G.A. Tzeng, A mutliobjective programming approach for selecting non-independent transportation alternatives, Transp. Res. B. 30 (1996) 291–307.
[4] M.M. Kostreva, W. Ogryczak, D.W. Tonkyn, Relocation problems arising in conservation biology, Comput. Math. Appl. 37 (1999) 135–150.
[5] K. Alanne, Selection of renovation actions using multi-criteria knapsack model, Autom. Constr. 13 (2004) 377–391.
[6] A. Higgins, S. Hajkowicz, E. Bui, A multi-objective model for environmental investment decision-making, Comput. Oper. Res. 35 (2008) 253–266.
[7] R.Z. Farahani, M. SteadieSeifi, N. Asgari, Multiple criteria facility location problems: a survey, Appl. Math. Model. 34 (2010) 1689–1709.
[8] S. Martello, P. Toth, Upper bounds and algorithms for hard 0–1 knapsack problems, Oper. Res. 45 (1997) 768–778.
[9] D. Pisinger, Where are hard knapsack problems? Comput. Oper. Res. 32 (2005) 2271–2284.
[10] M. Ehrgott, Multictriteria Optimization, second ed., Springer, Berlin, 2005.
[11] P. Serafini, Some considerations about computational complexity for multi-objective combinatorial problems, in: J. Jahn, W. Krabs (Eds.), Recent Advances and Historical Development of Vector Optimization, in: Lecture Notes in Economics and Mathematical Systems, vol. 294, Springer, Berlin, 1986, pp. 222–232.
[12] K. Klamroth, M. Wiecek, Dynamic programming approach to the multiple criteria knapsack problems, Nav. Res. Logisti. Q. 47 (2000) 57–67.
[13] F. Jolai, M.J. Rezaee, M. Rabbani, J. Razmi, P. Fattahi, Exact algorithm for bi-criteria 0–1 knapsack problems, Appl. Math. Comput. 194 (2007) 544–551.
[14] J. Figueira, G. Tavares, M. Wiecek, Labeling algorithms for multiple objective integer knapsack problems, Comput. Oper. Res. 37 (2010) 700–711.

[15] M. Visée, J. Teghem, M. Pirlot, E.L. Ulungu, Two-phases Method and branch and bound procedures to solve the bi-objective knapsack problem, J. Glob. Optim. 12 (1998) 139–155.
[16] M.E. Captivo, J. Clímaco, J. Figueira, E. Martins, J.L. Santos, Solving bicriteria 0–1 knapsack problems using a labeling algorithm, Comput. Oper. Res. 30 (2003) 1865–1886.
[17] C. Bazgan, H. Hugot, D. Vanderpooten, Solving efficiently the 0–1 multi-objective knapsack problem, Comput. Oper. Res. 36 (2009) 260–279.
[18] C. Delort, O. Spanjaard, Using bound sets in multiobjective optimization: application to the biobjective binary knapsack problem, Lect. Notes Comput. Sci. 6049 (2010) 253–265.
[19] R. Bellman, Dynamic Programming, University Press, Princeton, NJ, USA, 1957.
[20] C. Gomes da Silva, J. Clímaco, J. Figueira, Geometrical configuration of the Pareto frontier of bi-criteria {0,1}-knapsack problem, Technical Report, University of Coimbra, 2004.
[21] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Maryland, USA, 1978.
[22] R. Andonov, S. Rajopadhye, A sparse knapsack algo-tech-cuit and its synthesis, in: P. Cappello, R. Owens, E. Swartzlander, B. Wah (Eds.), International Conference on Application Specific Array Processors, ASAP'94, IEEE, San Francisco, CA, 1994, pp. 302–313.
[23] R. Andonov, V. Poirriez, S. Rajopadhye, Unbounded knapsack problem: dynamic programming revisited, Eur. J. Oper. Res. 123 (2000) 394–407.