INFORMS Journal on Computing
Articles in Advance, pp. 1–12
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1120.0521

Uncommon Dantzig-Wolfe Reformulation for the Temporal Knapsack Problem

Alberto Caprara
(*deceased*)
Formerly at the University of Bologna, Bologna, Italy

Fabio Furini, Enrico Malaguti
Department of Electronics, Computer Sciences and Systems, University of Bologna, 40136 Bologna, Italy
{fabio.furini@unibo.it, enrico.malaguti@unibo.it}

We study a natural generalization of the knapsack problem, in which each item exists only for a given time interval. One has to select a subset of the items (as in the classical case), guaranteeing that for each time instant, the set of existing selected items has total weight no larger than the knapsack capacity. We focus on the exact solution of the problem, noting that prior to our work, the best method was the straightforward application of a general-purpose solver to the natural integer linear programming formulation. Our results indicate that much better results can be obtained by using the same general-purpose solver to tackle a nonstandard Dantzig-Wolfe reformulation in which subproblems are associated with groups of constraints. This is also interesting because the more natural Dantzig-Wolfe reformulation of single constraints performs extremely poorly in practice.

Key words: temporal knapsack problem; Dantzig-Wolfe reformulation; column generation
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received August 2011; revised March 2012; accepted June 2012. Published online in *Articles in Advance*.

1. Introduction

The well-known *knapsack problem* (KP), being a very basic problem in *integer linear programming* (ILP), has been widely studied from all perspectives, including its practical solution to proven optimality; see, e.g., Kellerer et al. (2004) and Martello and Toth (1990). The same holds for most of its natural variants and generalizations. We were surprised to find out that not much was known on the practical solution of the following generalization of KP, which we encountered in railway service design (Caprara et al. 2011).

We are given a single resource and a set of n *tasks*, the ith having a *profit* p_i, a resource *request* w_i, and being *active* for a time interval $[s_i, t_i)$. The total resource available at any point in time is C. The problem calls for the selection of a subset of tasks of maximum total profit, so that at any point in time, the total resource required by the active tasks selected is at most C. The associated straightforward ILP formulation is the following. Clearly, it is enough to impose that the resource constraint is satisfied at the start time s_i of each task i. Letting $S_j := \{i: s_i \le s_j \text{ and } t_i > s_j\}$ be the set of tasks active at time s_i, and x_i a binary variable equal to 1 if task i is selected, the ILP reads:

$$\max \sum_{i=1}^{n} p_i x_i, \tag{1}$$

$$\text{subject to } \sum_{i \in S_j} w_i x_i \le C, \quad j = 1, \dots, n, \tag{2}$$

$$x_i \in \{0, 1\}, \quad i = 1, \dots, n. \tag{3}$$

This problem has been referred to in the literature as *temporal KP* (TKP) (Bartlett et al. 2005), *resource allocation* (Calinescu et al. 2002), *bandwidth allocation* (Chen et al. 2002), or *unsplittable flow on a line* (Bansal et al. 2006). The KP arises as a special case when all tasks are active for the same time interval, or to put it differently, the problem we consider is the extension of KP in which tasks occupy the knapsack capacity only during the time interval in which they are active. Among the names above we prefer and we will use in this paper, TKP, because it indicates the relationship with KP.

The last one of the names above is because the time dimension may, in fact, be a (one-dimensional) spatial dimension, as it is the case in our railway application. In this case, we are given a railway corridor with m stations $1, \dots, m$ and n railcars, the ith having weight w_i and to be transported from station s_i to station f_i, gaining profit p_i in this case. Moreover, we have a train that travels from station 1 to station m and can carry railcars simultaneously of a total weight of up to C. The problem is to decide which railcars the train carries to maximize the profit, which is clearly a TKP.

The TKP was, in fact, only a part of the much more complex railway design problem that we solved in Caprara et al. (2011), and the TKP instances resulting from the overall instances were fairly easy to solve. On the other hand, we decided to produce larger and more challenging instances and to try alternative solution methods on them. We did this also in light of the well-known observation that for the classical KP, there are methods that widely outperform the textbook approach of solving the natural ILP formulation with a general-purpose solver.

The surprising outcome of our experiments was that the best method, among those we tried, to solve our TKP instances is based on an uncommon way to apply the *Dantzig-Wolfe reformulation* (DWR) to ILP (1)–(3). For general references on DWR, see, e.g., Lübbecke and Desrosiers (2004) and Vanderbeck and Wolsey (2010). The most natural form of DWR for this ILP, though not entirely standard, is to convexify each constraint (2) separately. The associated slave (or column-generation) problem is a classical KP and can be solved within a very short time. This is an indirect way to impose all valid inequalities associated with single knapsack constraints, as opposed to heuristically separate only a subset of them as every advanced general-purpose solver would do. Our preliminary computational experiments with this approach indicated that the *linear programming* (LP) relaxation was too slow to be solved to compensate the LP upper bound improvement due to the convexification. Accordingly, the use of DWR for TKP did not appear promising.

However, this changed once we attempted to convexify not single but *multiple* constraints, i.e., to partition the set of constraints into groups and to solve a slave/column-generation problem for each group. For TKP, it seems natural to put into groups constraints associated with consecutive points in time. Along with the fact that it also seems natural to form groups of (about) the same size; say, k constraints per group, the unique real degree of freedom remains the value of k. The resulting slave/column generation is nothing but a smaller TKP, which considers only the tasks that are active in a given time window. When k is not very small, the best way to tackle it is to use a general-purpose ILP solver.

Our finding was that the time required to solve the LP relaxation of the reformulated problem was not, as we initially expected, an increasing function of k, but, in fact, a *unimodal* function of k. In other words, the time required for the LP relaxation is first *decreasing* and then increasing as k increases. Jointly with the fact that the integrality gap is decreasing with k (this is guaranteed if we double k and form new groups by merging pairs of previous groups), this implies that there is an optimal trade-off value of k, which is indeed fairly large, around 100 for our instances. For a choice of k close to this value, a rudimentary depth-first branch-and-price algorithm applied to the reformulation outperforms a state-of-the-art ILP solver applied to the original ILP formulation. (The same state-of-the-art ILP solver is, in fact, used for the slave/column-generation problems in our method.)

These results inspired the application of the same idea to generic mixed-ILPs discussed in Bergner et al. (2011) and Furini (2012), where part of the results of this work are reported. In this case, there are two fundamental differences regarding TKP. First, there is the major issue to decide how to form groups of constraints, possibly leaving some of the constraints in the master problem. Second, the outcome is that the LP relaxations yield good bounds, but generally a branch-and-price algorithm based on them is not competitive in terms of time to prove optimality. This makes the focus of Bergner et al. (2011) entirely different from the one of the present paper.

A subset of the instances considered in this paper were also used in Gamrath and Lübbecke (2010) to test a generic solver for DWR. In that paper, however, the focus is on the effective integration of the generic solver within the general purpose open-source solver SCIP (2011), without discussing how to form groups of constraints.

This paper is organized as follows. We conclude the introduction with a short literature review on TKP. Then in §2, we illustrate the use of DWR for the problem. In §3, we present the three methods that we considered to find a provably optimal solution, and in §4, we report the associated computational results.

It is elementary to observe that everything in the paper also applies to what may be called *temporal ILPs*, which are defined starting from a regular *static* ILP and letting each variable x_i be active for a time interval $[s_i, t_i)$. A feasible solution is an assignment of values $\bar{x}_i$ to each variable x_i such that, at any point in time, setting each variable x_i to $\bar{x}_i$ if it is active, and to 0 otherwise, yields a feasible solution of the associated *static* ILP. We restrict attention to TKP here because it is the simplest case of temporal ILP and also, to the best of our knowledge, the only one that has been studied in the optimization literature.

1.1. Literature Review

Almost all papers on TKP in the literature addressed theoretical aspects of the problem. However, papers on the practical solution of generalizations of the problem also exist.

Theoretical Results. Arkin and Silverberg (1987) consider the problem of scheduling jobs with predefined starting and ending times on C identical machines, by maximizing the value of the completed jobs, where each machine can process one job at a

time. This corresponds to a TKP in which all the requests are identical: in this case, the problem is trivially polynomially solvable because the constraint matrix of the ILP models (1)–(3) is totally unimodular. Arkin and Silverberg (1987) give an $O(n^2 \log n)$ algorithm. They also prove that the variant in which each job can be processed only by a subset of machines is NP-hard, and they give an $O(n^{C+1})$ algorithm for the case of a fixed number of machines C.

Chen et al. (2002) sketch a dynamic programming approach, which is closely related with the $O(n^{C+1})$ algorithm in Arkin and Silverberg (1987). In addition, they propose a polynomial-time approximation algorithm for the special case of TKP in which the profit of a task i is proportional to $(t_i - s_i)w_i$.

Calinescu et al. (2002) show a randomized polynomial-time approximation algorithm for TKP with guarantee arbitrarily close to 2.

Bansal et al. (2006) show that the restricted case of TKP in which the task requests and the amount of resource available are integers in $[0, L]$ and $L \leq 2^{\text{polylog}(n)}$ has a polynomial-time approximation scheme. The results hold for the generalization of TKP in which the resource available is not necessarily a constant C but may vary over time.

Darmann et al. (2010) discuss some generalizations of the problem, review complexity results for special cases of TKP, and prove that the problem remains NP-hard for the case in which all the tasks' profits are identical. In addition, they give a deterministic polynomial-time approximation algorithm with guarantee arbitrarily close to 2 for the special case in which the tasks' start and end times define a proper interval graph (i.e., no interval of activity of a task is contained in another interval of activity).

Finally, the *strong* NP-hardness of TKP was proved by Bonsma et al. (2011), answering the most relevant open question on the problem complexity. The existence of a polynomial-time approximation scheme remains open.

Practical Solution. The only paper on the practical solution of TKP in the literature appears to be the one of Bartlett et al. (2005), who propose an algorithm that integrates tree search techniques with cut generation. The main feature of the approach is to recursively decompose the problem into independent subproblems, performing a branch-and-bound search on an and/or tree. The algorithm is experimentally tested on instances with up to 600 tasks and compared with the solution of models (1)–(3) by means of the general-purpose solver CPLEX 8.1, concluding that the latter is much faster in practice. (In the paper, the authors discuss the generalization of TKP in which the total resource available is not a constant C but varies over time, but this is not the case for the instances they solve.) These instances are fairly small compared to the ones that we can solve with our method, which are one order of magnitude larger; namely, contain up to about 9,000 tasks.

There are other papers in the literature considering generalizations of TKP, which appear to be much harder to solve—compare the sizes mentioned below with those mentioned in the previous paragraph—and not suited for our DWR approach. Hall and Magazine (1994) consider a generalization of TKP in which the starting time of each task can vary in a specified interval (its duration being fixed as in TKP). This models the scheduling of tasks during a space mission, where each task can take place only during a specified time interval. They propose some heuristic algorithms and upper bounds, which are integrated in an exact approach, and tested on a set of randomly generated instances with 50–200 tasks. The proposed exact approach can consistently solve instances with up to 50 tasks. In addition, the most famous generalization of TKP is the *multidimensional KP*, which is nothing but a 0-1 ILP with positive coefficients and "$\leq$" constraints, and contains extremely difficult problems such as *set packing* as special cases. Reviewing the entire state of the art on multidimensional KP is out of the scope of this paper. We simply mention that the current best way to solve the instances in the literature appears to be the application of a general-purpose solver, possibly driving the branch-and-bound search in a proper way, see Puchinger et al. (2010), leading to the optimal solution for up to 250 variables and 5 constraints.

2. DWRs

First of all, let us slightly change the ILP formulation (1)–(3) to get rid of dominated constraints (2) associated with a set S_j such that $S_j \subset S_k$ for some index k. Letting $N := \{j\colon S_j \backslash S_k \neq \varnothing \text{ for all } k \neq j\}$ be the set of indices of the nondominated constraints, we will work with the ILP:

$$\max \sum_{i=1}^{n} p_i x_i, \tag{4}$$

$$\text{subject to} \sum_{i \in S_j} w_i x_i \leq C, \quad j \in N, \tag{5}$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n. \tag{6}$$

The textbook approach to TKP is to solve ILP (4)–(6) directly with a general-purpose solver, possibly relying on the latter for the addition of valid inequalities to strengthen the bounds. All other approaches that come to mind to a practitioner are tentative generalizations of the solution methods for the classical KP. In this respect, even considering just the DWR of single constraints (2) appears to be an original way to approach the problem, though certainly based on an extremely well-known tool.

2.1. DWR of Single Constraints

Let

$$\mathcal{P}_j := \operatorname{conv}\left\{x \in \{0,1\}^{S_j} : \sum_{i \in S_j} w_i x_i \le C\right\}$$

denote the convex hull of the feasible 0-1 solutions to constraint j in (5), restricted to the space of the variables with positive coefficient in the constraint. Moreover, let $\mathcal{V}_j$ denote the set of vertices of $\mathcal{P}_j$, and for convenience, for $v \in \mathcal{V}_j$, let v_i denote the component of v associated with task i. The DWR of constraint j in (5) amounts to imposing that the projection of x onto the components in S_j is a convex combination of the vertices in $\mathcal{V}_j$. This is achieved by introducing variables y_v^j for $v \in \mathcal{V}_j$ denoting the coefficients in the convex combination. Applying this to all constraints in (5), which are then removed, yields the reformulation:

$$\max \sum_{i=1}^{n} p_i x_i, \tag{7}$$

$$\text{subject to } x_i = \sum_{v \in \mathcal{V}_j} v_i y_v^j, \quad j \in N,\ i \in S_j, \tag{8}$$

$$\sum_{v \in \mathcal{V}_j} y_v^j = 1, \quad j \in N, \tag{9}$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n, \tag{10}$$

$$y_v^j \ge 0, \quad j \in N,\ v \in \mathcal{V}_j, \tag{11}$$

where the reformulated constraints have been removed, as they are no longer necessary. Note that one could also remove the original variables, by appropriately defining the profits of the new ones. However, our experiments showed that it is better to keep them, working with an *explicit master*, as it was called in de Aragao and Uchoa (2003). The effect of the reformulation of constraint j is to implicitly impose all the linear constraints valid for $\mathcal{P}_j$ to the LP relaxation of the original formulation (4)–(6). This is at the price of introducing the $O(2^{|S_j|})$ variables y_v^j.

The LP relaxation of (7)–(11) can be solved by column generation as follows. Consider its dual, letting α_i^j and β_j denote, respectively, the dual variables associated with constraints (8) and (9). The dual constraints associated with variables y_v^j are:

$$\beta_j - \sum_{i \in S_j} v_i \alpha_i^j \ge 0, \quad j \in N,\ v \in \mathcal{V}_j. \tag{12}$$

Given values $\bar{\alpha}_i^j$ and $\bar{\beta}_j$ for these dual variables, the separation of (12), which corresponds to the generation of y_v^j variables with positive reduced cost, calls for the optimization of a linear objective function over $\mathcal{P}_j$. This is a classical KP, that can be solved by very effective ad hoc methods without resorting to general-purpose ILP solvers. In fact, the time required to generate columns is negligible for this approach: the bad performances anticipated in the introduction are due to the very high number of iterations required for convergence combined with the fact that the LP relaxation quickly gets slow to solve as variables are added to it.

2.2. DWR of Multiple Constraints

Here, we come to the successful approach that is the heart of this paper. By reasoning as for single constraints, one may reformulate multiple constraints. Formally, for $Q \subseteq N$, let

$$\mathcal{P}_Q := \operatorname{conv}\left\{x \in \{0,1\}^{\bigcup_{j \in Q} S_j} : \sum_{i \in S_j} w_i x_i \le C,\ j \in Q\right\}$$

denote the convex hull of the feasible 0-1 solutions to the constraints in Q among (5), $\mathcal{V}_Q$ the set of vertices of $\mathcal{P}_Q$, and y_v^Q for $v \in \mathcal{V}_Q$ the coefficients in a convex combination of the vertices in $\mathcal{V}_Q$. One may apply separately the reformulation to any collection $\mathcal{Q}$ of subsets of N. Assuming for simplicity that $\mathcal{Q}$ covers N (i.e., each $j \in N$ belongs to some $Q \in \mathcal{Q}$), this leads to the overall reformulation of TKP:

$$\max \sum_{i=1}^{n} p_i x_i, \tag{13}$$

$$\text{subject to } x_i = \sum_{v \in \mathcal{V}_Q} v_i y_v^Q, \quad Q \in \mathcal{Q},\ i \in \bigcup_{j \in Q} S_j, \tag{14}$$

$$\sum_{v \in \mathcal{V}_Q} y_v^Q = 1, \quad Q \in \mathcal{Q}, \tag{15}$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n, \tag{16}$$

$$y_v^Q \ge 0, \quad Q \in \mathcal{Q},\ v \in \mathcal{V}_Q. \tag{17}$$

A major difference regarding DWR for general ILPs discussed in Bergner et al. (2011) is that for TKP, all constraints have the same form and that each variable has a positive coefficient only for a consecutive set of constraints. Accordingly, for TKP, it appears to be natural to (i) let $\mathcal{Q}$ be a partition of N into pairwise disjoint subsets, hereafter called *blocks*, of (ii) almost the same size, each (iii) containing a consecutive set of constraints. We have tested a few variants of this without practical improvements, although, in principle, there are pathological cases in which, even if the cardinality of all blocks is fixed, the best LP bound is obtained by violating (iii).

In a sense, our method is a divide-and-conquer approach: column generation of variables y_v^Q associated with block Q calls for the optimization of a linear objective function over $\mathcal{P}_Q$, which is nothing but a smaller TKP. The difficulty of this problem tends to increase with the size $|Q|$ of the block, the case of $|Q| = 1$ being the reformulation of a single constraint discussed above, and the case of $Q = N$

being the original TKP (with a slightly different objective function). Also, the strength of the LP relaxation bound tends to increase with $|Q|$. On the other hand, the number of times that column generation is called within the solution of the LP relaxation tends to decrease with $|Q|$. A clear indication of our study is that there is an "optimal" *trade-off* value for $|Q|$ (assuming all blocks $\mathcal{Q}$ have the same size, see previously) leading to much smaller LP solution times regarding the two extremes $|Q| = 1$ and $Q = N$, and to tight enough LP bounds to ensure quick solution by using a very basic branch-and-price algorithm. When $|Q|$ is not very small (as is the case for the trade-off above), the best option to solve the column-generation problem appears to be the use of a general-purpose ILP solver applied to the natural formulation (4)–(6) (with N replaced by Q).

2.3. DWR and Cut Generation

It is well known and easy to check that reformulating the constraints in block Q, by introducing the y_v^Q variables and generating them, is equivalent to separating all valid inequalities for $\mathcal{P}_Q$ without introducing further variables. Formally, given values $\bar{x}_i$ for the original variables, testing if there is a valid inequality for $\mathcal{P}_Q$ violated by $(\bar{x}_i)$ is the same as testing if $(\bar{x}_i)$ is a convex combination of the vertices in $\mathcal{V}_Q$, i.e., if there is a solution (y_v^Q) to:

$$\bar{x}_i = \sum_{v \in \mathcal{V}_Q} v_i y_v^Q, \quad i \in \bigcup_{j \in Q} S_j, \tag{18}$$

$$\sum_{v \in \mathcal{V}_Q} y_v^Q = 1, \tag{19}$$

$$y_v^Q \geq 0, \quad v \in \mathcal{V}_Q. \tag{20}$$

Adding the dummy objective function $\min 0$, this system can be solved by column generation as discussed above. If it is infeasible, an optimal dual ray yields a violated inequality. This can also be interpreted as the Benders decomposition of the constraints involving the y_v^Q variables in (13)–(17).

It is then natural to check if the direct separation of valid inequalities for $\mathcal{P}_Q$ is computationally competitive with the DWR. Our computational results clearly indicate that this is absolutely not the case. We refer the reader to, e.g., Ralphs and Galati (2006) and Sherali et al. (2005) for further details on cut generation for ILPs.

3. Exact Solution Methods

In this section, we illustrate the three methods we used to solve TKP to proven optimality.

3.1. General-Purpose ILP Solver

The first method is the direct application of a general-purpose ILP solver to the basic formulation (4)–(6).

3.2. Branch and Price Based on DWR

As already mentioned, the exact method based on reformulation (13)–(17) is an elementary branch-and-price algorithm that solves at each node the LP relaxation by column generation, branches on the original x_i variable whose value is closest to 1 in the LP solution, and imposes the bound on x_i in all the slaves in which this variable appears. The column-generation subproblems are solved by a general-purpose ILP solver with default parameters; we tried different options and tricks to speed it up without achieving any noticeable savings. The master LP initially contains all the x variables, and for each block $Q \in \mathcal{Q}$, the variable y_0^Q associated with the null vector. This means that the unique initial solution of the master LP is to select no task. No primal heuristic is used by our method, all integer solutions are obtained by branching and solving LP relaxations.

The solution times of the LP relaxation are greatly reduced by replacing the "=" by "≤" in constraints (14), which leads to an equivalent formulation, also in terms of the LP relaxation, since the set of TKP solutions forms an independence system, i.e., given any feasible subset T of tasks, every $T' \subset T$ is also feasible. The main reason for the time reduction is the well-known fact that imposing nonnegativity on the dual variables associated with (14) ensures that the dual solutions are much more stable within the column-generation process.

3.3. Dynamic Programming

We next illustrate a simple alternative exact solution method, which turns out to be viable when the number of feasible solutions to the KP associated with constraint j among (5), neglecting the tasks not in S_j, is not too large. (Or, equivalently, when the number $|\mathcal{V}_j|$ of vertices of $\mathcal{P}_j$ as defined in §2.1 is not too large.) This simple approach turns out to be the best way to solve some of the instances in our test bed.

The method is a simple dynamic programming scheme that computes the maximum profit path in a directed acyclic graph in which nodes are partitioned into layers, one for each constraint $j \in N$. The associated layer contains $O(2^{|S_j|})$ nodes, one for each vertex in $\mathcal{V}_j$, corresponding to a subset $T \subseteq S_j$ such that $\sum_{i \in T} w_i \leq C$. Arcs connect only nodes associated with consecutive layers. Specifically, letting $\nu(j)$ denote the index after j in N, there is an arc (T, U) for $T \subseteq S_j$ and $U \subseteq S_{\nu(j)}$ if $T \cap S_{\nu(j)} = U \cap S_j$, i.e., the set of tasks active in both constraints S_j and $S_{\nu(j)}$ are the same in T and U. The profit associated with arc (T, U) is given by $\sum_{i \in U \setminus S_j} p_i$, i.e., it is the sum of the profits of the tasks in U that become active in constraint $S_{\nu(j)}$.

It is easy to check that by adding a dummy source σ connected to all nodes T in the first layer with arcs (σ, T) of profit $\sum_{i \in T} p_i$, and a dummy sink τ connected to all nodes in the last layer with arcs (σ, T) of

profit 0, the determination of a maximum profit path from σ to τ yields an optimal TKP solution of the same profit, in which the tasks selected are given by the union of the subsets visited by the path. A simple implementation using a forward search, in which the graph arcs need not be stored explicitly, has time complexity $O(\sum_{j\in N} 2^{|S_j|}\cdot 2^{|S_{\nu(j)}\setminus S_j|})$ and space complexity $O(\sum_{j\in N} 2^{|S_j|})$.

It is worth mentioning that the straightforward generalization of the classical dynamic programming scheme for KP, with one state entry for the capacity occupied in each constraint in N, was never competitive for the instances we considered, having time and space complexity $O(n(C+1)^{|N|})$.

4. Computational Results

In this section, we report our computational experiments with the solution methods discussed in the paper. All computational tests were conducted by using 1 core of an INTEL Core2 Duo E6550 at 2.33 GHz with 2 GB of RAM under Windows XP. The general-purpose solver used was IBM-CPLEX 12.1 (2011), with the default parameter setting, which turned out to be a good choice for the instances we considered (we tried a few other settings without improvements). In all experiments, the time limit was set to one hour CPU time. Average times are always computed by including the time limits. When the time limit is reached at least once, we indicate by (*TL*) the number of times this happens. In the last row denoted by *avg*, with average of average results, we indicate the total (rather than average) number of time limits.

4.1. Test Instances

As already mentioned in §1, our interest toward TKP was motivated by an application in railway service design, but the TKP instances coming from the real-world application in Caprara et al. (2011) (which are a wide simplification of the instances of the real-world problem) turn out to be elementary to solve by a general-purpose ILP solver. The same thing happens with the only TKP instances mentioned in the literature, to the best of our knowledge, which are those of Bartlett et al. (2005). We therefore decided to create new, hopefully challenging, TKP instances and to make them publicly available in the hope of stimulating further research on the topic. We created two classes of instances, one by trying to control the number of active tasks per constraint and the other by generalizing in the (apparently) most natural way the classical random test instances of KP. We used the first class to set up our algorithms and tuning the parameters, and the second one to verify the results in a double-blind manner. All instances are publicly available at http://www.or.deis.unibo.it/research.html, with detailed computational results for the single instances, while in the following, we report average results. See also the online supplement to this paper (available at http://dx.doi.org/10.1287/ijoc.1120.0521).

First Class. The generation of the first class of instances has the following input parameters. The number $|N|$ of constraints in the formulation (4)–(6) is specified on input (and the generation guarantees that none of these constraints is dominated). The tasks are generated by considering the constraints one after the other. The number of tasks active in each constraint is uniformly distributed in $[a_{\min}, a_{\max}]$. If the constraint is not the first one, among these active tasks, a percentage uniformly distributed in $[b_{\min}, b_{\max}]$ is selected uniformly from the tasks that were already active in the previous constraint, whereas the other tasks become active in the new constraint. Profits and requests are integer values uniformly distributed in $[p_{\min}, p_{\max}]$ and $[w_{\min}, w_{\max}]$, respectively.

We considered 10 groups of instances, called I, ..., X, each with the values of the parameters as reported in Table 1, where the column $p = w$? indicates whether profit and requests are the same (this is the case only for Group VII). The last four columns report, respectively, the minimum and maximum number of variables and constraints in the formula-

Table 1 Parameter Values Used to Generate the Test Instances in the First Class

Group	$a_{\min}$	$a_{\max}$	$b_{\min}$	$b_{\max}$	$p_{\min}$	$p_{\max}$	$w_{\min}$	$w_{\max}$	$p = w$?	$n_{\min}$	$n_{\max}$	$\lvert N\rvert_{\min}$	$\lvert N\rvert_{\max}$
I	10	10	90	95	10	100	10	100	No	2,697	3,849	2,688	3,840
II	15	15	90	95	10	100	10	100	No	4,480	6,462	2,688	3,840
III	20	20	90	95	10	100	10	100	No	4,972	7,026	2,688	3,840
IV	25	25	90	95	10	100	10	100	No	6,322	8,977	2,688	3,840
V	30	30	90	95	10	100	10	100	No	2,071	5,129	768	1,920
VI	30	30	70	90	10	100	10	100	No	4,948	12,498	768	1,920
VII	30	30	90	95	10	100	10	100	Yes	2,071	5,142	768	1,920
VIII	25	35	90	95	10	100	10	100	No	2,916	7,241	768	1,920
IX	25	35	70	90	10	100	10	100	No	5,210	13,025	768	1,920
X	30	40	90	95	10	100	10	100	No	3,117	7,709	768	1,920

tion (4)–(6). We generated 10 instances for each group, which differ from each other by the number $|N|$ of constraints. Although instances within each group have a different size, average values reported in the computational experiments allow a meaningful comparison among different groups. For Groups I–IV, $|N| = 2{,}688 + 128(i-1)$ for instances $i = 1, \ldots, 10$. For Groups V–X, $|N| = 768 + 128(i-1)$ for instances $i = 1, \ldots, 10$.

Second Class. For the second class of instances, we considered the way in which the most common class of KP instances; namely, the so-called *uncorrelated problems* (Kellerer et al. 2004, Martello and Toth 1990) are generated. For KP, one simply has to generate p_i and w_i, which are both uniformly random in $[1, p_{\max}]$ in this case. In our case, we also have to generate the task interval $[s_i, t_i)$. To this end, we let s_i be uniformly random in $[1, s_{\max}]$ and the duration $t_i - s_i$ be uniformly random in $[1, d_{\max}]$. Finally, the capacity is set to $\alpha \sum_{i=1}^{n} w_i$.

In the classical KP case, typical values for $p_{\max}$ and α are 100 and 0.5, respectively. In our case, we also set $p_{\max} = 100$. In addition, we set $s_{\max} = 1{,}000$ and considered 10 groups of instances, each with $n = 1{,}000$ tasks, with $d_{\max} \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. The resulting 10 groups are called XI, ..., XX. To obtain challenging instances for this choice of parameters, we set $\alpha = 0.01$ (for larger values of α, a wide fraction of the tasks can be selected in the solution, and the instances tend to be easy). By construction, all these instances have roughly the same number $|N|$ of constraints in the formulation (4)–(6), around 450.

4.2. Setting the Size of the Groups in DWR

First of all, a key issue to address for the approach based on DWR is the number $|Q|$ of constraints in each block. We present the effect of using values that are integer powers of 2 for all the instances in the first class, presenting average values for each of the 10 Groups I–X. The results for the instances in the second class are entirely similar and not reported here. In Table 2, for $|Q| \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$, we report the time needed to solve the LP relaxation of the DWR (13)–(17), the total time needed to solve the LP relaxation of the master problem during the procedure, the number of columns generated, and the associated percentage integrality gap regarding the optimal TKP solution value. The integrality gaps for the instances reaching the time limit refer to the Lagrangian upper bound on the optimal LP value, obtained by summing to the current master LP value the violation of the dual constraints.

The table points out that by increasing the block size $|Q|$, the computing time needed to solve the LP relaxation initially decreases. This is because the decreasing number of iterations of column generation compensates, up to a certain point, the fact that generating columns becomes harder. Specifically, for $|Q| = 32$ and $|Q| = 64$, the computing time is one order of magnitude smaller than for $|Q| = 1$, but also one or more orders of magnitude smaller than for $|Q| = 256$. On the other hand, the integrality gap is already very small for $|Q| = 32$ and $|Q| = 64$. The advantage of reformulating multiple constraints, for a correct choice of the block size, is therefore extremely evident. Also clear is that this choice is *robust*, in that there is no magic value but a fairly wide range around the value for which the minimum LP solution time is attained. Note that solution time for the master LP is fairly small; namely, the solution time is widely dominated by the time needed to generate the columns.

In Figure 1, we give a graphical representation of the results in Table 2, plus some additional information. Parts (a)–(d) are in logarithmic scale. Part (a) shows the total time to solve the LP relaxation of the DWR (13)–(17) (column *time* in Table 2). Part (b) shows the total time needed to solve the LP relaxation of the master problem during the procedure (*timeLP*). Part (c) shows the total time needed to solve the ILPs associated with the slave problems for column generation. Part (d) shows the number of columns generated (*nCols*). Part (e) shows the percentage integrality gap regarding the optimal TKP solution value (*gap*). Finally, Part (f) shows the number of x variables present in more than one block, which affects the number of constraints (14).

In the following, we will present results only for $|Q| \in \{32, 64, 128\}$, which are by far the three best choices for our instances (restricting attention to powers of 2). Without reporting the associated results, we let the reader imagine how bad a branch-and-price approach based on DWR of single constraints may behave in view of Table 2.

4.3. DWR vs. Cut Generation

In Table 3, we consider the solution of the LP relaxation of the DWR (13)–(17) by the Benders decomposition approach explained in §2.3, which avoids the explicit handling of the y variables and separates valid inequalities for $\mathcal{P}^Q$ in the space of the original variables. As in §4.2, we consider only the instances in the first class, the results for the second class being analogous. For $|Q| \in \{1, 2, 4, 8, 16, 32\}$, we report the same information as in Table 2, the number of columns being replaced by the number of generated cuts. If the time limit is not reached, the integrality gap coincides with the one in Table 2. Otherwise, the optimal value of current LP provides a valid bound.

The results clearly show the huge advantage of DWR regarding the cut generation version, because

Table 2 Comparison of the LP Relaxations of (13)–(17) Solved with Column Generation for Various Values of the Block Size $|Q|$

	$\|Q\| = 1$				$\|Q\| = 2$				$\|Q\| = 4$			
Group	*time*	*timeLp*	*nCols*	*gap*	*time*	*timeLp*	*nCols*	*gap*	*time*	*timeLp*	*nCols*	*gap*
I	357.9	23.6	39,469	0.99	103.2	4.6	18,099	0.74	38.1	1.0	8,268	0.42
II	653.0	97.4	56,444	1.36	181.1	12.2	25,061	0.87	84.4	2.0	10,737	0.46
III	1,348.5	259.1	81,935	1.58	302.6	44.7	34,898	1.04	127.3	5.8	14,379	0.55
IV	1,546.1	403.8	97,651	1.85	491.8	79.3	41,542	1.19	196.6	9.4	16,861	0.61
V	612.0	183.4	47,161	2.13	198.7	34.2	20,317	1.40	89.8	3.9	8,068	0.72
VI	351.4	38.9	29,718	1.35	131.5	4.5	12,326	0.62	83.7	1.1	4,895	0.26
VII	207.3	21.2	23,338	0.20	70.0	2.8	10,220	0.13	39.5	0.7	4,338	0.08
VIII	362.7	82.2	35,121	1.43	144.2	15.4	15,772	0.95	72.4	1.7	6,490	0.47
IX	275.5	26.7	26,725	1.08	119.8	3.8	11,638	0.54	78.6	1.1	4,838	0.25
X	543.6	134.4	42,437	1.57	208.0	28.6	19,130	1.03	108.1	3.8	7,738	0.49
Avg	625.8	127.1	48,000	1.35	195.1	23.0	20,900	0.85	91.9	3.1	8,661	0.43

	$\|Q\| = 8$				$\|Q\| = 16$				$\|Q\| = 32$			
Group	*time*	*timeLp*	*nCols*	*gap*	*time*	*timeLp*	*nCols*	*gap*	*time*	*timeLp*	*nCols*	*gap*
I	24.2	0.4	3,640	0.19	14.9	0.2	1,608	0.08	11.0	0.1	771	0.04
II	54.1	0.7	4,495	0.21	33.4	0.3	1,976	0.10	25.7	0.2	911	0.05
III	81.1	1.0	5,702	0.23	50.9	0.4	2,428	0.11	40.0	0.3	1,064	0.06
IV	141.9	1.5	6,519	0.27	79.0	0.6	2,720	0.12	72.8	0.4	1,265	0.07
V	54.1	0.7	3,102	0.32	33.8	0.3	1,278	0.12	37.9	0.2	564	0.05
VI	63.0	0.5	2,100	0.11	68.0	0.3	985	0.05	147.0	0.3	485	0.03
VII	22.4	0.3	1,691	0.03	13.5	0.1	751	0.02	10.7	0.1	353	0.00
VIII	44.8	0.5	2,546	0.18	32.6	0.3	1,077	0.09	38.9	0.1	518	0.04
IX	58.1	0.5	2,080	0.10	61.7	0.3	975	0.05	119.4	0.3	477	0.02
X	60.4	0.7	3,024	0.21	44.1	0.3	1,253	0.08	59.9	0.2	601	0.04
Avg	60.4	0.7	3,490	0.19	43.2	0.3	1,505	0.08	56.3	0.2	701	0.04

	$\|Q\| = 64$				$\|Q\| = 128$				$\|Q\| = 256$			
Group	*time*	*timeLp*	*nCols*	*gap*	*time* (*TL*)	*timeLp*	*nCols*	*gap*	*time* (*TL*)	*timeLp*	*nCols*	*gap*
I	11.1	0.0	369	0.02	11.9 (0)	0.1	184	0.02	12.6 (0)	0.0	87	0.00
II	34.2	0.1	442	0.02	39.9 (0)	0.1	227	0.02	57.5 (0)	0.1	106	0.01
III	58.8	0.2	547	0.02	79.1 (0)	0.2	264	0.01	164.8 (0)	0.1	128	0.01
IV	92.7	0.2	606	0.04	183.8 (0)	0.2	289	0.01	656.1 (0)	0.1	146	0.00
V	64.1	0.1	257	0.02	423.7 (0)	0.1	122	0.01	2,040.4 (4)	0.1	57	0.00
VI	321.7	0.2	241	0.01	179.3 (0)	0.2	121	0.01	1,040.9 (0)	0.1	33	0.14
VII	12.8	0.1	172	0.00	1,133.9 (1)	0.1	85	0.00	3,350.9 (9)	0.0	45	0.00
VIII	91.5	0.1	252	0.02	14.8 (0)	0.1	117	0.01	18.2 (0)	0.1	49	0.00
IX	210.3	0.2	226	0.01	168.6 (0)	0.2	110	0.01	599.9 (0)	0.1	44	0.03
X	127.3	0.2	307	0.02	441.5 (0)	0.1	144	0.01	1,931.8 (3)	0.1	53	0.06
Avg	102.4	0.2	342	0.02	267.7 (1)	0.1	166	0.01	987.3 (16)	0.1	75	0.03

the separation of valid inequalities requires in itself the generation of columns. Note that in this case, the average time increases with $|Q|$. For the *good* choices of Q indicated in §4.2, the cut generation almost always reaches the time limit, and the associated upper bound is of very bad quality.

4.4. DWR and Branch and Price vs. a General-Purpose Solver

We now illustrate the results of the first two methods of §3 for instances in Groups V–X in the first class, and for the instances in the second class, all of which cannot be solved by dynamic programming, the third method of §3, as the average number of active tasks per constraint is too large.

For the first class, in Table 4, we compare the solution times and the integrality gaps of the following LP relaxations of TKP: the basic one, associated with the formulation (4)–(6), the one obtained at the root node by CPLEX applied to the basic formulation (with the addition of the default CPLEX cutting planes), and finally, the one obtained with the reformulation (13)–(17) considering $|Q| \in \{32, 64, 128\}$ (according to §4.2). Table 4 clearly shows that for the block sizes $|Q|$ considered the optimality gap of DWR with respect to the CPLEX root is reduced by about two orders of magnitude at the cost of increasing the running time by one to two orders of magnitude.

Table 5 shows that the considerable increase in the root node solution time from CPLEX to DWR indeed

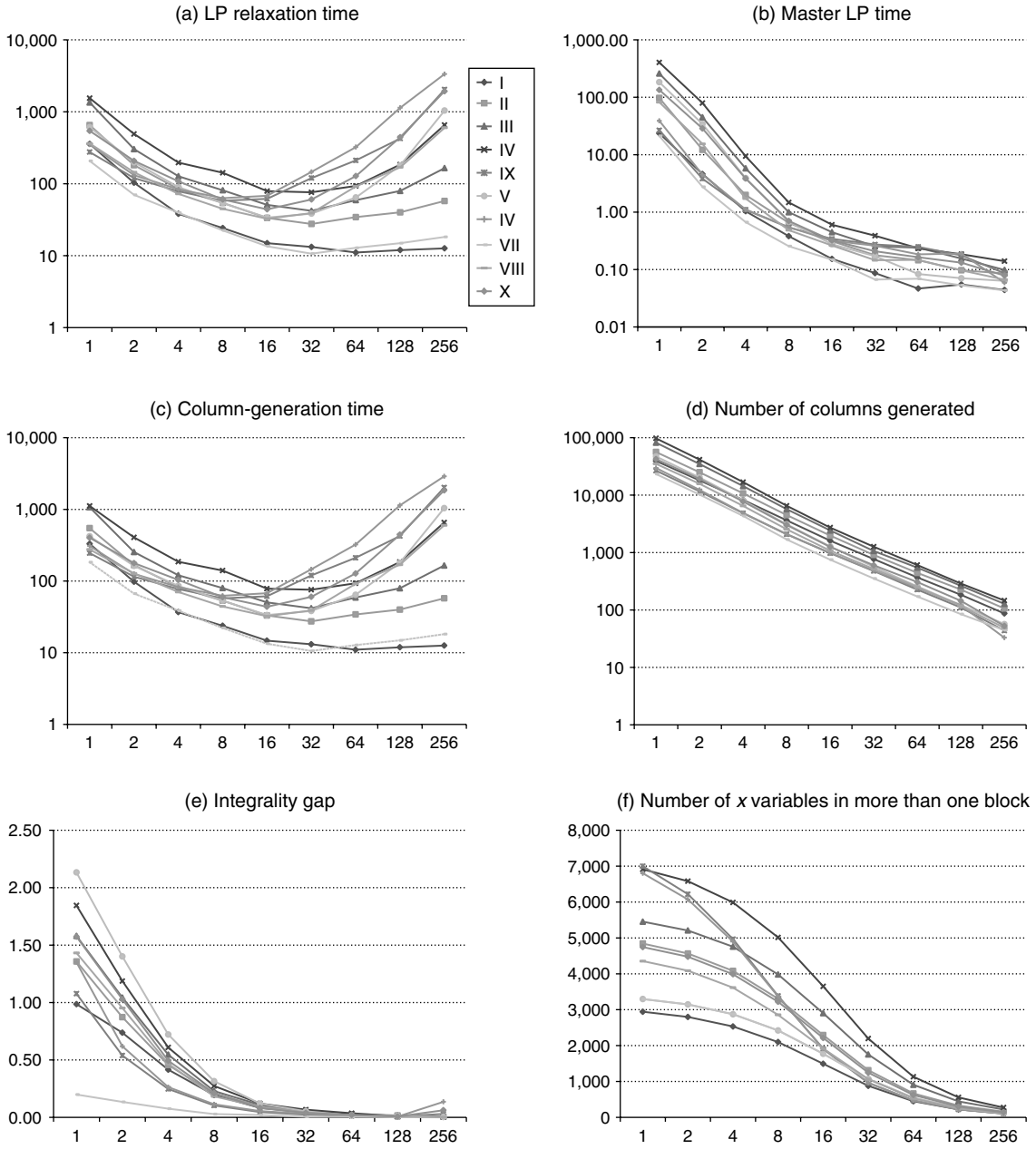

Figure 1 **Comparison of the LP Relaxations of (13)–(17) Solved with Column Generation for Various Values of the Block Size $|Q|$**

pays off when it comes to find an optimal solution by branch and bound, with a significant reduction in the number of branching nodes. Except from Group VI, for which the fact that profits and requests coincide makes the instances easy, CPLEX is unable to solve a single instance out of the 50 in the other groups within the one-hour time limit. On the other hand, DWR solves to optimality 42 of these instances for $|Q| = 32$, 47 for $|Q| = 64$, and 43 for $|Q| = 128$. The average running time for the instances solved by DWR is a few minutes. The optimality gaps are reported only if a method reaches the time limit. When also DWR reaches the time limit, its optimality gap is one order of magnitude smaller than for CPLEX. In the following tables, we highlight in bold the best exact method, i.e., the one reaching the time limit the smallest number of times breaking ties by lowest average computing time.

The results for the instances of the second class are reported in Tables 6 and 7, which have the same format as Tables 4 and 5, respectively. In this case, for DWR, $|Q| = 128$ is slightly better than $|Q| = 64$ and $|Q| = 32$. In addition, we restrict attention to the 50 instances in Groups XIII–XVII, as the instances in Groups XI and XII are very easy, and none of the instances in Groups XVIII–XX could be

Table 3 Comparison of the LP Relaxations of (13)–(17) Solved with Cut Generation for Various Values of the Block Size $|Q|$

	$\|Q\| = 1$				$\|Q\| = 2$				$\|Q\| = 4$			
Group	*time*	*timeLp*	*nCuts*	*gap*	*time*	*timeLp*	*nCuts*	*gap*	*time*	*timeLp*	*nCuts*	*gap*
I	146.1	0.9	13,748	0.99	63.9	0.7	8,545	0.74	42.3	0.6	5,319	0.42
II	284.3	3.0	22,057	1.36	151.3	2.3	13,911	0.87	136.0	2.0	8,773	0.46
III	606.4	7.4	28,775	1.58	261.7	5.4	18,279	1.04	252.9	4.2	11,316	0.55
IV	621.2	14.3	33,994	1.85	458.4	11.4	23,229	1.19	522.7	8.6	14,427	0.61
V	345.1	11.6	17,713	2.13	248.3	7.8	11,453	1.40	344.6	5.9	7,148	0.72
VI	464.0	12.5	23,531	1.35	692.9	10.6	15,227	0.62	2,528.1	11.9	10,120	0.26
VII	224.9	4.6	17,692	0.20	148.4	3.6	11,494	0.13	163.0	2.6	7,298	0.08
VIII	349.6	10.1	18,716	1.43	329.6	7.9	13,004	0.95	666.3	7.4	8,623	0.47
IX	447.7	11.8	23,252	1.08	735.2	10.5	15,913	0.54	2,529.3	10.8	10,175	0.25
X	486.3	17.0	21,393	1.57	487.7	13.8	14,874	1.03	969.1	11.9	9,610	0.49
Avg	397.5	9.3	22,087	1.35	357.7	7.4	14,593	0.85	815.4	6.6	9,281	0.43

	$\|Q\| = 8$				$\|Q\| = 16$				$\|Q\| = 32$			
Group	*time* (*TL*)	*timeLp*	*nCuts*	*gap*	*time* (*TL*)	*timeLp*	*nCuts*	*gap*	*time* (*TL*)	*timeLp*	*nCuts*	*gap*
I	47.9 (0)	0.6	3,528	0.19	130.8 (0)	0.7	2,804	0.08	656.1 (0)	1.0	2,621	0.04
II	266.0 (0)	2.0	6,091	0.21	1,170.1 (0)	2.6	5,150	0.10	3,600.0 (10)	1.6	3,337	0.87
III	526.3 (0)	4.1	7,549	0.23	2,220.6 (0)	5.1	6,107	0.11	3,600.0 (10)	1.5	2,855	1.81
IV	1,426.5 (0)	8.7	9,665	0.27	3,600.0 (10)	5.0	5,673	0.59	3,600.0 (10)	0.9	2,112	4.09
V	1,031.5 (0)	6.3	4,792	0.32	3,551.0 (6)	4.9	3,185	0.34	3,600.0 (10)	0.6	1,159	3.29
VI	3,600.0 (10)	1.8	4,204	1.83	3,600.0 (10)	0.1	532	9.11	3,600.0 (10)	0.2	410	9.77
VII	379.9 (0)	2.2	5,080	0.03	1,752.1 (0)	2.8	4,272	0.02	3,600.0 (10)	1.0	2,749	0.35
VIII	2,717.8 (3)	9.1	6,376	0.19	3,600.0 (10)	1.2	2,473	1.62	3,600.0 (10)	0.1	405	7.93
IX	3,600.0 (10)	1.5	4,280	1.75	3,600.0 (10)	0.1	545	8.22	3,600.0 (10)	0.1	140	10.91
X	3,236.0 (4)	10.7	6,326	0.27	3,600.0 (10)	0.9	2,118	2.26	3,600.0 (10)	0.1	296	8.91
Avg	1,683.2 (27)	4.7	5,789	0.53	2,682.5 (56)	2.3	3,286	2.24	3,305.6 (90)	0.7	1,608	4.80

Table 4 Comparison of LP Relaxations of TKP for the Instances in Groups V–X in the First Class

	Basic model, LP		Basic model, root		DWR, $\|Q\| = 32$, LP		DWR, $\|Q\| = 64$, LP		DWR, $\|Q\| = 128$, LP	
Group	*time*	*gap*	*time*	*gap*	*time*	*gap*	*time*	*gap*	*time*	*gap*
V	0.0	15.67	2.7	2.23	37.9	0.05	64.1	0.02	179.3	0.01
VI	0.1	13.63	5.3	1.52	147.0	0.03	321.7	0.01	1,133.9	0.01
VII	0.0	10.99	1.8	0.25	10.7	0.00	12.8	0.00	14.8	0.00
VIII	0.1	13.21	2.9	1.62	38.9	0.04	91.5	0.02	168.6	0.01
IX	0.1	12.41	4.6	1.23	119.4	0.02	210.3	0.01	423.7	0.01
X	0.1	13.23	3.3	1.94	59.9	0.04	127.3	0.02	441.5	0.01
Avg	0.1	13.19	3.4	1.47	69.0	0.03	137.9	0.01	393.6	0.01

Table 5 Comparison of Exact Methods for TKP for the Instances in Groups V–X in the First Class

	Basic model, B&B			DWR, $\|Q\| = 32$, B&P			DWR, $\|Q\| = 64$, B&P			DWR, $\|Q\| = 128$, B&P		
Group	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*
V	3,600.0 (10)	0.90	414,538	480.0 (0)		170	**176.2 (0)**		36	501.7 (0)		36
VI	3,600.0 (10)	0.75	358,125	1,638.4 (2)	0.1	108	**1,154.3 (0)**		37	1,884.8 (3)	0.01	24
VII	**5.1 (0)**	0.90	520	15.3 (0)		5	17.8 (0)		6	16.0 (0)		1
VIII	3,600.0 (10)	0.56	368,934	1,002.1 (1)	0.1	275	572.2 (0)		72	**304.0 (0)**		10
IX	3,600.0 (10)	0.54	367,807	1,410.4 (3)	0.0	104	1,085.6 (2)	0.03	41	**812.1 (1)**	0.02	9
X	3,600.0 (10)	0.86	371,790	1,270.8 (2)	0.1	195	**911.9 (1)**	0.05	82	1,388.5 (3)	0.02	32
Avg	3,000.8 (50)	0.76	313,619	969.5 (8)	0.1	143	653.0 (3)	0.04	46	817.8 (7)	0.01	19

solved to proven optimality. The results reported in Table 7 show that CPLEX solves all the instances in Group XIII to proven optimality within a time that is almost an order of magnitude smaller than the time taken by DWR to solve the root node, at which it always finds the optimum except in one case. Already for Group XIV, CPLEX solves only half of the instances, whereas DWR solves all of them

Table 6 Comparison of LP Relaxations of TKP for the Instances in the Second Class

	Basic model, LP		Basic model, root		DWR, $\|Q\| = 32$, LP		DWR, $\|Q\| = 64$, LP		DWR, $\|Q\| = 128$, LP	
Group	*time*	*gap*	*time*	*gap*	*time* (*TL*)	*gap*	*time*	*gap*	*time* (*TL*)	*gap*
XIII	0.0	1.79	0.3	0.24	557.2 (0)	0.01	169.7 (0)	0.00	77.8 (0)	0.00
XIV	0.0	1.88	0.4	0.45	1,857.2 (3)	0.02	1,194.5 (1)	0.01	1,414.1 (1)	0.01
XV	0.0	1.81	0.5	0.59	3,067.8 (6)	0.04	2,655.9 (4)	0.03	2,621.5 (5)	0.01
XVI	0.0	1.80	0.6	0.67	3,563.1 (9)	0.12	3,354.3 (8)	0.12	3,351.1 (9)	0.15
XVII	0.0	1.78	0.6	0.76	3,562.3 (9)	0.17	3,600.0 (10)	0.19	3,600.0 (10)	0.20
Avg	0.0	1.81	0.5	0.54	2,521.5 (27)	0.07	2,194.9 (23)	0.07	2,212.9 (25)	0.07

Table 7 Comparison of Exact Methods for TKP for the Instances in the Second Class

	Basic model, B&B			DWR, $\|Q\| = 32$, B&P			DWR, $\|Q\| = 64$, B&P			DWR, $\|Q\| = 128$, B&P		
Group	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*
XIII	**11.2 (0)**		4,101	881.4 (0)		575	211.2 (0)		72	82.8 (0)		3
XIV	2,228.7 (5)	0.08	639,348	2,852.7 (6)	0.0	792	**1,847.7 (2)**	0.03	445	1,907.4 (2)	0.04	123
XV	3,600.0 (10)	0.17	666,381	3,600.0 (10)	0.0	202	3,217.0 (7)	0.04	209	**2,830.2 (6)**	0.02	16
XVI	3,600.0 (10)	0.25	553,044	3,600.0 (10)	0.1	44	3,493.0 (9)	0.13	28	**3,371.7 (9)**	0.16	2
XVII	3,600.0 (10)	0.34	460,084	3,600.0 (10)	0.2	8	3,600.0 (10)	0.19	0	3,600.0 (10)	0.20	0
Avg	2,608.0 (35)	0.21	464,592	2,906.8 (36)	0.1	324	2,473.8 (28)	0.10	151	2,358.4 (27)	0.11	29

Table 8 Comparison of Exact Methods for TKP for the Instances in Groups I–IV of the First Class

	Basic model, B&B			DWR, $\|Q\| = 32$, B&P			DWR, $\|Q\| = 64$, B&P			DWR, $\|Q\| = 128$, B&P		Dyn. prog
Group	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*gap*	*nodes*	*time* (*TL*)	*nodes*	*time* (*TL*)
I	7.6 (0)		658	91.9 (0)		39	24.9 (0)		10	20.2	6	**1.0 (0)**
II	1,988.4 (3)	0.06	182,120	636.6 (0)		69	158.0 (0)		24	128.2	20	**9.5 (0)**
III	3,600.0 (10)	0.40	354,977	1,972.6 (4)	0.08	216	627.2 (0)		82	284.4	22	**182.2 (0)**
IV	3,600.0 (10)	0.93	331,616	2,906.6 (7)	0.07	239	1,091.5 (1)	0.07	106	**433.2**	16	3,600.0 (10)
Avg	2,299.0 (23)	0.46	217,342	1,401.9 (11)	0.08	141	475.4 (1)	0.07	55	216.54	16	948.2 (10)

but one (the two unsolved instances for $|Q| = 128$ and $|Q| = 64$ have only one common element). This trend continues for Groups XV–XVII, with CPLEX reaching the time limit in all cases and DWR solving six instances in Group XV (four with $|Q| = 128$ and two more for $|Q| = 64$), one instance in Group XVI, and no instance in Group XVII. In the unsolved cases, the final DWR gap is generally smaller, sometimes much smaller, than the final CPLEX gap. Differently from the instances in the first class, for these DWR generally reaches the time limit already at the root node.

4.5. Dynamic Programming vs. All for Small Support Constraints

For instances in Groups I–IV in the first class, the dynamic programming method of §3.3 can also be applied, as the average number of active tasks per constraint is sufficiently small. The associated results are reported in Table 8.

Note that when dynamic programming reaches the time limit, there are no feasible solutions nor upper bounds available, and therefore no associated gap. Table 8 shows that dynamic programming is the best method for the instances in Groups I and II, and also for III, being comparable to DWR. On the other hand, it cannot solve any instance in Group IV. As to CPLEX, it is the second best method for Group I but the worst one for Groups II–IV, being unable to solve any of the instances in the last two. DWR with $|Q| = 128$ is the only method capable of solving all the 40 instances within the time limit.

5. Conclusions

Generally, DWR is used when the problem at hand admits a natural decomposition, the prototype example being the well-known bin packing (or cutting stock) problem. In this paper, we have shown the potential of the method in a context in which there is no natural decomposition. As already mentioned in the paper, this outcome for TKP inspired the investigation of the method for general mixed-ILPs, which started with Bergner et al. (2011). However, in terms of exact solution, the TKP case is strikingly successful,

and therefore deserves a separate treatment, given in this paper.

Our computational experiments also showed that DWR combined with column generation widely outperforms the equivalent approach based on cut generation. In passing, we also proposed a dynamic programming algorithm, which turns out to be the best method for solving our TKP instances with small support constraints.

On the other hand, among the instances we proposed, three in the first class and quite a few in the second remained unsolved. We hope that this will stimulate further research on this very natural generalization of KP.

Electronic Companion

An electronic companion to this paper is available as part of the online version at http://dx.doi.org/10.1287/ijoc.1120.0521.

Acknowledgments

The authors are grateful to Marco Lübbecke for illuminating discussions on the subject, and to the associate editor and two anonymous referees for their helpful comments.

References

Arkin EM, Silverberg EB (1987) Scheduling jobs with fixed start and end times. *Discrete Appl. Math.* 18(1):1–8.

Bansal N, Chakrabarti A, Epstein A, Schieber B (2006) A quasi-PTAS for unsplittable flow on line graphs. *Proc.* 38*th Annual ACM Sympos. Theory Comput.* (*STOC, Seattle*), 721–729.

Bartlett M, Frisch AM, Hamadi Y, Miguel I, Tarim SA, Unsworth C (2005) The temporal knapsack problem and its solution. *Proc.* 2*nd Internat. Conf. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optim. Problems* (*CP-AI-OR* 2005) (Springer-Verlag, Berlin), 34–48.

Bergner M, Caprara A, Furini F, Lübbecke M, Malaguti E, Traversi E (2011) Partial convexification of general MIPs by Dantzig-Wolfe reformulation. *Proc.* 15*th Internat. Conf. Integer Programming and Combinatorial Optim.* (*IPCO* 2011) (Springer-Verlag, Berlin), 39–51.

Bonsma P, Schulz J, Wiese A (2011) A constant factor approximation algorithm for unsplittable flow on paths. *CoRR* abs/1102.3643.

Calinescu G, Chakrabarti A, Karloff HJ, Rabani Y (2002) Improved approximation algorithms for resource allocation. *Proc.* 9*th Internat. Conf. Integer Programming and Combinatorial Optim.* (*IPCO* 2002) (Springer-Verlag, Berlin), 401–414.

Caprara A, Malaguti E, Toth P (2011) A freight service design problem for a railway corridor. *Transportation Sci.* 45(2):147–162.

Chen B, Hassin R, Tzur M (2002) Allocation of bandwidth and storage. *IIE Trans.* 34(5):501–507.

Darmann A, Pferschy U, Schauer J (2010) Resource allocation with time intervals. *Theoret. Comput. Sci.* 411(49):4217–4234.

de Aragao MP, Uchoa E (2003) Integer program reformulation for robust branch-and-cut-and-price algorithms. *Proc. Conf. Math. Program in Rio: A Conf. Honour of Nelson Maculan (Rio de Janeiro)*, 56–61.

Furini F (2012) Decomposition and reformulation of integer linear programming problems. *4OR* 10(2):219–220.

Gamrath G, Lübbecke ME (2010) Experiments with a generic Dantzig-Wolfe decomposition for integer programs. *Proc.* 9*th Internat. Sympos. Experiment. Algorithms* (*SEA*) (Springer-Verlag, Berlin), 239–252.

Hall NG, Magazine MJ (1994) Maximizing the value of a space mission. *Eur. J. Oper. Res.* 78(2):224–241.

IBM-CPLEX. Accessed April 2011, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Kellerer H, Pferschy U, Pisinger D (2004) *Knapsack Problems* (Springer-Verlag, Berlin).

Lübbecke M, Desrosiers J (2004) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.

Martello S, Toth P (1990) *Knapsack Problems: Algorithms and Computer Implementations* (John Wiley & Sons, Chichester, UK).

Puchinger J, Raidl GR, Pferschy U (2010) The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. Comput.* 22(2):250–265.

Ralphs TK, Galati MV (2006) Decomposition and dynamic cut generation in integer linear programming. *Math. Programming* 106(2):251–285.

SCIP. Accessed April 2011, http://scip.zib.de/.

Sherali HD, Lee Y, Kim Y (2005) Partial convexification cuts for 0-1 mixed-integer programs. *Eur. J. Oper. Res.* 165(3):625–648.

Vanderbeck F, Wolsey LA (2010) Reformulation and decomposition of integer programs. Jiinger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, eds. *50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art* (Springer-Verlag, Berlin, Heidelberg), 431–502.