A Fully Polynomial Time Approximation Scheme for Packing While Traveling

Frank Neumann[1(✉)], Sergey Polyakovskiy[2], Martin Skutella[3], Leen Stougie[4], and Junhua Wu[1]

[1] Optimization and Logistics, School of Computer Science, The University of Adelaide, Adelaide, Australia
Frank.neumann@adelaide.edu.au

[2] School of Information Technology, Deakin University, Geelong, Australia

[3] Institut für Mathematik, Technische Universität Berlin, Berlin, Germany

[4] CWI, INRIA-Erable and Department of Econometrics and Operations Research, Vrije Universiteit, Amsterdam, The Netherlands

Abstract. Understanding the interaction between different combinatorial optimization problems is a challenging task of high relevance for numerous real-world applications including modern computer and memory architectures as well as high performance computing. Recently, the Traveling Thief Problem (TTP), as a combination of the classical traveling salesperson problem and the knapsack problem, has been introduced to study these interactions in a systematic way. We investigate the underlying non-linear Packing While Traveling Problem (PWTP) of the TTP where items have to be selected along a fixed route. We give an exact dynamic programming approach for this problem and a fully polynomial time approximation scheme (FPTAS) when maximizing the benefit that can be gained over the baseline travel cost. Our experimental investigations show that our new approaches outperform current state-of-the-art approaches on a wide range of benchmark instances.

1 Introduction

Combinatorial optimization problems play a crucial role in diverse application areas such as planning, scheduling, and routing, as well as for the efficient use of modern cloud-based computer architectures as well as high performance computing. Many combinatorial optimization problems have been studied extensively in the literature. Two of the most prominent ones are the traveling salesperson problem (TSP) and the knapsack problem (KP). Numerous high performing algorithms have been designed for these two problems.

Looking at combinatorial optimization problems arising in real-world applications, one can observe that real-world problems often are composed of different types of combinatorial problems. For example, delivery problems usually consists of a routing part for the vehicle(s) and a packing part of the goods onto the vehicle(s). Recently, the Traveling Thief Problem (TTP) [1] has been introduced

Y. Disser and V. S. Verykios (Eds.): ALGOCLOUD 2018, LNCS 11409, pp. 59–72, 2019.
https://doi.org/10.1007/978-3-030-19759-9_5

to study the interactions of different combinatorial optimization problems in a systematic way and to gain better insights into the design of multi-component problems. The TTP combines the TSP and KP by making the speed that a vehicle travels along a TSP tour dependent on the weight of the already selected items. Furthermore, the overall objective is given by the sum of the profits of the collected items minus the weight dependent travel cost along the chosen route. A wide range of heuristic search algorithms [2,3,8] and a large benchmark set [12] have been introduced for the TTP in recent years. However, up to now there are no high performing exact approaches to deal with the TTP. On the other hand, the study of non-linear planning problems is an important topic and the design of efficient approximation algorithms has gained increasing interest in recent years [6,15].

The non-linear Packing While Traveling Problem (PWTP) has been introduced in [13] to push forward systematic studies on multi-component problems and deals with the packing part combined with the non-linear travel cost function of the TTP. The PWTP can be seen as the TTP when the route is fixed but the cost still depends on the weight of the items on the vehicle.

Problem Definition. The PWTP is formally defined as follows. Given are n cities $1, \ldots, n$, distances $d_i \geq 0$, $1 \leq i \leq n-1$, from city i to city $i+1$, together with n items, one at each city. The item at city i has a non-negative integer profit p_i and weight w_i. A vehicle of capacity W travels through the cities in the given order $1, \ldots, n$, and can collect any subset of items $S \subseteq \{1, \ldots, n\}$ of total weight $w(S) := \sum_{i \in S} w_i \leq W$. When traveling from city i to city $i+1$, the speed v of the vehicle depends on the total weight of so far collected items $S_i := S \cap \{1, \ldots, i\}$. More precisely, its speed is an affine linear function of the weight $k = w(S_i)$ given by

$$v(k) := v_{\max} + \frac{k}{W}(v_{\min} - v_{\max}), \tag{1}$$

where $v_{\max}$ is the given maximum possible speed (for the unloaded vehicle) and $v_{\min}$ the given minimum speed (for the fully loaded vehicle). The time $t_i(S_i)$ to travel from city i to city $i+1$ is thus equal to the distance d_i divided by the speed $v_i\big(w(S_i)\big)$. The objective is to choose a subset of items $S \subseteq \{1, \ldots, n\}$ that maximizes the total benefit $b(S) := p(S) - t(S)$, where $p(S) = \sum_{i \in S} p_i$ is the total profit of selected items and $t(S) := \sum_{i=1}^{n-1} t_i(S_i)$ is the total travel time.

In a slightly more general version of the PWTP, there may be several items or no item at any city i. Notice, however, that this can be easily reduced to the special case introduced above. A city with $k > 1$ items can be split into a subsequence of k cities with distances 0 between them. Moreover, at a city with no item we may place a dummy item of profit and weight zero.[1] Further generalizations and interesting variants of the PWTP include other models of weight-dependent travel times occurring in a variety of different application contexts discussed below that can also be handled by the algorithmic techniques introduced in this paper.

[1] Alternatively, an intermediate city with no item might be deleted from the sequence.

The PWTP is NP-hard even without the capacity constraint usually imposed on the knapsack. Furthermore, exact and approximate mixed integer programming approaches as well as a branch-infer-and-bound approach [11] have been developed for this problem.

Applications. The Packing While Traveling Problem is originally motivated by gaining advanced precision when minimizing transportation costs that may have non-linear nature, for example, in applications where weight impacts the fuel costs [4,7]. From this point of view, the problem is a baseline problem in various vehicle routing problems with non-linear costs. Some specific applications of the PWTP may deal with a single truck collecting goods in large remote areas without alternative routes, that is, there may exist a single main route that a vehicle has to follow while any deviations from it in order to visit particular cities are negligible [11].

Applications in the area of modern computing systems include the collection and processing of data by streaming algorithms [16]. Here the sequence of cities/items $1, \ldots, n$ corresponds to a data stream and the capacity W models a bound on the available memory. For multi-level memory architectures, the PWTP's weight-dependent 'travel times' can be interpreted as data processing and computing times that increase with higher memory load; see, e.g., [9]. Further applications in this context include the efficient processing of large amounts of data in social networks and related contexts.

Our Contribution. We introduce a dynamic programming approach for the PWTP. The key idea is to consider the items in the order $1, \ldots, n$ they appear along the route that needs to be traveled and apply dynamic programming similar as for the classical knapsack problem [14]. When considering an item, the decision has to be made on whether or not to pack the item. The dynamic programming approach computes for the first i items, $1 \leq i \leq n$, and possible subsets of weight $\bar{w}$ the maximal objective value that can be obtained. As the programming table that is used depends on the number of different possible weights, the algorithm runs in pseudo-polynomial time.

After having obtained the exact approach based on dynamic programming, we consider the design of a fully polynomial approximation scheme (FPTAS) [5]. First, we show that it is NP-hard to decide whether a given instance of the PWTP has a non-negative objective value. This rules out any polynomial time algorithm with finite approximation ratio, unless $P = NP$. Due to this, we design an FPTAS for the amount that can be gained over the travel cost when the vehicle travels empty (which is the minimal possible travel cost). Our FPTAS is based on the observation that the item with the largest benefit leads to an objective value of at least OPT/n and uses appropriate rounding in the previously designed dynamic programming approach. An interesting and distinguishing feature of our FPTAS is the fact that, in contrast to the standard approach in the area of approximation schemes, we do not explicitly round values to arrive at a polynomial-size state space of the dynamic program. Instead, an approximate domination criterion is used to restrict to a polynomial number of intermediate states.

We evaluate our two approaches on a wide range of instances from the TTP benchmark set [12], and compare them to the exact and approximative approaches given in [11]. Our results show that the large majority of the instances that can be handled by exact methods, are solved much faster by dynamic programming than the previously developed mixed integer programming and branch-infer-and-bound approaches. Considering instances with a larger profit and weight range, we show that the choice of the approximation guarantee significantly impacts the runtime behavior.

Outline. The paper is structured as follows. In Sect. 2 we present the exact dynamic programming approach, and design an FPTAS in Sect. 3. Our experimental results are discussed in Sect. 4. Finally, we finish with some conclusions.

2 Dynamic Programming

We introduce a dynamic programming approach for solving the PWTP. Dynamic programming is one of the traditional approaches for the classical knapsack problem [14]. The dynamic programming table β consists of n rows, indexed by $i = 1, \ldots, n$, and $W + 1$ columns, indexed by $k = 0, \ldots, W$. Items are processed in the order $i = 1, \ldots, n$ they appear along the tour. The entry $\beta(i, k)$ shall denote the maximal benefit that can be obtained by considering all subsets of the first i items $\{1, \ldots, i\}$ of total weight exactly k, for $k = 0, \ldots, W$. We denote by $\beta(i, \cdot)$ the row containing the entries $\beta_{i,k}$. In the case that a subset of total weight k does not exist, we set $\beta(i, k) := -\infty$.

Let $d_{i,n} := \sum_{j=i}^{n-1} d_j$ be the distance from city i to the last city n. We denote by $b(\emptyset) := -d_{1,n}/v_{\max}$ the benefit of the empty set, that is, the travel cost when the vehicle travels empty. Furthermore, the benefit when only item i is chosen is

$$b(\{i\}) := b(\emptyset) + p_i - \frac{d_{i,n}}{v(w_i)} + \frac{d_{i,n}}{v_{\max}},$$

as the vehicle will now only travel at speed $v(w_i)$ from city i on. The entries in the first row can be easily computed as

$$\beta(1, k) := \begin{cases} b(\emptyset) & \text{if } k = 0 \neq w_1, \\ b(\{1\}) & \text{if } k = w_1, \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

For $i = 2, \ldots, n$, based on the row $\beta(i-1, \cdot)$ we can compute the next row $\beta(i, \cdot)$. To keep notation simple, we let $\beta(i-1, q) := -\infty$ for $q < 0$. Then,

$$\beta(i, k) := \max\left\{\beta(i-1, k), \beta(i-1, k - w_i) + p_i - \frac{d_{i,n}}{v(k)} + \frac{d_{i,n}}{v(k - w_i)}\right\}. \tag{3}$$

The correctness of this recursive formula is discussed in the proof of the next theorem.

Theorem 1. *For each i and k, the entry $\beta(i,k)$ stores the maximal possible benefit $b(S)$ over all subsets S of $\{1,\ldots,i\}$ having weight exactly k. In particular,* $\max_k \beta(n,k)$ *is the value of an optimal solution, which can be obtained via backtracking.*

Proof. We use induction on i. The statement is true for $i = 1$ as there are only the two options of choosing or not choosing the first item, which are both considered in (2). Now assume that $\beta(i-1,k)$ stores the maximal benefit for each weight k when considering all subsets of $\{1,\ldots,i-1\}$. Notice that for a subset $S' \subseteq \{1,\ldots,i-1\}$ of weight at most $W - w_i$, the benefit of $S' \cup \{i\}$ equals

$$b(S' \cup \{i\}) = b(S') + p_i - \left(\frac{d_{i,n}}{v\big(w(S') + w_i\big)} - \frac{d_{i,n}}{v\big(w(S')\big)} \right), \tag{4}$$

since adding item i to subset S' leads to the reduced speed $v\big(w(S') + w_i\big)$ of the vehicle, instead of $v\big(w(S')\big)$, from city i on. Consider now a subset $S \subseteq \{1,\ldots,i\}$ with $w(S) = k$ of maximum benefit $b(S)$. If $i \notin S$, then S must obviously be a maximum benefit subset of $\{1,\ldots,i-1\}$ of weight k as well. In particular, $b(S) = \beta(i-1,k)$; see the first term on the right-hand side of (3). Otherwise, if $i \in S$, then $S = S' \cup \{i\}$ for a maximum benefit subset $S' \subseteq \{1,\ldots,i-1\}$ of weight $k - w_i$, that is, $b(S') = \beta(i-1,k-w_i)$. Notice that the second term on the right-hand side of (3) thus coincides with (4). This concludes the proof.

Finally, we investigate the runtime for this dynamic program. If $d_{i,n}$ has been computed for each i, which takes $O(n)$ time in total, then each entry of the dynamic programming table β can be computed in constant time. Thus, the running time of the dynamic program is in $O(nW)$. To empirically speed up the computation of the dynamic program, it is sufficient to only store an entry for $\beta(i,k)$ if it is not dominated by any other entry in $\beta(i,\cdot)$, that is, if there is no $k' < k$ with $\beta(i,k') \geq \beta(i,k)$. This is justified by the following lemma.

Lemma 1. *The increase in travel cost due to a new item i given by the term in brackets on the right-hand side of (4) is an increasing function of the weight $w(S')$ of so far collected items.*

Proof. For $v(k)$ as defined in (1), let $t(k) := 1/v(k)$ denote the travel time per unit distance when the vehicle has collected items of total weight k. Notice that the thereby defined function $t : [0,W] \to \mathbb{R}_{\geq 0}$ is convex and increasing.

3 Approximation Algorithms

We now turn our attention to approximation algorithms. The NP-hardness proof for the PWTP given in [11] does not rule out polynomial time approximation algorithms. In this section, we first show that polynomial time approximation algorithms with a finite approximation ratio do not exist, unless $P = NP$. This results motivates the design of an FPTAS for the shifted objective function given by the amount that can be gained over the baseline cost when the vehicle is traveling empty.

3.1 Inapproximability of the Packing While Traveling Problem

The objective function for PWTP can take on positive and negative values. We show that deciding whether a given PWTP instance has a solution that is non-negative is already NP-complete.

Theorem 2. *Given a PWTP instance, the problem to decide whether there is a solution $S \subseteq \{1, \ldots, n\}$ with $b(S) \geq 0$ is NP-complete.*

Proof. The problem is obviously in NP as one can verify in polynomial time for a given solution S whether $b(S) \geq 0$ holds by evaluating the objective function. It remains to show that the problem is NP-hard.

We reduce the *NP*-complete *Subset Sum Problem* (SSP) to our problem. An instance of SSP is given by n positive integers $\{s_1, \ldots, s_n\}$ and a positive integer Q. The question is whether there exists a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} s_i = Q$. Given an instance of SSP, we construct an instance of PWTP consisting of n cities and items of profit and weight $p_i = w_i = s_i$, for $i = 1, \ldots, n$. The distances d_i between cities are all equal to zero except for the last distance $d_{n-1} := Q^2$. Finally, the vehicle has capacity $W := Q$ and its minimum and maximum speed are $v_{\min} := v_{\max} := Q$, that is, the speed does not depend on the weight of collected items. It is easy to see that the benefit of any solution $S \subseteq \{1, \ldots, n\}$ is equal to $b(S) = p(S) - Q = \sum_{i \in S} s_i - Q$. In particular, as $p(S) = w(S) \leq W = Q$, it holds that $b(S) \geq 0$ if and only if S is a solution to the underlying instance of the SSP.

We can even prove the following slightly stronger complexity result.

Proposition 1. *The decision version of the PWTP stated in Theorem 2 is even NP-hard if the vehicle capacity is large enough to fit all items, that is, if $W \geq w(\{1, \ldots, n\})$.*

Proof. We modify the reduction given in the proof of Theorem 2 as follows. First of all we restrict to instances of the SSP with $\sum_{i=1}^{n} s_i = 2Q$ (in other words, we give a reduction from the NP-complete Partition Problem). The vehicle capacity is then set to $W := 2Q$, the maximum speed to $v_{\max} := 2Q$, and the minimum speed to $v_{\min} := 0$. Then, the benefit of a subset of items $S \subseteq \{1, \ldots, n\}$ is

$$b(S) = p(S) - \frac{Q^2}{2Q - w(S)} = w(S) - \frac{Q^2}{2Q - w(S)}.$$

We consider the right-hand side term as a function of $w(S)$. It is easy to check that this function attains its unique maximum of value 0 for $w(S) = Q$.

As a corollary of Theorem 2, we obtain the following non-approximability result.

Corollary 1. *There is no polynomial time approximation algorithm for PWTP with a finite approximation ratio, unless $P = NP$.*

3.2 An FPTAS for Amount over Baseline Travel Cost

In view of Corollary 1, we shift the objective function value and consider the amount that can be gained over the cost when the vehicle travels empty as the new objective. More precisely, for a subset of items $S \subseteq \{1, \ldots, n\}$ the new objective is

$$b'(S) := b(S) - b(\emptyset).$$

This is motivated by the scenario where the vehicle has to travel along the given route anyway, and the goal is to maximize the gain over this (negative) baseline cost $b(\emptyset)$. Notice that an optimal solution for this objective is also an optimal solution for the original PWTP objective. Approximation results, however, do not carry over as the objective value is shifted by $b(\emptyset)$.

As in the proof of Lemma 1, let $t(k)$ be the travel time per unit distance when the vehicle has collected items of total weight k. It follows from the proof of Lemma 1 that, for each item i and $0 \leq k \leq W - w_i$, we get

$$t(k + w_i) - t(k) \geq t(w_i) - t(0).$$

This means that the marginal cost (with respect to the travel time) of adding an item is lowest if there are no other items chosen. As a consequence, we get for each subset $S \subseteq \{1, \ldots, n\}$ with $w(S) \leq W$ that

$$b'(S) \leq \sum_{i \in S} b'(\{i\}).$$

In particular, when choosing an optimal subset S maximizing $b'(S) =: \text{OPT}$, there is an $i \in S$ with $b'(i) \geq \text{OPT}/|S| \geq \text{OPT}/n$. Thus, $L := \max_{1 \leq i \leq n} b'(\{i\})$ provides an efficiently computable lower bound on the value of an optimal solution satisfying $\text{OPT}/n \leq L \leq \text{OPT}$.

In order to obtain a fully polynomial time approximation scheme (FPTAS) for the problem of maximizing $b'(S)$ over all feasible subsets $S \subseteq \{1, \ldots, n\}$, we start by carefully modifying the dynamic programming scheme from Sect. 2 given by Eqs. (2) and (3) as follows. Let

$$\beta'(1, k) := \begin{cases} b'(\emptyset) & \text{if } k = 0 \neq w_1, \\ b'(\{1\}) & \text{if } k = w_1, \\ -\infty & \text{otherwise.} \end{cases}$$

Then, for $i = 2, \ldots, n$, let

$$\beta'(i, k) := \max\left\{\beta'(i-1, k), \beta'(i-1, k-w_i) + p_i - \frac{d_{i,n}}{v(k)} + \frac{d_{i,n}}{v(k-w_i)}\right\}.$$

As discussed at the end of Sect. 2, we can speed up the dynamic program by setting $\beta'(i, k) := -\infty$ in case there is a $k' < k$ with $\beta'(i, k') \geq \beta'(i, k)$.

The idea of the FPTAS described in Algorithm 1 is to further speed up the dynamic program by ignoring entries $\beta'(i, k)$ such that there is a $k' < k$ with

Algorithm 1. FPTAS for maximizing $b'(S)$

1. set $L := \max_{1 \leq i \leq n} b'(\{i\})$, $r := \epsilon L/n$, and $d_{i,n} := \sum_{j=i}^{n-1} d_j$ for $1 \leq i \leq n$;
2. initially, all values $\beta(i, k)$ are assumed to be $-\infty$;
3. set $\beta'(1, 0) := b'(\emptyset)$ and $\beta'(1, w_1) := b'(\{1\})$;
4. for $i = 1, \ldots, n-1$ do:
5. for each k with $\lfloor \beta'(i, k)/r \rfloor > \max\{\lfloor \beta'(i, k')/r \rfloor, -\infty\}$ for all $k' < k$ do:
6. set $\beta'(i+1, k) := \max\{\beta'(i, k), \beta'(i+1, k)\}$;
7. if $k^+ := k + w_{i+1} \leq W$, set
$$\beta'(i+1, k^+) := \max\{\beta'(i, k) + p_{i+1} - \tfrac{d_{i+1,n}}{v(k^+)} + \tfrac{d_{i+1,n}}{v(k)}, \beta'(i+1, k^+)\}$$
8. determine $\max_k \beta'(n, k)$ and corresponding solution S by backtracking;

$\lfloor \beta'(i, k)/r \rfloor > \lfloor \beta'(i, k')/r \rfloor$ for $r := \epsilon L/n$. Due to this, in terms of the objective function we lose at most r in every row of the dynamic programming table. The overall loss is thus bounded by $nr = \epsilon L \leq \epsilon \mathrm{OPT}$.

Theorem 3. *Algorithm 1 is an FPTAS for the problem to maximize $b'(S)$ over all subsets of items $S \subseteq \{1, \ldots, n\}$ with $w(S) \leq W$.*

Proof. As argued above, the value of the computed solution is at least $(1 - \epsilon)\mathrm{OPT}$. It remains to argue that the running time of Algorithm 1 is bounded by a polynomial in the input size and $1/\epsilon$. This can be seen as follows:

Claim. For the dynamic programming table β' computed by Algorithm 1, there are at most $O(n^2/\epsilon)$ entries of finite value in row $\beta'(i, \cdot)$, for $i = 1, \ldots, n$.

Proof of the Claim: We use induction on i. The case $i = 1$ is clear by Step 3 of Algorithm 1. Moreover, the for-loop in Step 5 considers at most $1 + \mathrm{OPT}/r = 1 + n\mathrm{OPT}/(\epsilon L) \leq 1 + n^2/\epsilon$ different values of k. For each such k, at most two entries in the next row $i + 1$ are modified. This concludes the proof of the claim. The overall running time is thus polynomial in the input size and $1/\epsilon$.

We conclude this section with the following generalizing remark.

Remark 1. The construction of the FPTAS only used the fact that the travel time per unit distance is monotonically increasing and convex. Hence, the FPTAS holds for any PWTP problem where the travel time per unit distance has this property.

4 Experiments and Results

In this section, we investigate the effectiveness of the proposed DP and FPTAS approaches based on our implementations in Java. We mainly focus on two issues: (1) studying how the DP and FPTAS perform compared to the state-of-the-art approaches; (2) investigating how the performance and accuracy of the FPTAS change when the parameter ϵ is altered.

In order to be comparable to the mixed integer programming (MIP) and the branch-infer-and-bound (BIB) approaches presented in [11], we conduct our experiments on the same families of test instances. Our experiments are carried out on a computer with 4 GB RAM and a 3.06 GHz Intel Dual Core processor, which is also the same as the machine used in the paper mentioned above.

We compare the DP to the exact MIP (*eMIP*) and the branch-infer-and-bound approaches as well as the FPTAS to the approximate MIP (*aMIP*), as the former three are all exact approaches and the latter two are all approximations. Table 1 demonstrates the results for a route of 101 cities and various types of packing instances. For this particular family, we consider three types of instances: *uncorrelated* (uncorr), *uncorrelated with similar weights* (uncorr-s-w) and *bounded strongly correlated* (b-s-corr), which are further distinguished by the different correlations between profits and weights. In combination with three different numbers of items and three settings of the capacity, we have 27 instances in total, as shown in the column called "*Instance*". Similarly to the settings in [11], every instance with "_01" postfix has a relatively small capacity. We expect such instances to be potentially easy to solve by DP and FPTAS due to the nature of the algorithms. The *OPT* column shows the optimum of each instance and the *RT(s)* columns illustrate the running time for each of the approaches in the time unit of a second. To demonstrate the quality of an approximate approach applied to the instances, we use the ratio between the objective value obtained by the algorithm and the optimum obtained for an instance as the approximation rate $AR(\%) = 100 \times \frac{OBJ}{OPT}$.

In the comparison of exact approaches, our results show that the DP is much quicker than the exact MIP and BIB in solving the majority of the instances. The exact MIP is slower than the DP in every case and this dominance is mostly significant. For example, it spends around 35 min to solve the instance *uncorr-s-w_10* with $1,000$ items, where the DP needs around 15 s only. On the other hand, the BIB slightly beats the DP on three instances, but the DP is superior for the rest 24 instances. An extreme case is *b-s-corr_01* with $1,000$ items where the BIB spends above 1.5 h while the DP solves it in 11 s only. Concerning the running time of the DP, it significantly increases only for the instances having large amount of items with strongly correlated weights and profits, such as *b-s-corr_06* and *b-s-corr_10* with $1,000$ items. However, *b-s-corr_01* seems exceptional due to the limited capacity assigned to the instance.

Our comparison between the approximation approaches shows that the FPTAS has significant advantages as well. The approximation ratios remain 100% when ϵ equals 0.0001 and 0.01. Only when ϵ is set to 0.25, the FPTAS starts to output the results having similar accuracies as the ones of *aMIP*. With regard to the performance, the FPTAS takes less running time than *aMIP* on the majority of the instances despite the setting of ϵ. As an extreme case, *aMIP* requires hours to solve the *uncorr-s-w_01* instance with $1,000$ items, but the FPTAS takes less than a second. However, the *aMIP* performs much better on *b-s-corr_06* and *b-s-corr_10* with $1,000$ items. This somehow indicates that the underlying factors that make instances hard to solve by approximate MIP and

Table 1. Results on small range instances

Instance	m	OPT	Exact sproaches			Approximation approaches											
			eMIP	BIB	DP	aMIP		FPTAS									
								$\epsilon = 0.0001$		$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0.25$		$\epsilon = 0.75$	
			RT(s)	RT(s)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)
Instance family eil101																	
uncorr_01	100	1651.697	1.217	5.694	0.027	100	3.838	100	0.001	100	0.001	100	0.001	100	0.001	100	0.025
uncorr_06	100	10155.4942	12.605	3.698	0.065	100	4.961	100	0.012	100	0.011	100	0.011	100	0.011	99.9928	0.063
uncorr_10	100	10297.7134	3.525	0.795	0.036	100	0.624	100	0.017	100	0.017	99.9939	0.016	99.9939	0.016	99.9653	0.037
uncorr-s-w_01	100	2152.6188	0.328	7.566	0.001	100	3.978	100	0	100	0	100	0	100	0	100	0.003
uncorr-s-w_06	100	4333.8512	12.59	2.215	0.012	100	2.699	100	0.008	100	0.007	100	0.007	99.9569	0.008	99.9569	0.017
uncorr-s-w_10	100	9048.4908	37.144	1.107	0.022	100	1.763	100	0.012	100	0.012	100	0.012	100	0.013	99.9355	0.02
b-s-corr_01	100	4441.9852	1.42	125.954	0.014	100	5.366	100	0.01	100	0.009	100	0.009	100	0.008	100	0.013
b-s-corr_06	100	10260.9767	4.509	22.541	0.101	100	2.761	100	0.058	100	0.057	100	0.048	100	0.043	100	0.087
b-s-corr_10	100	13630.6153	11.013	27.081	0.187	99.9971	3.713	100	0.103	100	0.101	99.9971	0.081	99.9606	0.065	99.8143	0.113
uncorr_01	500	17608.5781	19.594	27.581	0.247	100	5.757	100	0.171	100	0.161	100	0.153	100	0.163	100	0.377
uncorr_06	500	56294.5239	384.213	13.354	2.829	100	7.8	100	2.37	100	2.344	100	2.3	100	2.212	100	2.34
uncorr_10	500	66141.484	211.302	2.325	4.01	100	0.718	100	3.72	100	3.645	100	3.446	100	3.531	100	3.632
uncorr-s-w_01	500	13418.8406	4.337	34.866	0.09	100	50.31	100	0.085	100	0.09	100	0.084	100	0.087	99.991	0.085
uncorr-s-w_06	500	34280.473	346.43	7.285	1.04	100	9.609	100	0.964	100	0.933	100	0.905	100	0.936	100	0.92
uncorr-s-w_10	500	50836.6588	519.902	3.338	2.022	100	3.354	100	2.005	100	1.783	100	1.753	100	1.784	100	2.147
b-s-corr_01	500	21306.9158	40.482	624.204	1.534	100	13.338	100	1.373	100	1.279	100	1.116	100	0.949	100	0.716
b-s-corr_06	500	69370.2367	236.387	97.313	14.616	99.9996	7.847	100	13.393	100	12.975	100	11.642	99.9996	9.741	99.9996	6.018
b-s-corr_10	500	82033.9452	376.569	218.728	22.011	100	2.309	100	21.372	100	20.829	100	18.573	100	15.313	99.9943	8.84
uncorr_01	1000	36170.9109	218.306	114.567	1.872	99.9993	11.918	100	1.891	100	1.875	100	1.832	100	1.845	100	1.764
uncorr_06	1000	93949.1981	1261.949	36.847	20.944	100	17.971	100	17.024	100	16.615	100	16.545	100	16.378	100	15.713
uncorr_10	1000	122963.6617	620.896	4.821	30.116	100	2.184	100	27.305	100	26.783	100	26.541	100	26.051	100	23.905
uncorr-s-w_01	1000	27800.9614	241.957	399.158	0.802	100	4985.566	100	0.73	100	0.69	100	0.688	100	0.724	100	0.687
uncorr-s-w_06	1000	61764.4599	1152.624	12.792	9.872	100	19.063	100	8.686	100	8.812	100	8.56	100	8.74	100	8.396
uncorr-s-w_10	1000	103572.4074	2146.408	7.644	15.047	100	9.688	100	14.03	100	13.912	100	13.797	100	13.982	100	13.492
b-s-corr_01	1000	46886.1094	378.551	6129.531	11.783	99.9988	46.394	100	11.714	100	11.358	100	10.793	100	9.592	100	6.536
b-s-corr_06	1000	125830.6887	643.533	919.201	94.523	99.9999	10.311	100	92.411	100	91.039	100	83.002	99.9999	71.078	100	45.433
b-s-corr_10	1000	161990.5015	862.572	1646.52	151.601	100	7.16	100	150.279	100	149.722	100	134.764	100	113.049	99.9981	70.135

Table 2. Results of DP and FPTAS on large range instances

Instance	m	DP		FPTAS													
				$\epsilon = 0.0001$		$\epsilon = 0.001$		$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0.25$		$\epsilon = 0.5$		$\epsilon = 0.75$	
		OPT	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)
Instance family `eil101_large-range`																	
uncorr_01	100	69802802.2801	0.03	100	0.002	100	0.002	100	0.002	100	0.002	100	0.002	100	0.002	100	0.029
uncorr_06	100	204813765.6933	0.053	100	0.019	100	0.02	100	0.019	100	0.019	100	0.019	100	0.019	100	0.049
uncorr_10	100	172176182.1249	0.041	100	0.028	100	0.028	100	0.028	100	0.028	100	0.027	100	0.026	99.9628	0.037
uncorr-s-w_01	100	36420530.5753	0.006	100	0.003	100	0.003	100	0.003	100	0.003	100	0.003	100	0.002	100	0.004
uncorr-s-w_06	100	148058928.2952	0.098	100	0.072	100	0.502	100	0.072	100	0.069	100	0.065	100	0.059	100	0.07
uncorr-s-w_10	100	142538516.4602	0.136	100	0.101	100	0.104	100	0.103	99.9978	0.096	99.9978	0.086	99.9978	0.073	99.9978	0.089
m-s-corr_01	100	19549602.2671	0.003	100	0.002	100	0.002	100	0.002	100	0.002	100	0.002	100	0.001	100	0.002
m-s-corr_06	100	137203175.1921	0.147	100	0.115	100	0.118	100	0.113	100	0.089	100	0.063	100	0.04	100	0.043
m-s-corr_10	100	225584278.6004	0.424	100	0.326	100	0.329	100	0.312	100	0.2	100	0.179	100	0.086	100	0.073
uncorr_01	500	385692662.0930	0.47	100	0.451	100	0.454	100	0.619	100	0.508	100	0.445	100	0.43	100	0.517
uncorr_06	500	958013934.6172	3.539	100	3.749	100	7.431	100	3.947	100	3.69	99.9996	3.677	99.9996	3.486	99.9993	3.021
uncorr_10	500	844949838.4389	4.87	100	5.393	100	5.716	100	5.483	100	5.135	100	4.851	99.9992	4.609	99.9992	4.295
uncorr-s-w_01	500	182418888.9364	1.157	100	1.157	100	1.199	100	1.145	99.9995	1.112	99.9995	1.063	99.9995	0.977	99.9904	0.929
uncorr-s-w_06	500	780432253.0187	22.39	100	25.04	100	26.276	100	24.024	100	23.282	99.9997	21.756	99.9997	18.293	99.9997	18.411
uncorr-s-w_10	500	714433353.7957	30.959	100	34.458	100	39.004	100	34.308	100	32.308	99.9996	28.792	99.999	26.392	99.999	25.971
m-s-corr_01	500	96463941.1275	2.335	100	2.478	100	2.782	100	2.695	100	1.509	100	0.963	100	0.546	100	0.408
m-s-corr_06	500	666701000.1488	108.705	100	126.833	100	139.63	100	122.75	100	62.479	100	33.547	100	17.959	100	10.642
m-s-corr_10	500	1082009880.5886	262.999	100	299.862	100	317.352	100	274.284	100	145.087	100	78.47	99.9994	41.816	99.9994	25.924
uncorr_01	1000	777386336.9660	4.222	100	4.397	100	4.347	100	4.309	100	4.341	100	4.377	100	4.28	100	4.24
uncorr_06	1000	1933319297.4248	46.043	100	51.383	100	53.087	100	48.861	100	52.957	99.9999	52.062	99.9997	50.286	99.9996	51.488
uncorr_10	1000	1693797490.1704	64.485	100	76.744	100	78.847	100	74.128	100	82.754	100	77.057	100	72.283	100	72.567
uncorr-s-w_01	1000	361991311.8336	14.254	100	15.072	100	15.67	100	14.523	100	14.11	100	14.039	100	12.088	100	11.129
uncorr-s-w_06	1000	1574469459.3163	286.843	100	318.096	100	330.508	100	337.289	100	334.318	100	307.588	99.9998	270.013	99.9996	245.927
uncorr-s-w_10	1000	1439410696.3695	393.793	100	438.775	100	455.83	100	464.527	100	441.955	100	433.672	99.9994	378.917	99.9994	340.813
m-s-corr_01	1000	191170309.5684	46.858	100	58.031	100	59.987	100	58.101	100	31.703	100	18.771	100	10.728	100	6.831
m-s-corr_06	1000	1315708161.7720	2393.205	100	2512.281	100	2606.412	100	1921.573	100	666.749	100	364.452	100	208.969	100	150.06
m-s-corr_10	1000	2163713055.3759	6761.49	100	6668.535	100	6441.906	100	4526.653	100	1334.882	100	703.258	100	397.527	100	282.211

FPTAS have different nature. Understanding these factors more and using them wisely should help to build a more powerful algorithm with mixed features of MIP and FPTAS.

In our second experiment, we use test instances which are slightly different to those in the benchmark used in [11]. This is motivated by our findings that relaxing ϵ from 0.0001 to 0.75 improves the runtime performance of FPTAS by around 50% for the b-s-corr instances, while does not degrade the accuracy noticeably. At the same time, there is no significant improvement for other instances. It's surprising as shows that the performance improvement can be easily achieved on complex instances. Therefore, we study how the FPTAS performs if the instances are more complicated. The idea is to use instances with large weights, which are known to be difficult regarding dynamic programming based approaches for the classical knapsack problem. We follow the same way to create TTP instances as proposed in [12] and generate the knapsack component of the problem as discussed in [10]. Specifically, we extend the range to generate potential profits and weights from $[1, 10^3]$ to $[1, 10^7]$ and focus on *uncorrelated* (uncorr), *uncorrelated with similar weights* (uncorr-s-w), and *multiple strongly correlated* (m-s-corr) types of instances. Additionally, in the stage of assigning the items of a knapsack instance to particular cities of a given TSP tour, we sort the items in descending order of their profits and the second city obtains k, $k \in \{1, 5, 10\}$, items of the largest profits, the third city then has the next k items, and so on. We expect that such assignment should force the algorithms to select items in the first cities of a route making the instances more challenging for the DP and FPTAS. In reality, these instances indeed are harder than the ones in the first experiment, which forces us to switch to the 128 GB RAM and $8 \times$ (2.8 GHz AMD 6 core processors) cluster machine to carry out the second experiment.

Table 2 illustrates the results of running the DP and FPTAS on the instances with the large range of profits and weights. Generally speaking, we can observe that the instances are significantly harder to solve than those ones from the first experiment, that is they take comparably more time. Similarly, the instances with large number of items, larger capacity, and strong correlation between profits and weights are now hard for the DP as well. Oppositely to the results of the previous experiment, the FPTAS performs much better when dealing with such instances in the case when ϵ is relaxed. For example, its performance is improved by 95% for the instance *m-s-corr_10* with 1,000 items when ϵ is raised from 0.0001 to 0.75 while the approximation rate remains at 100%.

5 Conclusion

Multi-component combinatorial optimization problems play an important role in many real-world applications. We have examined the non-linear Packing While Traveling Problem which results from the interactions in the Traveling Thief Problem. We designed a dynamic programming algorithm that solves the problem in pseudo-polynomial time. Furthermore, we have shown that the original objective of the problem is hard to approximate and have given an FPTAS for optimizing the amount that can be gained over the smallest possible travel cost. It should be noted that the FPTAS applies to a wider range of problems as our proof only assumed that the travel cost per unit distance in dependence of the weight is increasing and convex. Our experimental results on different types of knapsack instances show the advantage of the dynamic program over the previous approaches based on mixed integer programming and branch-infer-and-bound concepts. Furthermore, we have demonstrated the effectiveness of the FPTAS on instances with a large weight and profit range.

Acknowledgements. The first, second, and fifth author were supported by Australian Research Council grants DP130104395 and DP140103400. The third author is supported by the Einstein Foundation Berlin in the framework of Matheon.

References

1. Bonyadi, M., Michalewicz, Z., Barone, L.: The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1037–1044, June 2013. https://doi.org/10.1109/CEC.2013.6557681
2. El Yafrani, M., Ahiod, B.: Population-based vs. single-solution heuristics for the travelling thief problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016 GECCO 2016, pp. 317–324. ACM, New York (2016). https://doi.org/10.1145/2908812.2908847
3. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation GECCO 2015, pp. 385–392. ACM, New York (2015).https://doi.org/10.1145/2739480.2754716
4. GOODYEAR: Factors Affecting Truck Fuel Economy (2008). http://www.goodyeartrucktires.com/pdf/resources/publications/FactorsAffectingTruckFuelEconomy.pdf
5. Hochbaum, D.: Appromixation Algorithms for NP-hard Problems. PWS Publishing Company (1997)
6. Hoy, D., Nikolova, E.: Approximately optimal risk-averse routing policies via adaptive discretization. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 3533–3539. AAAI Press, Austin, 25–30 January 2015
7. Lin, C., Choy, K., Ho, G., Chung, S., Lam, H.: Survey of green vehicle routing problem: past and future trends. Expert Syst. Appl. **41**(4, Part 1), 1118–1138 (2014). https://doi.org/10.1016/j.eswa.2013.07.107
8. Mei, Y., Li, X., Yao, X.: On investigation of interdependence between subproblems of the travelling thief problem. Soft Comput. **20**, 157–172 (2016). https://doi.org/10.1007/s00500-014-1487-2
9. Meyer, U., Sanders, P., Sibeyn, J. (eds.): Algorithms for Memory Hierarchies. LNCS, vol. 2625. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36574-5
10. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. **32**, 2271–2284 (2005). https://doi.org/10.1016/j.cor.2004.03.002

11. Polyakovskiy, S., Neumann, F.: The packing while traveling problem. Eur. J. Oper. Res. **258**, 424–439 (2017)
12. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation GECCO 2014, pp. 477–484. ACM, New York (2014). https://doi.org/10.1145/2576768.2598249
13. Polyakovskiy, S., Neumann, F.: Packing while traveling: mixed integer programming for a class of nonlinear knapsack problems. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 332–346. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18008-3_23
14. Toth, P.: Dynamic programming algorithms for the zero-one knapsack problem. Computing **25**, 29–45 (1980). https://doi.org/10.1007/BF02243880
15. Yang, G., Nikolova, E.: Approximation algorithms for route planning with nonlinear objectives. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 3209–3217. AAAI Press, Phoenix, 12–17 February 2016
16. Zhang, J.: A survey on streaming algorithms for massive graphs. In: Aggarwal, C., Wang, H. (eds.) Managing and Mining Graph Data. Advances in Database Systems, vol. 40, pp. 393–420. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-6045-0_13