Online appendix to the paper "The prize collecting traveling salesman problem in warehouses"

by Stefan Bock and Nils Boysen

EC.1. An alternative MIP approach

This section elaborates on an alternative MIP approach in order to benchmark the branch&bound algorithm in our computational study. Specifically, we apply the MIP of Bérubé, Gendreau, and Potvin (2009), which maps the routing problem into an undirected weighted graph with a complete edge set. While each open storage position and the IO point (with index 0) constitute the set of vertices $V = \{0, 1, \ldots, n\}$, the edges of set $E = \{(u, v) \mid u < v \in V\}$ represent the service of two related storage positions in direct succession. This is indicated by setting the corresponding binary variable x_e to one. In a feasible tour, each serviced node $v \in V$, indicated by setting $y_v = 1$ and contributing a storage capacity of c_v, is connected by exactly two edges, while no sub-cycle occurs. The latter is ensured by an exponential number of subtour elimination constraints, and we dub the resulting model the BGP-PCTSP-MIP. By additionally assuming a minimum number of three visited vertices, this leads to the following MIP (see Bérubé, Gendreau, and Potvin 2009):

$$\text{BGP-PCTSP-MIP: Minimize} \sum_{e \in E} t_e \cdot x_e \tag{EC.1}$$

$$\sum_{v \in V} c_v \cdot y_v \geq m \tag{EC.2}$$

$$\sum_{e=(v,u) \in E} x_e + \sum_{e=(u,v) \in E} x_e = 2 \cdot y_v \qquad \forall\, v = 1, \ldots, n \tag{EC.3}$$

$$y_0 = 1 \tag{EC.4}$$

$$\sum_{u \in S} \Big(\sum_{e=(w,u) \in E} x_e + \sum_{e=(u,w) \in E} x_e \geq 2 \cdot y_v \Big) \quad \forall S \subsetneqq V \text{ with } 0 \in S : v \in V \setminus S \tag{EC.5}$$

$$x_e \in \{0, 1\} \qquad \forall\, e = (u, v) \in E \tag{EC.6}$$

$$y_v \in \{0, 1\} \qquad \forall\, v = 1, \ldots, n \tag{EC.7}$$

When applying this MIP to the W-PCTSP, analogously to the valid inequalities (10) added to our alternative MIP of Section 3, we can derive additional restrictions for exploiting the specific parallel-aisle structure of warehouses. Each aisle can only be visited from the north or south, and every open storage position v visited is filled to capacity until all m products are readily stowed. Thus, we can exclude edges between non-neighboring positions along each vertical aisle. This is realized by the following restrictions:

$$x_e = 0 \;\; \forall\, 1 \leq u < v < w \leq n \text{ with } \epsilon(u) = \epsilon(v) = \epsilon(w) \text{ and } e = (u, w). \tag{EC.8}$$

By adding restrictions (EC.8), BGP-PCTSP-MIP turns into BGP-W-PCTSP-MIP and is only applicable to parallel-aisle structures.

However, the scalability of both resulting MIPs is strongly limited by the exponential number of subtour elimination constraints (EC.5). Preliminary tests have shown that instances with more than $n = 15$ open storage positions are beyond the capabilities of CPLEX when fed with these two MIPs. Thus, we add subtour elimination constraints in an iterative manner as so-called lazy constraints. We start with the above MIP where all restrictions (EC.5) are excluded. In each iteration, the solution generated by CPLEX is checked for feasibility. This is done by following the generated tour T^c starting from the IO point. If all storage positions v with $y_v = 1$ are visited, this tour is optimal, so that the procedure terminates. In the opposite case, however, subtours are detected. To exclude them in the next iteration, we add the respective constraints (EC.5) for each non-visited position u with $y_u = 1$, while setting S to the positions visited by tour T^c, and repeat the above.

EC.2. Proof of computational complexity

This appendix provides the proof that W-PCTSP is binary $\mathcal{NP}$-hard, even if only $h = 2$ horizontal cross aisles exist.

The proof is given by a reduction from the binary $\mathcal{NP}$-complete knapsack problem (KP). We consider an instance $\phi = (n, \kappa, P, p_1, \ldots, p_n, w_1, \ldots, w_n)$ of the KP with knapsack capacity κ, price bound P, and elements $i \in \{1, \ldots, n\}$ each having a positive weight w_i and a positive price p_i. Instance ϕ obtains a yes-certificate if and only if there exists a subset $S \subseteq \{1, \ldots, n\}$, such that $\sum_{i \in S} w_i \leq \kappa$ and $\sum_{i \in S} p_i \geq P$.

We define a corresponding instance of the W-PCTSP $f(\phi)$ with two cross aisles β_1 and β_2. Between them, each element i of the KP is mapped by a vertical aisle α_i possessing a significant length L such that the stower never travels along back cross aisle β_2 and only uses front cross aisle β_1 where the IO point is located. In each aisle α_i, we have the ith open storage position with capacity of $c_i = p_i$ products, which is located such that the cyclical tour from crossing node $w_{i,1}$ to v_i and back takes $k \cdot w_i$ time units for some suitably defined k. The IO point is located at crossing node $w_{1,1}$, whereas the symmetric distance between two neighboring vertical aisles is one. Hence, for a round trip along the front cross aisle that covers all vertical aisles with intermediate visits of nodes v_j at aisles α_j, the stower requires no more than $2(n-1)$ time units and we set $k = 2n$ wherefore all travel times along cross aisle β_1 together are less than k time units. Furthermore, it holds that $L = k \cdot (\kappa + 1) + 1$. Hence, by setting the number of products to be stored to $m = P$, we claim that the resulting W-PCTSP instance $f(\phi)$ has a feasible tour with a total duration of less than $k \cdot (\kappa + 1)$ if and only if ϕ obtains a yes-certificate.

If ϕ obtains a yes-certificate, there exists a set $S \subseteq \{1,\dots,n\}$ fulfilling $\sum_{i\in S} w_i \le \kappa$ and $\sum_{i\in S} p_i \ge P$. Hence, we store $\sum_{i\in S} c_i = \sum_{i\in S} p_i \ge P = m$ at the open storage positions $i \in S$ by conducting round trips between $w_{i,1}$ and v_i on aisle α_i. Since the resulting travel along cross aisle β_1 takes less than k time units, we have a total tour duration of less than $k\cdot\kappa + k$ time units.

Conversely, we assume that $f(\phi)$ possesses a feasible tour schedule with a makespan of less than $k\cdot(\kappa+1)$. Due to the dominating length $L = k\cdot(\kappa+1)+1$ of the vertical aisles, this tour does not use cross aisle β_2 and services a subset of storage positions $S \subseteq \{1,\dots,n\}$ fulfilling $\sum_{i\in S} c_i = \sum_{i\in S} p_i \ge m = P$. Thus, the total duration of all detours necessary for servicing the storage positions of set S is also less than $k\cdot(\kappa+1)$. Since for $j \in S$ the cyclical detour on the vertical aisle α_j takes $k\cdot w_j$ time units, we conclude that $\sum_{i\in S} w_i \le \kappa$. Thus, ϕ obtains a yes-certificate.

EC.3. Proof of runtime complexity of our branch&bound algorithm

This appendix provides the proof for Lemma 5.

We prove this lemma in two steps: First, we derive an upper bound on the number of partial solutions that are inserted into our min-heap during the enumeration process. Second, we specify the effort that is required for each branching step of a partial solution.

Due to Lemma 4, for each tuple $(\zeta^e,\zeta^d,\zeta^c)$ with $0 \le \zeta^e \le |E|$, $\zeta^d \in \{0,1,2\}^h$ and $\zeta^c \in \{-1,1,\dots,h\}^h$ there exists at most one dominance list of partial tours. As shown by Cambazard and Catusse (2018), the number of feasible tuples $(\zeta^e,\zeta^d,\zeta^c)$ is bounded by $\mathcal{O}(n\cdot h\cdot 7^h)$. Thus, we can obtain an upper bound estimate on the total number of partial tours of $\mathcal{O}(n\cdot h\cdot 7^h\cdot T\cdot m)$, because $\mathcal{O}(T\cdot m)$ is an upper bound on the length of each dominance list. This results from the fact that different non-dominated partial solutions ζ that are inserted into the same dominance list differ by the number of stored products ζ^s and the tour duration ζ^t.

During each branching step, we have to evaluate all possible child nodes. In the worst case, we cannot reduce the number of traversal modes to be considered. For a vertical aisle possessing $\tilde{n} \le n$ open storage positions, the number is at most $\frac{1}{2}\tilde{n}^2 + \frac{3}{2}\tilde{n} + 3 \in \mathcal{O}(n^2)$. Hence, we can derive an upper bound on the asymptotic total runtime of our branch&bound algorithm of $\mathcal{O}(n^3\cdot h\cdot 7^h\cdot T\cdot m)$. If it holds that $h \in \mathcal{O}(\log n)$, our runtime estimate amounts to $\mathcal{O}(n^3\cdot\log n\cdot 7^h\cdot T\cdot m)$ and is therefore pseudo-polynomial.

EC.4. Additional performance tests to explore the relation to the knapsack problem

This section extends the computational tests to explore the runtime of our branch&bound algorithm of Section 5.1. Our W-PCTSP contains a subproblem that is similar to the famous knapsack problem: Storage positions (equal to the products of knapsack) allow products to be stowed there (profit) but demand travel effort (costs), while a given number of products must be stowed in

total (similar to the capacity constraint of knapsack). Pisinger (2005) shows that specific knapsack instances require significantly more solution effort than others. Consequently, this appendix explores whether W-PCTSP instances that resemble hard knapsack instances are also a special challenge for our branch&bound algorithm.

Before we come to the two new data sets employed in this section, however, we would like to note that there are also important differences between both problems. First, the distances to be traveled in order to visit an additional storage position depend on the positions already chosen. In contrast to this, the weights and prices of knapsack products are constant. Moreover, due to Lemma 2, storage positions located along a vertical aisle are selected in a sequential manner, whereas products added to a knapsack are not restricted by such an arrangement.

Capacity interval $[c^{\min}, c^{\max}]$	Average total capacity	Average capacity of storage positions	Total travel time of tour	Computational time in seconds
$[1,1]$ uniform	150.0	1.0	8,715.4	0.2
$[1,2]$ uniform	223.8	1.5	4,549.2	11.2
$[1,3]$ uniform	299.9	2.0	3,348.0	12.7
$[1,5]$ uniform	450.6	3.0	2,130.0	4.0
$[1,10]$ uniform	799.2	5.3	1,504.6	1.1
$[1,15]$ uniform	1,196.3	8.0	939.6	0.4
$[1,20]$ uniform	1,565.9	10.4	848.0	<0.1
$[1,30]$ uniform	2,344.2	15.6	761.4	0.1
$[1,40]$ uniform	3,020.6	20.1	536.6	<0.1
$[1,50]$ uniform	3,810.0	25.4	463.0	<0.1

Table EC.1 **Runtime analysis of our branch&bound algorithm for instances with $n = 150$, $m = 150$, $v = 10$, $h = 5$ and stowing capacities that are uniformly drawn out of the interval $[c^{\min} = 1, c^{\max}]$ with $c^{\max} \in \{1, 2, 3, 5, 10, 15, 20, 30, 40, 50\}$.**

Due to the fact that the knapsack problem is binary $\mathcal{NP}$-hard (as is W-PCTSP), one important criterion for the hardness of knapsack instances is the size of the coefficients (see also Pisinger 2005). Since extended distances between storage positions do not effect the enumeration process of our branch&bound algorithm, the first novel benchmark test solely increases the stowing capacities at each storage position in order to obtain instances with larger coefficients. Specifically, we vary $c^{\max} \in \{1, 2, 3, 5, 10, 15, 20, 30, 40, 50\}$ and draw the capacity of each storage position randomly from interval $[c^{\min} = 1, c^{\max}]$ (with uniform distribution). For each setting, we report the average measured results of 10 instances. The impact of these values on the runtime of our branch&bound algorithm is summarized in Table EC.1. These results, however, rather replicate our previous finding of Section 5.1 in Figure 5(c): It is mainly the capacity ratio $\sum_{i=1}^{n} c_i/m$ that influences the runtime of our algorithm.

Therefore, we extend this first experiment by keeping the capacity ratios constantly equal to 4, i.e., after randomly drawing the stowing capacities of the open storage positions, we set the number of products to be stored to $m = \lfloor \sum_{i=1}^{n} c_i/4 \rfloor$. In this way, we are able to analyze the computational

Capacity interval $[c^{\min}, c^{\max}]$	Average total capacity	Average capacity of storage positions	m	Total travel time of tour	Computational time in seconds
$[1,1]$ uniform	150.0	1.0	37.0	1,826.4	0.8
$[1,2]$ uniform	223.8	1.5	55.6	1,837.0	1.2
$[1,3]$ uniform	305.2	2.0	75.8	1,822.6	1.6
$[1,5]$ uniform	451.9	3.0	112.6	1,805.8	2.4
$[1,10]$ uniform	839.4	5.6	209.6	1,825.2	3.0
$[1,20]$ uniform	1,584.0	10.6	395.6	1,818.4	3.7
$[1,30]$ uniform	2,243.2	15.0	560.3	1,819.4	3.6
$[1,40]$ uniform	3,076.9	20.5	768.8	1,730.0	2.8
$[1,50]$ uniform	3,776.0	25.2	943.6	1,812.2	3.7
$[1,100]$ uniform	7,594.9	50.6	1,898.3	1,785.6	3.0
$[1,1000]$ uniform	74,870.9	499.1	18,717.4	1,775.6	3.7

Table EC.2 Runtime analysis of our branch&bound algorithm for instances with fixed capacity ratio $\sum_{i=1}^{n} c_i/m = 4$, $n = 150$, $v = 10$, $h = 5$ and stowing capacities that are uniformly drawn out of the interval $[c^{\min} = 1, c^{\max}]$ with $c^{\max} \in \{1,2,3,5,10,15,20,30,40,50\}$.

performance for increasing coefficients, while the capacity ratios remain constant. The performance results of our branch&bound algorithms for these instances are summarized in Table EC.2. It can be observed that the impact of larger coefficients is existent, yet comparatively small. This is in line with our theoretical runtime analysis, where the complexity is mainly driven by the number of aisles.

Capacity interval $[c^{\min}, c^{\max}]$	Average total capacity	Average capacity of storage positions	Total travel time of tour	Computational time in seconds
$[1,10]$ correlated	714.9	4.8	1,842.2	579.7
$[1,10]$ uncorrelated	850.1	5.7	1,214.2	85.1
$[1,20]$ correlated	1,424.6	9.5	1,369.8	121.2
$[1,20]$ uncorrelated	1,577.9	10.5	741.2	6.0
$[1,30]$ correlated	2,134.2	14.2	1,117.4	31.8
$[1,30]$ uncorrelated	2,314.8	15.4	595.0	1.5

Table EC.3 Runtime analysis of our branch&bound algorithm for instances with $n = 150$, $m = 150$, $v = 10$, $h = 8$, $c^{\min} = 1$, and $c^{\max} \in \{10,20,30\}$ where the storage positions' stowing capacities and distances from the IO point are either correlated or uncorrelated

A second characteristic of hard-to-solve knapsack instances identified by Pisinger (2005) is the strong correlation of price and weight values. Transferred to our W-PCTSP, this means that storage positions farther away from the IO point offer more space for stowing products. In fact, this is indeed observed in many real-world instances. Naturally, storage positions closer to the IO point are preferred by stowers, so that here open storage space is relatively more scarce. To emulate this, we extend our data generator and derive stowing capacities of the open storage positions only after assigning them to their locations in the warehouse. Specifically, after deriving the distance d_i^{IO} of each storage position i from the IO point, the total range of distances are determined by $d_{\min}^{IO} = \min\{d_i^{IO} \mid i \in \{1,\dots,n\}\}$ and $d_{\max}^{IO} = \max\{d_i^{IO} \mid i \in \{1,\dots,n\}\}$. Then, the stowing capacities are derived as follows:

$$\forall i \in \{1,\dots,n\} : c_i = c^{\min} + \left\lfloor \frac{(d_i^{IO} - d_{\min}^{IO}) \cdot (c^{\max} - c^{\min})}{d_{\max}^{IO} - d_{\min}^{IO}} \right\rfloor. \tag{EC.9}$$

These instances, which we dub *correlated*, are then compared to *uncorrelated* instances, where the stowing capacities are drawn uniformly distributed from interval $[c^{\min}, c^{\max}]$. The performance results of our branch&bound procedure for these two types of instances are summarized in Table EC.3. These results confirm the effect known for knapsack instances: Strongly correlated coefficients lead to harder problem instance. For an interval of $[1, 10]$ for stowing capacities c_i, for instance, the average running time for uncorrelated instances is just 85.1 seconds compared to almost 10 minutes for correlated ones. We would like to note, however, that these instances are very challenging because of the $h = 8$ horizontal cross aisles. Hence, we come to the conclusion that even for correlated coefficients our branch&bound algorithm can be successfully applied in real-world warehouses.

EC.5. Specification of instance generator

This appendix details our instance generator that is used for all our computational tests. The generator receives the following parameters as its own input data: n (number of open storage positions), m (number of products to be stored), h (number of horizontal cross aisles), v (number of vertical picking aisles), $c^{\min}$ (minimum capacity at an open storage position), and $c^{\max}$ (maximum capacity at an open storage position). Given these parameters, each single instance is derived as follows: First, the time to completely traverse a cross aisle and a picking aisle, l_h and l_v, respectively, are randomly drawn with a uniform distribution out of interval $[500, 1000]$. Interpreting seven of our internal time units as one second, for instance, leads to a traversal duration of 94.75 seconds. At a typical stower walking speed of about 2.5kph, which reflects the slow-down effect of the cart, we have an average aisle length of approximately 65m. After fixing these lengths and setting the first and last aisle exactly l_h and l_v units apart, the positions of all intermediate aisles are determined by randomly drawing and sorting the respective x-coordinates of the vertical picking aisles and the y-coordinates of the horizontal cross aisles. When doing so, we ensure a minimum distance of $l^{d,\min} = 20$ time units between two neighboring aisles.

Subsequently, n open storage positions are assigned to the vertical picking aisles. This is done by drawing a random number out of the uniformly distributed interval $[1, v]$ for each open position. Then, for each vertical aisle, the assigned, lets say k, open positions are located by randomly drawing k numbers out of the uniformly distributed interval $[1, l_v - 1]$. After being sorted, these k numbers define, one by one, the y-coordinate of the assigned k storage positions located along the aisle. Note that the sorting of the drawn numbers and their iterative assignment ensures that higher-indexed storage positions obtain a larger y-coordinate. Consequently, along each vertical picking aisle, open storage positions are located in sequence of their indices. The IO point is positioned at the front cross aisle by randomly drawing an y-coordinate from the uniformly distributed interval $[1, l_h - 1]$. Finally, instance generation ends with the definition of the storage capacities of each open storage position. These capacities are randomly drawn out of the uniform interval $[c^{\min}, c^{\max}]$.