

A Space-Indexed Formulation of Packing Boxes into a Larger Box

Sam D. Allen, Edmund K. Burke, Jakub Mareček

*University of Nottingham, School of Computer Science
Jubilee Campus, Nottingham, NG8 1BB, UK*

Abstract

The problem of packing many small boxes into a single larger box underlies a number of cutting, packing, scheduling, and transportation applications. There are a number of heuristic solvers, but the progress in exact solvers, in general, and integer programming solvers, in particular, has been limited. Padberg [Math. Methods Oper. Res., 52(1):1–21, 2000] estimated his extension of the integer linear programming formulation of Chen et al. could cope with “about twenty boxes”. Our computational experiments confirm that the seemingly trivial decision whether twelve unit-cubes can be packed into a box with the unit-base and height eleven (“3D Pigeon Hole”), cannot be made within an hour by modern integer programming solvers using this formulation.

We present a new, “space-indexed”, linear programming relaxation, which often provides lower bounds within 1 percent of optimality, and makes it possible to solve instances of 3D Pigeon Hole problem with ten million boxes within an hour. Results of extensive computational tests of both formulations are reported.

1. Introduction

Multi-dimensional packing problems have a number of applications in cutting, packing, scheduling, and transportation. Problems in dimension three with rotations around combinations of axes in multiples of 90 degrees are of particular interest in many natural applications. Let us fix the order of six such allowable rotations in dimension three arbitrarily and define:

The CONTAINER LOADING PROBLEM (CLP): Given dimensions of a large box (“container”) $x, y, z > 0$ and dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$ with associated values $w \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0, 1\}^{n \times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \dots, n\}$ into the container with value $k = \sum_{i \in I} w_i$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

The VAN LOADING PROBLEM (VLP): Given dimensions of a large box (“van”) $x, y, z > 0$, maximum mass $p \geq 0$ it can hold (“payload”), dimensions of n small boxes $D \in \mathbb{R}^{n \times 3}$

Symbol	Meaning
n	The number of boxes.
H	A fixed axis, in the set $\{X, Y, Z\}$.
α	An axis of a box, in the set $\{1, 2, 3\}$.
$L_{\alpha i}$	The length of axis α of box i .
$l_{\alpha i}$	The length of axis α of box i halved.
D_H	The length of axis H of the container.
w_i	The volume of box i in the CLP.

Table 1: Notation used.

with associated values $w \in \mathbb{R}^n$, mass $m \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0, 1\}^{n \times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \dots, n\}$ into the container with value $k = \sum_{i \in I} w_i$ and mass $\sum_{i \in I} m_i \leq p$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

Both problems are NP-Hard even to approximate [4]. For example in branch-and-price-and-cut solvers for realistic models of transportation considering both weight and volume of vehicle load [6, 7], however, one needs to solve the van loading problem to optimality as the pricing subproblem. In scheduling, the problem corresponds to scheduling non-malleable parallel jobs, requiring a certain number of clock cycles, a certain number of processors, and a certain amount of another discrete resource. Although there are a number of excellent heuristic solvers, the progress in exact solvers for the Container Loading Problem has been limited, so far. Chen et al. [3] proposed an integer linear programming (ILP) formulation, which allowed for up to six boxes to be packed into a smaller box. Fasano [5] and Padberg [8] improved the formulation and suggested it could cope with “about twenty boxes”. We confirm these limits using ILOG CPLEX, Gurobi, and SCIP in Section 4. This observation motivates the rest of the paper, where we present a new space-indexed formulation providing strong linear programming relaxations. Throughout, we use the notation as described in Table 1.

2. Related Work

Chen et al. [3] introduced an integer linear programming formulation using the relative placement indicator:

$$\lambda_{ij}^H = \begin{cases} 1 & \text{if box } i \text{ precedes box } j \text{ along axis } H \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_{\alpha i}^H = \begin{cases} 1 & \text{if box } i \text{ is rotated so that axis } \alpha \text{ is parallel to fixed } H \\ 0 & \text{otherwise} \end{cases}$$

Using implicit quantification, it reads:

$$\max \sum_{i=1}^n \sum_H w_i \delta_{1i}^H \quad (1)$$

$$\text{s.t. } \sum_H \delta_{2i}^H = \sum_H \delta_{1i}^H = \sum_\alpha \delta_{\alpha i}^H \quad (2)$$

$$L_{1j(i)} \lambda_{j(i)i}^H + \sum_\alpha l_{\alpha i} \delta_{\alpha i}^H \leq x_i^H \leq \sum_\alpha (D_H - l_{\alpha i}) \delta_{\alpha i}^H - L_{1j(i)} \lambda_{ij(i)}^H \quad (3)$$

$$D_H \lambda_{ji}^H + \sum_\alpha l_{\alpha i} \delta_{\alpha i}^H - \sum_\alpha (D_H - l_{\alpha j}) \delta_{\alpha j}^H \leq x_i^H - x_j^H \quad (4)$$

$$\leq \sum_\alpha (D_H - l_{\alpha i}) \delta_{\alpha i}^H - \sum_\alpha l_{\alpha j} \delta_{\alpha j}^H - D_H \lambda_{ij}^H \quad (5)$$

$$\sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_H \delta_{1i}^H, \sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_H \delta_{1j}^H \quad (6)$$

$$\sum_H \delta_{1i}^H + \sum_H \delta_{1j}^H \leq 1 + \sum_H (\lambda_{ij}^H + \lambda_{ji}^H) \quad (7)$$

$$\sum_{i=1}^n \sum_H \left(\prod_\alpha L_{\alpha i} \right) \delta_{1i}^H \leq \prod_H D_H \quad (8)$$

$$\delta_{\alpha i}^H \in \{0, 1\}, \lambda_{ij}^H \in \{0, 1\} \quad (9)$$

$$L_{1i} \leq L_{2i} \leq L_{3i}, j(i) \text{ such that } L_{1j(i)} = \max\{L_{1j}\} \text{ for } 1 \leq i \neq j \leq n. \quad (10)$$

Padberg [8] has studied its properties. In particular, he identified the subsets of constraints with the integer property. Despite the interesting theoretical properties, modern integer programming solvers fail to solve instances larger than 10–20 boxes using this formulation, as evidenced by Table 2. This is far from satisfactory.

2.1. Extensions to the Formulation of Chen/Padberg

Arguably, the formulation could be strengthened by addition of further valid constraints. We have identified three classes of such constraints. Compound constraints stop combinations of certain boxes being placed next to each other if they would violate the domain constraint. For example:

$$\delta_{ki}^H + \delta_{mj}^H + \lambda_{ij}^H \leq 2 \quad \text{if } l_{ki} + l_{mj} > D_H \quad \forall k, m \in \{1, 2, 3\}, H \in \{X, Y, Z\}$$

$$\text{and } 1 \leq i \neq j \leq n \quad (11)$$

The example for 2 boxes can be easily extended to 3 or more boxes “in a row.” One can also attempt to break symmetries in the problem. If boxes i and j (where $i < j$) are identical (i.e. $L_{\alpha i} = L_{\alpha j} \forall \alpha$):

$$\lambda_{ij}^H = 0 \text{ and } \sum_H \delta_{1i}^H \geq \sum_H \delta_{1j}^H \quad (12)$$

Furthermore, if a box has two or more sides of the same length then we can limit the rotations. We define function f , which provides a canonical mapping of $\{X, Y, Z\}$ to

$\{1, 2, 3\}$:

$$\sum_H (f(H) \cdot \delta_{ki}^H) \geq \sum_H (f(H) \cdot \delta_{mi}^H) \quad \text{if } l_{ki} = l_{mi} \quad \forall 1 \leq i \leq n \text{ and } 1 \leq k < m \leq 3 \quad (13)$$

Nevertheless, whilst the addition of these constraints improves the performance somewhat, the formulation remains impractical.

3. A Space-Indexed Formulation

We hence propose a new formulation, where the large box is discretised into units of space, whose dimensions are given by the largest common denominator of the respective dimensions of the small boxes. The small boxes are grouped by their dimensions into types $t \in \{1, 2, \dots, n\}$. A_t is the number of boxes of type t available. The formulation uses the space-indexed binary variable:

$$\mu_{x,y,z}^t = \begin{cases} 1 & \text{if a box of type } t \text{ is placed such that its lower-back-left} \\ & \text{vertex is at coordinates } x, y, z \\ 0 & \text{otherwise} \end{cases}$$

Without allowing for rotations, the formulations reads:

$$\max \sum_{x,y,z,t} \mu_{xyz}^t w_t \quad (14)$$

$$\text{s.t. } \sum_t \mu_{xyz}^t \leq 1 \quad \forall x, y, z \quad (15)$$

$$\mu_{xyz}^t = 0 \quad \forall x, y, z, t \text{ where } x + L_{1t} > D_X \text{ or} \\ \text{or } y + L_{2t} > D_Y \text{ or } z + L_{3t} > D_Z \quad (16)$$

$$\sum_{x,y,z} \mu_{xyz}^t \leq A_t \quad \forall t \quad (17)$$

$$\sum_{x',y',z'} f(x, y, z, x', y', z', t) \leq 1 \quad \forall x, y, z, t \quad (18)$$

where

$$f(x, y, z, x', y', z', t) = \begin{cases} \mu_{x'y'z'}^t & \text{if } x \leq x' \leq x + L_{1t} \text{ and} \\ & y \leq y' \leq y + L_{2t} \text{ and} \\ & z \leq z' \leq z + L_{3t} \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

The constraints are very natural: No region in space may be occupied by more than one box type (15), boxes must be fully contained within the container (16), there may not be more than A_t boxes of type t (17), and boxes cannot overlap (18). There is one non-overlapping constraint (18) for each discretised unit of space and type of box. In order to support rotations, new box types need to be generated for each allowed rotation

and linked via set packing constraints similar to Constraint 17. In order to extend the formulation to the van loading problem, it suffices to add the payload capacity constraint:

$$\sum_{x,y,z,t} \mu_{xyz}^t m_t \leq p \tag{20}$$

3.1. Adaptive Discretisation

In order to reduce the number of regions of space, and thus the number of variables in the formulation, a sensible space-discretisation method should be employed. The greatest common divisor (GCD) reduction can be applied on a per-axis basis, finding the greatest common divisor between the length of the container for an axis and all the valid lengths of boxes that can be aligned along that axis and scaling by the inverse of the GCD. This is trivial to do and is useful when all lengths are multiples of each other, which may be common in certain problems. In other problems this may not reduce the number of variables at all. In this case we can apply a non-linear space-discretisation approach. To do this we use the same values as before on a per-axis basis, i.e. the lengths of any box sides that can be aligned along the axis. We then use dynamic programming to generate all valid locations for a box to be placed. For example, given an axis of length 10 and box lengths of 3, 4 and 6, we can place boxes at positions 0, 3, 4, 6, and 7 (8, 9 and 10 are of course also possible, but no length is small enough to still lie within the container if placed at these points.) This has reduced the number of regions along that axis from 10 to 5. An improvement on this scale may not be particularly common in practice, though the approach can help where the GCD is 1. It is obvious that this approach can be no worse than the GCD method at discretising the container. This also adds some implicit symmetry breaking into the model.

4. Computational Experience

We have tested the formulations above on three sets of instances:

- 3D Pigeon Hole Problem instances, Pigeon- n , where $n + 1$ unit cubes are to be packed into a container of dimensions $(1 + \epsilon) \times (1 + \epsilon) \times n$.
- SA and SAX datasets, which are used to test the dependence of solvers' performance on parameters of the instances, notably the number of boxes, heterogeneity of the boxes, and physical dimensions of the container. There is 1 pseudo-randomly generated instance for every combination of container sizes ranging from 5–100 in steps of 5 units cubed and the number of boxes to pack ranging from 5–100 in steps of 5. The SA datasets are perfectly packable, i.e. are guaranteed to be possible to load the container with 100% utilisation with all boxes packed. The SAX are similar but have no such guarantees; the summed volume of the boxes is greater than that of the container.
- Van-loading instances, generated so as to be similar to those used in pricing sub-problems of branch-and-cut-and-price approaches to vehicle routing with both load weight and volume considerations [6, 7]. The van-loading and Van-loading-X instances use containers of 10x10x30 and 10x10x50 units, respectively, representing

the approximate ratios of a small commercial van and a larger freight truck. 2, 4 or 6 out of 6 pseudo-randomly chosen orientations are allowed. The load weights and knapsack values are generated pseudo-randomly and independently of the volume of the small boxes. 10 instances were generated for each class of problem.

All the tests were performed on a 64-bit computer running Linux, which was equipped with 2 quad-core processors (Intel Xeon E5472) and 16 GB memory. The solvers tested were IBM ILOG CPLEX 12.2.0, Gurobi Solver 4.0, and SCIP 2.0.1 [1] with CLP as the linear programming solver. Instances are available¹ on-line.

For the 3D Pigeon Hole Problem, results obtained within one hour using the three solvers and the Chen/Padberg formulation are shown in Table 2, while Table 3 compares the results of Gurobi Solver on both formulations. None of the solvers managed to prove optimality of the incumbent solution for twelve unit-cubes within an hour using the Chen/Padberg formulation, although the instance of linear programming (after pre-solve) had only 616 rows and 253 columns and 4268 non-zeros. In contrast, the space-indexed formulation allowed Gurobi Solver to solve Pigeon-10000000 within an hour, where the instance of linear programming had 10000002 rows, 10000000 columns, and 30000000 non-zeros.

For the SA and SAX datasets, Figures 1–2 summarise solutions obtained within an hour using the Chen/Padberg formulation, the space-indexed formulation, and a metaheuristic approach described by Allen et al. [2]. In the case of the SAX dataset, the volume utilisation is given with respect to the tightest upper bound found by any of the solvers. Finally, for the van-loading instances, the results of Gurobi Solver with the space-indexed model after one hour can be seen in Table 4. As there were 10 instances tested for each class of problem, the mean number of boxes, number of unique box types and mean gap between solution value and bound are also given.

Overall, the space-indexed relaxation provides a particularly strong upper bound. The mean integrality gap, or the ratio of the difference between root linear programming relaxation value and optimum to optimum has been 10.49 % and 0.37 % for the Chen/Padberg and the space-indexed formulation, respectively, on the SA and SAX instances solved to optimality within the time limit of one hour. The mean gap, or the ratio of the difference between linear programming relaxation value and value of the best solution found to the bound, left after an hour on Van-loading instances was 1.5%. This is encouraging, albeit perhaps not particularly surprising, as similar discretised relaxations proved to be very strong in scheduling problems corresponding to one-dimensional packing [11, 12, 9] and can be shown to be asymptotically optimal for various geometric problems both in dimensions two [10] and higher [13] dimensions.

5. Conclusions

We have presented a space-indexed linear programming formulation of the container loading problem with a pseudo-polynomial numbers of variables. The space-indexed formulation provides very strong bounds, and hence performs particularly well on instances, where it can be solved. Future work could focus on the development of solvers specific to

¹ <http://www.cs.nott.ac.uk/~sda/pigeon>

Table 2: The performance of various solvers on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg formulation. “–” denotes that optimality of the incumbent solution has not been proven within an hour.

	Time (s)		
	Gurobi 4.0	CPLEX 12.2.0	SCIP 2.0.1 + CLP
Pigeon-01	< 1	< 1	< 1
Pigeon-02	< 1	< 1	< 1
Pigeon-03	< 1	< 1	< 1
Pigeon-04	< 1	< 1	< 1
Pigeon-05	< 1	< 1	3.3
Pigeon-06	< 1	< 1	37.9
Pigeon-07	1.5	3.6	779.3
Pigeon-08	7.4	25.6	–
Pigeon-09	88.6	398.4	–
Pigeon-10	1381.4	–	–
Pigeon-11	–	–	–
Pigeon-12	–	–	–

Table 3: The performance of Gurobi 4.0 on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg and the space-indexed formulations. “–” denotes no integer solution has been found.

	Time (s)	
	Chen/Padberg	Space-indexed
Pigeon-01	< 1	< 1
Pigeon-02	< 1	< 1
Pigeon-03	< 1	< 1
Pigeon-04	< 1	< 1
Pigeon-05	< 1	< 1
Pigeon-06	< 1	< 1
Pigeon-07	1.5	< 1
Pigeon-08	7.4	< 1
Pigeon-09	88.6	< 1
Pigeon-10	1381.4	< 1
Pigeon-100	–	< 1
Pigeon-1000	–	1.0
Pigeon-10000	–	1.8
Pigeon-100000	–	4.2
Pigeon-1000000	–	45.1
Pigeon-10000000	–	664.0
Pigeon-100000000	–	–

Table 4: The performance of the space-indexed model on the Van-loading and Van-loading-X instances after one hour. Each row represents 10 pseudo-randomly generated instances.

Dataset	Box Types	Boxes (Mean)	Gap (Mean)
Van-loading-01	3	78	0.10%
Van-loading-02	5	92	0.31%
Van-loading-03	8	81	0.42%
Van-loading-04	10	73	0.32%
Van-loading-05	12	73	0.09%
Van-loading-06	15	70	0.42%
Van-loading-07	20	68	0.50%
Van-loading-08	30	71	0.99%
Van-loading-09	40	69	1.26%
Van-loading-10	50	73	1.10%
Van-loading-X-01	3	133	0.16%
Van-loading-X-02	5	155	0.16%
Van-loading-X-03	8	132	0.56%
Van-loading-X-04	10	119	0.87%
Van-loading-X-05	12	123	0.29%
Van-loading-X-06	15	115	0.36%
Van-loading-X-07	20	116	0.30%
Van-loading-X-08	30	122	0.29%
Van-loading-X-09	40	119	0.18%
Van-loading-X-10	50	121	0.28%

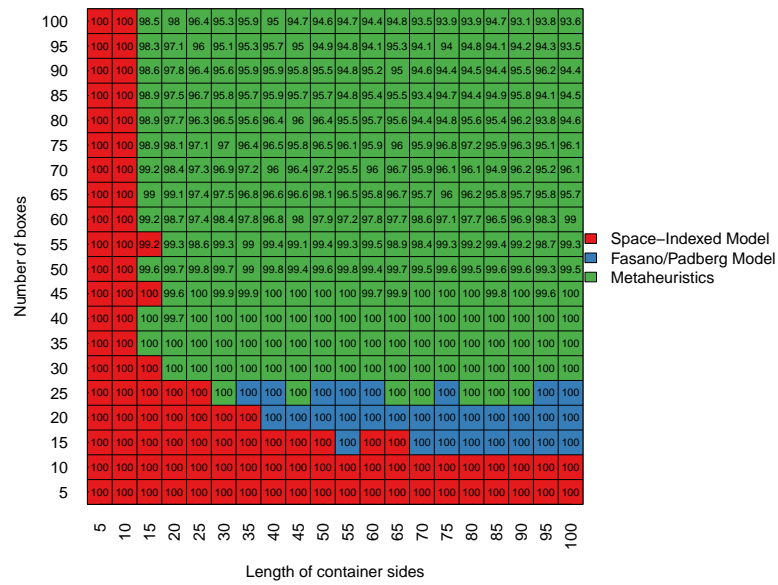


Figure 1: The best solutions obtained within an hour per solver per instance from the SA dataset. Each square represents 1 pseudo-randomly generated instance. The number is the volume utilisation (in percent) with the tight upper bound of 100.

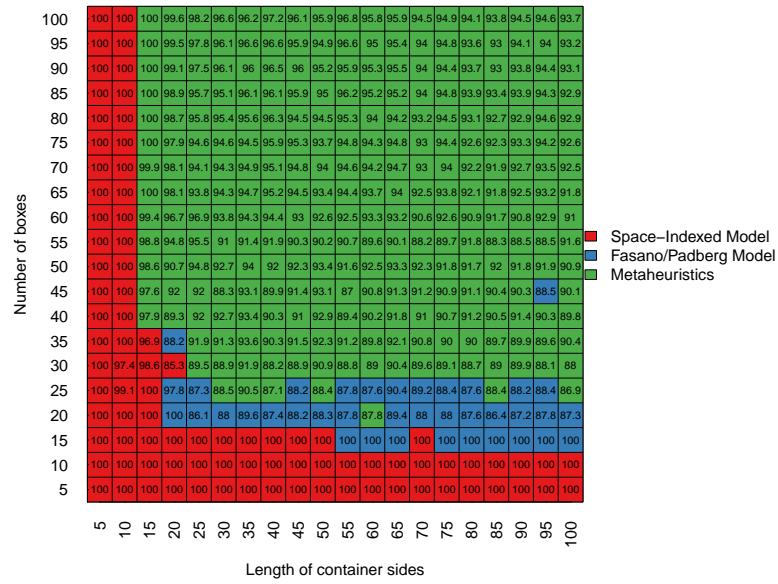


Figure 2: The best solutions obtained within an hour per solver per instance from the SAX dataset. Each square represents 1 pseudo-randomly generated instance. The number displayed is $100(1 - s/b)$ for solution with value s and upper bound b .

the relaxation. One could either implement a matrix-free interior point method exploiting the structure of the relaxation, or employ a column generation schema. The Dantzig-Wolfe decomposition could, perhaps, divide the problem into sub-problems pertaining to each type of box (18) and a master problem enforcing the set-packing constraint (15). Another interesting direction for future research is automated reformulations. Just as modern integer programming solvers often fail on small instances of the Chen/Padberg formulation, they fail on certain formulations of scheduling problems. Arguably, one could try to discretise such relaxations automatically, starting with the identification of what variables play the role of relative position in time or space. This could improve the performance of general-purpose integer linear programming solvers considerably.

References

- [1] T. Achterberg. SCIP: solving constraint integer programs. *Math. Program. Comput.*, 1(1):1–41, 2009.
- [2] S. D. Allen, E. K. Burke, and G. Kendall. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, 209(3):219 – 227, 2011.
- [3] C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.
- [4] M. Chlebík and J. Chlebíková. Hardness of approximation for orthogonal rectangle packing and covering problems. *J. Discrete Algorithms*, 7(3):291–305, 2009.
- [5] G. Fasano. Cargo analytical integration in space engineering: A three-dimensional packing model. In T. A. Ciriani, S. Gliozzi, E. L. Johnson, and R. Tadei, editors, *Operational Research in Industry*, pages 232–246. Purdue University Press, 1999.
- [6] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, 2006.
- [7] M. Lübbecke. Personal communication, August 27th, 2010.
- [8] M. Padberg. Packing small boxes into a big box. *Math. Methods Oper. Res.*, 52(1):1–21, 2000.
- [9] Y. Pan and L. Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Math. Program.*, 110(3, Ser. A):543–559, 2007.
- [10] C. H. Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM J. Comput.*, 10(3):542–557, 1981.
- [11] J. P. Sousa and L. A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54:353–367, 1992.
- [12] J. M. van den Akker. *LP-based solution methods for single-machine scheduling problems*. Technische Universiteit Eindhoven, Eindhoven, 1994. Dissertation.
- [13] E. Zemel. Probabilistic analysis of geometric location problems. *SIAM J. Algebraic Discrete Methods*, 6(2):189–200, 1985.