Schedae Informaticae Vol. 27 (2018): 9–17
doi: 10.4467/20838476SI.18.001.10406

Deep learning-based initialization for object packing

Maciej Wołczyk
Faculty of Mathematics and Computer Science
Jagiellonian University, Łojasiewicza 6, 30-348 Kraków, Poland
https://orcid.org/0000-0002-3933-9971

Abstract. One of the most important optimization tasks in the industry at the current time is the object packing problem. Although several methods have been developed for the purpose of solving it, they are usually only able to optimize placement locally and as such are heavily dependent on the choice of the initial setting – hence the need for trying out multiple possible starting points, which impacts algorithm running time. In this paper we present a neural network-based model which provides sensible starting points in a linear time.

Keywords: cutting & packing, optimization, object packing problem, phi-functions, deep learning

1. Introduction

Cutting & packing (C&P) is one of the most important optimization problems which naturally appears in the industry. The task is to minimize waste when placing a given set of objects in a container (often of variable dimensions) or, equivalently, cutting given shapes from a container (which represents e.g. a piece of cloth). We focus on the two-dimensional cutting stock problem, where we want to fit a set of objects in a container of variable size without any objects overlapping. In particularly difficult version of this problem shapes we want to fit are irregular (not necessarily rectangles or circles).

There are numerous real-world examples of this problem – e.g. cutting irregular shoe parts from leather roll [10] or cutting plate parts for inner frameworks of a ship

Received: 25 September 2018 / Accepted: 20 October 2018

[8] in such a way to minimize waste. Nevertheless, there are relatively few research papers concerning cutting & packing with irregular shapes, compared to other C&P problems [2], which is the motivation of our work. As the C&P problems are known to be NP-complete, it is infeasible to solve them analytically – instead heuristics are commonly used. In case of packing irregular objects, many methods focus on local optimization and are highly dependent on the initial solution [5].

Our main hypothesis was that we can use deep learning-based methods to produce an initial solution which is easier to optimize using a local optimizer than a random solution. Our experiments show that this is true – 95% of examples initialized with our method converged to a correct solution within 200 steps of a local solver. This was true only for 70% of randomly initialized examples. Examples initialized with our model needed on average three times fewer steps to converge than randomly initialized examples.

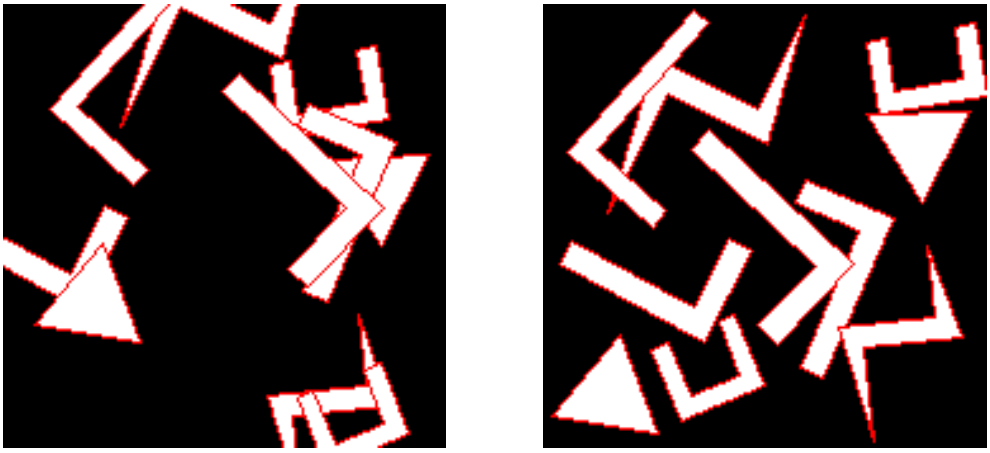
Our approach provides an initial solution for this optimization problem in a linear time and does not depend on any prior placement of the objects. The model can be optimized for a particular number or type of shapes to suit the properties of a specific given problem. This can be done by generating specific training examples – which is not costly, as the objects do not have to be manually labeled in any way. For example, when using our model for garment industry-specific problem, we could generate training examples including objects in shapes of trousers, shirts, dresses etc. since only such items will be then encountered in practice.

2. Theoretical model

We approach the object packing problem from a deep learning perspective. Although there have been many methods based on evolutionary algorithms [6], machine learning-based solutions are rare. Even research which does use such methods for the object packing problem, does not consider the objects of irregular shapes [7] – only regular, easier to work with shapes have been studied. We believe that rapid development of models using neural networks in recent years allows for refinement of currently used heuristics. In this paper we lay the groundwork for such model with hopes of further developing deep learning algorithms in optimization problems. Since prior research in this direction is very scarce, we apply several simplifications to the task to check if deep learning methods are even viable for this type of problems.

We consider a problem of fitting a set of given objects into a square container of fixed size. The task is completed if all objects are fully contained in the container and none of them are overlapping. This is a simplification of more difficult optimization problems (where the size of the container could be a variable to minimize). In our approach, we train an initialization network which is able to propose a good initial placement for a given set of objects. The initial solution is then fine-tuned using a gradient-based local optimization method.

Let O be the set of objects we want to fit into a container R of given, fixed dimensions. For each given object $o_i \in O$ initialization network will output the tuple



(a) **(b)**

Figure 1. Example of (1a) a bad random initialization for object packing problem and (1b) result after 200 steps of local optimization. Local optimizer cannot find a solution which satisfies the constraints – objects are still overlapping. The goal of this work is to build a model which provides better initial solutions, i.e. object placements for which the local optimizer needs fewer steps to find a correct solution.

$(\mathrm{u_i}, \mathrm{v_i}, \phi_\mathrm{i})$, which correspond respectively to horizontal and vertical shift (in pixels), and rotation angle (in radians). Object's center (point around which we rotate) is denoted by $c_i = \begin{bmatrix} c_i^1 \\ c_i^2 \end{bmatrix}$. After transformation object takes form of:

$$\hat{o}_i = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} o_i^1 - c_i^1 \\ o_i^2 - c_i^2 \end{bmatrix} + \begin{bmatrix} c_i^1 + u \\ c_i^2 + v \end{bmatrix} \tag{1}$$

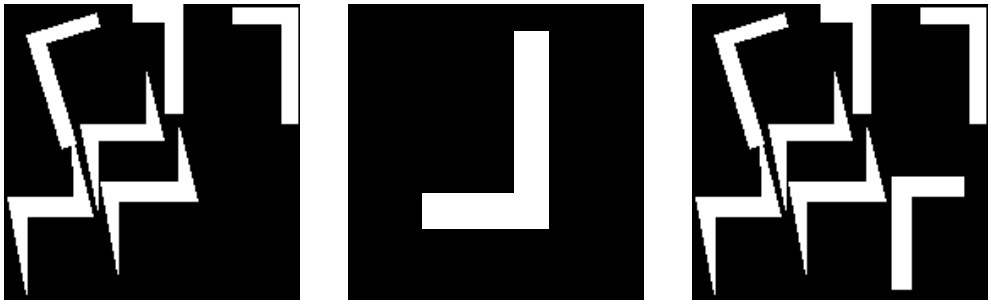
The transformed objects $\hat{O} = (\hat{o}_1, \ldots, \hat{o}_n)$ are the initial solution for the local optimization problem. A metric is needed to evaluate how good is this solution and, in particular, tell if the constraints are satisfied. We use phi-functions [3] for this purpose. Value of phi-function Φ^{AB}, which roughly approximates Euclidean distance between objects A and B, must fulfill the following conditions:

$$\begin{cases} \Phi^{AB} > 0, & \text{if A and B are not overlapping} \\ \Phi^{AB} = 0, & \text{if A and B are touching} \\ \Phi^{AB} < 0, & \text{if A and B are overlapping} \end{cases} \tag{2}$$

For example, a phi-function for two rectangles R_1, R_2 with centers in points (x_i, y_i) and sides of length a_i, b_i is given by formula:

$$\phi^{RR} = \max\left((|x_1 - x_2| - \frac{a_1 - a_2}{2}), (|y_1 - y_2| - \frac{b_1 - b_2}{2})\right) \tag{3}$$

In this paper we focus mainly on non-convex polygons, but our methods could be easily extended to all objects described in [3] or subsequent [4]. Since phi-functions



(a) **(b)** **(c)**

Figure 2. Inputs of the initialization network and an image based on its output. (2a) and (2b) are image inputs to the model, with the first being the container placement generated in previous steps and the second being the object to place in the container. The model then outputs coordinates and the rotation angle of the new object. (2c) shows the object figure placed according to output given by the model. This image will be used as input in the next step.

can always be computed via linear and quadratic formulas without radicals [3] and are differentiable, we can use gradient-based methods to find better placements of objects. In particular, we will use phi-functions for calculating the value of loss function of our model and for improving the initial solution until it satisfies all constraints.

Given our initial solution $\hat{O}$ we use a local solver which iteratively changes positions and rotations of $\hat{O}$ using gradient-based minimization of the loss

$$L = -\sum_{k,l,k\neq l} \min(\Phi(\hat{o}_k, \hat{o}_l), 0) - \sum_{k} \min(\Phi(\hat{o}_k, R^*), 0) \tag{4}$$

where the first term corresponds to overlap between objects and the second term checks if all objects are fully placed inside the container R, with A^* being the complement of A. The local solver minimizes this loss until all the constraints are satisfied ($L = 0$).

It is worth noting that although equation (2) is always true, the values of phi-functions approximate the Euclidean distance very roughly – e.g. when Euclidean distance between objects A and B is bigger than between objects A and C, the corresponding phi-functions Φ^{AB}, Φ^{AC} may be equal. This is not a serious issue for our methods, since we do not need a precise distance estimator and rather a way to maximize it.

We propose an iterative initialization network, which provides the initial solutions $\hat{O}$, by inserting the objects in the container one at the time. At each step i of the initialization network forward pass, the network is given an image of the object I_o^i we want to insert into the environment and an image of the objects already placed in the environment I_p^i. The model outputs the transformation (u_i, v_i, ϕ_i) for the new object, which is then inserted into the image. This new image becomes the input for the next step I_p^{i+1} along with a new object I_o^{i+1} from the input set. Example of a single step is presented in Figure 2. After N steps the model will have produced a transformation

Figure 3. Example of input to the model – 10 images of objects coming in 6 different types of shapes and varying in height or width. We want to fit them in a container of a fixed size without any of the objects overlapping.

for each object in the input. Thus the initial solution $\hat{O}$ for the object-placement problem is obtained. As such, the complexity of this method is $O(n)$.

During the training we minimize the modified truncated mean phi-function loss using the solution $\hat{O}$:

$$L = -\sum_{k,l,k\neq l} \min(\Phi(\hat{o}_k, \hat{o}_l) - \alpha, 0) - \sum_{k} \min(\Phi(\hat{o}_k, R^*) - \alpha, 0) \tag{5}$$

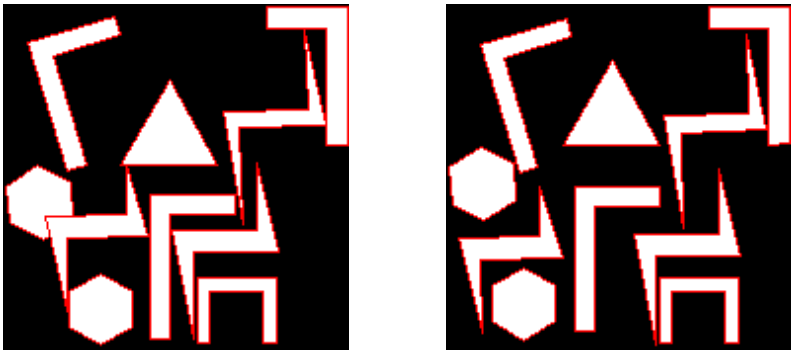
The loss is almost the same as equation (4), but includes α which is a hyperparameter representing the tolerance of our loss. If $\alpha < 0$, then slight overlapping is not penalized and if $\alpha > 0$, then we penalize the model if there is not some distance left between the objects.

The choice of an iterative model has several benefits. If we used a model that places all the objects at once, it would only accept problems with N objects, due to neural networks limitations. On the other hand our iterative model can be used for problems with any numbers of objects. The other possible choice was to use recurrent neural network, however after a few disappointing experiments with RNNs we decided to use the simpler, iterative model.

3. Experiments

To evaluate our method, we generated several data sets, which consist of irregular non-convex objects (Figure 3). We prepared six types of shapes for our objects – every object of given shape may also have varying width, height or thickness. We included shapes that are clearly non-convex to show that our model is able to deal with irregularities and in particular stack similar objects. This cannot be done with models which approximate objects with large rectangles.

For each training step we generate N black-and-white images of size (64px, 64px) of objects I_o that are supposed to be placed into the two-dimensional square of fixed



(a) Initial figure placement given by the initialization network

(b) Final figure placement after local optimization

Figure 4. Example of local optimization algorithm results

size. Since generation of those images is very quick and straightforward, we do not prepare a training dataset beforehand and instead generate a new batch of images before every training step.

We decided to represent the objects with images, with other viable choices being a list of vertices coordinates or a list of linear coefficients representing each edge. The image is the most flexible format, as the other two would not be usable if we chose circular objects. Additionally, using images means that during inference we can provide initial placement for objects which do not have defined phi-functions (however we would not be able to calculate loss for such objects). Images were generated using the Pillow library for Python language[1].

The architecture of the initialization network consists of two separate convolutional networks with four layers (one of which processes the image of the container, and the other processes the image of the new object) and a fully connected network with three layers. The outputs of convolutional networks are concatenated and passed to the fully connected network, which outputs three numbers. The first two represent the shift u_i, v_i. The third, z_i is passed through sigmoid to obtain the rotation angle:

$$\phi_i = 2\pi(\text{sigmoid}(z_i) - 0.5) \tag{6}$$

We used ReLU activations in all layers except the output layer. All calculations are performed using Python programming language with the PyTorch library[2].

We tested our model on a task of placing 10 randomly generated objects into a container of size (175px, 175px). We trained our model for 5000 batches of size 64. Hyperparameter α in equation (5) was set to 0.25. Every 100 batches we evaluated our model on a test set of 256 examples which were not used for training. The small size of the test set was dictated by the duration of local optimization which took much longer than the training itself.

We compared our model to a simple baseline which puts objects randomly into the container, with the position of objects' centers being sampled from $\text{unif}(0, \text{c})$, where c

[1] https://python-pillow.org/
[2] https://pytorch.org/

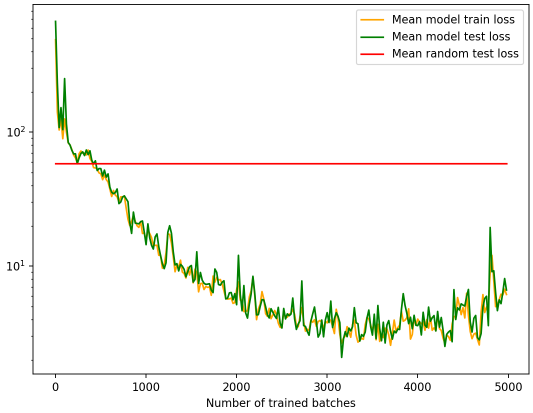
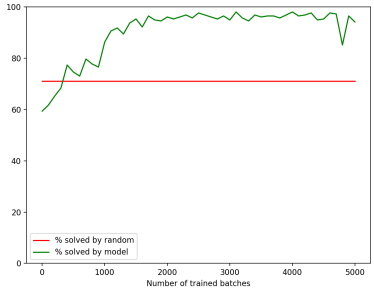


Figure 5. Training and test loss from equation (5) on initial solutions. Logarithmic scale provided for better readibility. Values are averaged over 20 batches at every point to reduce noise.

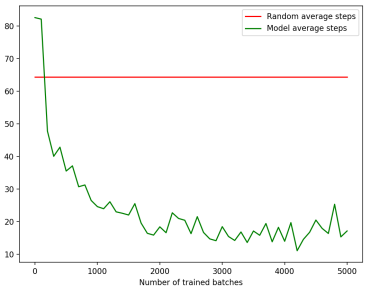
is the length of the side of the square-shaped container and the rotation angle being sampled from $\text{unif}(0, 2\pi)$. The baseline samples 1000 such placements and then picks the one with the lowest phi-function loss as defined in equation (4). The process of testing and evaluating different solutions takes about 500 times longer than forward pass of our network – which means that our method is quicker. The difficulty of the problem (number of polygons in a set of objects and the size of the container) was dictated by performance of our baseline model.

The average training and test losses are presented in Figure 5. Our model was able to steadily minimize the loss function. At the end of the training the average value of the loss function was 5.53 which corresponds to very slight overlapping such as that in Figure 4a.

We also evaluated solutions given by the model on the basis of how long it takes a local solver, which minimizes the loss given by equation (4), to converge to a solution which satisfies all constraints (i.e. $L = 0$). The local solver was implemented by us using PyTorch auto-grad mechanism which allowed us to perform gradient descent on the loss function. After training, our model's initialization converged much more quickly than random initialization – the average number of steps needed for convergence was three times lower for our model as may be seen in Figure 6b. We averaged only over those examples which converged before 200 steps of local solver to avoid running local optimization indefinitely – percentage of converged examples is plotted in Figure 6a. Convergence rate is around 95% for our model's initialization versus 70% for random initialization.



(a) Percentage of solutions which converged before 200th local optimization steps.



(b) Average number of local optimization steps the local solver had to perform before converging to solution which satisfies all constraints. Examples which did not converge before 200th local optimization step were not included in this average.

Figure 6. Metrics produced by using local solver on initial solutions provided by the initialization network and a random baseline. Evaluation of model's initialization was performed every 100 iterations of the training loop. Random baseline was evaluated only once as it does not change during the training.

4. Conclusions and further work

In this paper we have presented a deep learning-based method for finding initial solution for the object packing problem. In comparison with a random baseline our model shows definite superiority on all of the tested metrics. These results may be used as a proof that deep learning work could potentially be applied in more difficult versions of this problem, although further work is required.

Another useful aspect of our work is implementation of phi-functions-based local solver using deep learning libraries. Thanks to automatic gradient calculation which can parallelize computations with GPU, the computations can be sped up which means arriving to the solution more quickly.

There are various ways to improve on our work. Firstly, more baselines should be tested, especially those used currently in the field of optimization – unfortunately, this requires more time and computing power. Other experiments with different metrics could be performed – e.g. we could make the size of the container variable and ask the model to minimize size, while keeping all the current constraints.

Secondly, more sophisticated models could be tested. Although we have briefly experimented with recurrent neural network models to no avail, usage of Seq2Seq architecture [9] and attention mechanism [1] could fix these issues. Reinforcement learning techniques could also be tested, as such methods were used with success with other optimization problems [7].

5. References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Julia A Bennell and Jose F Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397–415, 2008.

[3] N Chernov, Yu Stoyan, and Tatiana Romanova. Mathematical model and efficient algorithms for object packing problem. *Computational Geometry*, 43(5):535–553, 2010.

[4] N Chernov, Yu Stoyan, Tatiana Romanova, and Aleksandr Pankratov. Phi-functions for 2d objects formed by line segments and circular arcs. *Advances in Operations Research*, 2012, 2012.

[5] A Miguel Gomes and José F Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829, 2006.

[6] E Hopper and B Turton. A genetic algorithm for a 2d industrial packing problem. *Computers & Industrial Engineering*, 37(1-2):375–378, 1999.

[7] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.

[8] Hiroyuki Okano. A scanline-based algorithm for the 2d free-form bin packing problem. *Journal of the Operations Research Society of Japan*, 45(2):145–161, 2002.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[10] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European journal of operational research*, 183(3):1109–1130, 2007.