Using the Gradient Projection Method in an Intelligent Cutting and Packing System

Mykola Popov[1,†], Oleksii Kartashov[1,*,†]

[1]National Aerospace University "Kharkiv Aviation Institute", 17 Vadym Manko St, Kharkiv, 61070, Ukraine

Abstract

This paper considers the problem of optimally placing circles in a rectangular region within an intelligent cutting and packaging system. A mathematical model and analysis of this problem are presented. A Python implementation of the gradient projection method for finding a local minimum in this problem is proposed, taking into account the specific features of the problem. The proposed implementation is compared with methods from the scipy.optimize Python library for constrained nonlinear optimization. It is shown that the authors' proposed implementation is faster.

Keywords

Local minimum, gradient projection method, packing, cutting, circle, optimization

1. Introduction

1.1. Motivation

The problem considered in this paper belongs to the class of optimal synthesis of spatial configurations[1], more specifically to Cutting and Packing (CP) problems. The relevance of CP problems is evidenced by a significant number of scientific publications related to this field. The review for the classification of these problems alone contains more than 100 references[2]. These problems belong to the class of NP-hard problems, which means that the search for effective methods for their solution is and will remain a relevant task.

One of the classic problems of spatial configurations is the problem of packing circles [3, 4, 5, 6, 7, 8]. It consists in placing a set of circles inside a rectangular area of fixed height (strip) so as to minimize its length. This type of problem has practical applications, particularly in the following areas: optimizing the placement of carbon nanotubes to improve the reliability and performance of microchips [9, 10, 11].

This work examines the effectiveness of using one implementation of the gradient projection method to find a local minimum and compares it with the built-in methods of the Python language.

1.2. State of the art

Like many other spatial optimization problems, circle packing problem is available on the packomania website[12]. Here it is referred to as the Circle Open Dimension Problem (CODP). The website updates information and provides current data on which algorithms are best in this field. At present, several of them can be highlighted.

The article [3] proposes a heuristic algorithm based on a combination of beam search, binary search, and a multi-start strategy. The method is not purely stochastic, although the multi-start strategy introduces an element of diversification to avoid local optima.

Beam search is the main part of the algorithm that implements local search. It is a shortened version of decision tree search. At each level of the tree, only a limited number of the best nodes (beam width)

ProfIT AI'25: 5th International Workshop of IT-professionals on Artificial Intelligence, October 15–17, 2025, Liverpool, UK

*Corresponding author.

†These authors contributed equally.

m.v.popov@khai.edu (M. Popov); alexeykartashov@gmail.com (O. Kartashov)

0009-0001-0863-6551 (M. Popov); 0000-0002-6282-553X (O. Kartashov)

are selected for further branching, and the rest are discarded. The potential of each node is evaluated using a greedy strategy called Minimum Local-Distance Position (MLDP). This strategy consists of sequentially placing each subsequent circle in a position where the distance to already placed circles or to the edges of the strip is minimal. This ensures dense packing at the local level.

The article [4] proposes Iterated Tabu Search, a metaheuristic method that optimizes the objective function in a limited area. However, it can fall into local optimum traps, so the method is combined with a perturbation operator to search for the global optimum.

The Tabu Search (TS) procedure is a local search that uses prohibition rules and convergence criteria to prevent search loops. After finding a local optimum, a perturbation operator is used to obtain a new solution, which in turn is optimized again using TS and accepted if it is better than the previous one.

Research in this area continues. For example, the DCPACK (Discretized-Space Circle Packer Algorithm) algorithm was proposed [5], which was used for both tape packing and circle packing problems. The main idea of the method is to discretize the continuous space of the container into small cells and sequentially solve two different formulations of the integer linear programming (ILP) problem—for the restricted and relaxed versions of the original problem.

Although the circle packing problem is a nonlinear programming problem with quadratic constraints, the proposed approach transforms it into a sequence of integer linear programming problems, which allows the use of efficient ILP methods to obtain guaranteed lower and upper bounds on the optimal value.

Article [6] proposes the jupm algorithm, the main idea of which is to temporarily consider the radii of circles as variables, which allows finding the best configurations. The method begins with the generation of the initial location of the circles, then uses a local optimization method with the help of the IPOPT (Interior Point OPTimizer) software package. After that, a “jump” is used, in which a pair of circles switch places, it allows the algorithm to break out of this local optimum and transition to another, potentially better configuration.

Since this algorithm is based on a mathematical model with complex nonlinear dependencies, it can be classified as a nonlinear programming method. However, due to the use of “jumps” to find better solutions, bypassing exhaustive search, it can also be classified as a heuristic method.

In [7], a certain generalization of the jupm algorithm is applied. In this approach, after each cycle of local optimization, which is also done using the IPOPT package, not two, but several circles are permuted. To do this, a problem is considered where the radii of the selected circles are variable and additional special constraints are introduced to ensure that during the optimization process, the circles from the selected set “swap places,” while the set of radii remains the same. The resulting new local optimum will potentially be better.

Another approach is used in [8]. This work uses a combination of a genetic algorithm and a local minimum search algorithm. The vector of circle coordinates acts as a chromosome. After crossing, the circles exchange the coordinates of their centers, and we obtain a new arrangement of circles, which may be unacceptable, and from this new point, the local minimum search method starts.

The last three works use the local optimization algorithm as a sub-task many times in global search. Thus, the search for an effective algorithm for local optimization is relevant.

1.3. Objective and tasks

The goal of this study is to implement a local optimization algorithm in Python based on the gradient projection method and taking into account the specifics of the task at hand. After that, compare the resulting implementation with the built-in optimization algorithms of the scipy.optimize library.

This article is structured as follows: Section 2 contains the problem statement, mathematical model, and its properties. Section 3 presents the algorithmic features of the implemented method. Section 4 provides the results of numerical experiments and their comparison with built-in Python local optimization methods. Section 5 contains conclusions.

2. Mathematical model of the problem and its properties

The circle packing problem considers an index set of n circles C_i with known radii r_i, $i \in N = \{1, \ldots, n\}$ and a strip S of fixed width W and (a priori) unbounded length L. The goal is to place the n circles inside the smallest rectangle of size $W \times L$ so that no two circles overlap and no circle extends beyond the rectangle's boundary.

The mathematical model of the problem of packing circles into a strip as a mathematical programming problem looks like this [6]:

$$\min_{L, x_i, y_i} L \tag{1}$$

with restrictions

$$g_{ij}(x) = (r_i + r_j)^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \leq 0, \quad \forall i, j : 1 \leq i \leq j \leq n, \tag{2}$$

$$g_i^{\text{bot}}(x) = r_i - y_i \leq 0, \quad g_i^{\text{top}}(x) = y_i + r_i - W \leq 0, \quad \forall i \in N, \tag{3}$$

$$g_i^{\text{left}}(x) = r_i - x_i \leq 0, \quad g_i^{\text{right}}(x) = x_i + r_i - L \leq 0, \quad \forall i \in N, \tag{4}$$

where (x_i, y_i) are the centers of circle C_i, (2) is the constraint for non-intersection of circles, (3) and (4) are the limits along the Y and X axes, respectively.

Properties of the mathematical model.

1. The number of variables in the problem is $2n + 1$, that is, we have an optimization problem in the space $\mathbb{R}^{2n+1}$.
2. The objective function is linear.
3. The number of constraints is $m = 4n + \dfrac{n(n-1)}{2}$.
4. $4n$ constraints are linear, and the rest are inversely convex.
5. The optimal solution is located at an extreme point of the feasible region.
6. At the extreme point of the feasible region, $2n + 1$ inequalities are active.
7. The upper estimate of the number of local minima is C_m^{2n+1}.

Thus, we have a nonlinear nonconvex conditional optimization problem.

3. Features of implementing the gradient projection method for the circle placement problem

To solve the problem, the gradient projection method (Rosen's method) is used[13]. The pseudocode of the algorithm is given in Algorithm 1. It starts working from any acceptable point. At each step, the direction of movement is constructed as a projection of the gradient onto a hyperplane formed on the basis of the gradients of the constraints active at the current point. These gradients form matrix A, and the constraint numbers form working list L[14]. If the direction constructed in this way is zero, the constraints included in the working list and forming matrix A are checked. If some of them are no longer active, they are excluded from the working list and matrix A, and the process continues. If all constraints in the working list are active, then the Kuhn-Tucker conditions are checked at the current point. If there are constraints that correspond to negative Kuhn-Tucker multipliers, then all of them or the constraint with the smallest multiplier are excluded from the working list, and the process continues. If there are no such constraints, then we have found a local optimum.

The problem under consideration has two important features. First, all constraints of the problem at any point are concave. This means that despite the nonlinearity of some of the constraints, the specified method of constructing the direction of movement will not immediately take us out of the feasible

region. And some constraints will cease to be active during movement. That is why such constraints must be periodically removed from the working list. Second, since the objective function is linear, any step in the direction of descent will decrease the objective function. This means that at each step we will stop only when we "hit" some constraint, which means that at each step a new constraint will become active and be added to the working list.

The pseudocode for the maximum step that does not violate linear constraints is given in Algorithm 2, and the pseudocode for the maximum step that does not violate nonlinear constraints is given in Algorithm 3.

Algorithm 1 Gradient projection method for strip packing

Require: `radii`, $x^{(0)}$, `tol`, `max_iter`
Ensure: x

$x \leftarrow x^{(0)}, \quad L \leftarrow \emptyset$ ▷ L is working set
Form matrix A from the constraints of the working list L
for all $i \in L$ **such that** $g_i(x) > -\texttt{tol}$ **do**
    $L \leftarrow L \cup \{i\}$
end for
for $k = 0$ **to** `max_iter` **do**
    $A \leftarrow [\nabla g_i(x) \mid i \in L]$
    $P \leftarrow I - A^\top (AA^\top)^{-1} A$
    $y \leftarrow -P\nabla f(x)$
    if $\|y\| < \texttt{tol}$ **then**
        Remove $i \in L$ where $g_i(x) < -\texttt{tol}$
        if some i was removed **then**
            continue
        end if
        Solve $A^\top \lambda = -\nabla f(x)$
        if $\min(\lambda) \geq 0$ **then**
            return x
        else
            $L \leftarrow L \setminus \{\arg\min(\lambda)\}$
            continue
        end if
    end if
    $d \leftarrow y/\|y\|$
    $\alpha_{lin} \leftarrow \texttt{findLinearAlpha}(d, all\_cons, tol)$
    $\alpha_{nl} \leftarrow \texttt{findNonLinearAlpha}(d, x, radii, all\_cons, tol)$
    $\alpha_{\max} \leftarrow \min(\alpha_l, \alpha_n)$
    $L \leftarrow L \cup i, i \in \alpha_{\max}$
    $x \leftarrow x + \alpha_{\max} d$
end for
return x

Figure 1 shows an example of the algorithm working on thirty circles. The circles on the top are in their initial position, and the circles on the bottom are the result of optimization.

4. Results of numerical experiments

The proposed algorithm was implemented in Python using matrix operations from the NumPy library[15]. This implementation was compared with local optimization methods from the Optimize section of the SciPy package[16]. This package has several algorithms for finding the local minimum of

Algorithm 2 Computation of α_{lin}

Require: d, all_cons, tol
Ensure: α_{lin}

for linear $i \in all\_cons$ **do**
  $deriv \leftarrow grad\_row \cdot d$
  if $deriv > tol$ **then**
    $\alpha \leftarrow -g_i / deriv$
    if $0 < \alpha < \alpha_{\text{lin}}$ **then**
      $\alpha_{\text{lin}} \leftarrow \alpha$
    end if
  end if
end for
return α_{lin}

Algorithm 3 Computation of α_{nl}

Require: d, x, $radii$, all_cons, tol
Ensure: α_{nl}

for nonlinear $i \in all\_cons$ **do**
  $A \leftarrow (dx_i - dx_j)^2 + (dy_i - dy_j)^2$
  if $A < tol$ **then**
    continue
  end if
  $B \leftarrow 2\big((dx_i - dx_j)(x_i - x_j) + (dy_i - dy_j)(y_i - y_j)\big)$
  $C \leftarrow (x_i - x_j)^2 + (y_i - y_j)^2 - (radii[i] + radii[j])^2$
  $disc \leftarrow B^2 - 4AC$
  if $disc < tol$ **then**
    continue
  end if
  $\alpha_{\text{nl}} \leftarrow \min\{\, \alpha \mid \alpha \in \{\frac{-B-\sqrt{disc}}{2A}, \frac{-B+\sqrt{disc}}{2A}\},\ \alpha > tol \,\}$
end for
return α_{nl}

conditional nonlinear optimization: SLSQP, COBYLA, trust-constr. The result of the algorithms depends on the initial position of the circles, i.e., if you use the algorithm with the same set of circles but with different positions, the length of the ribbon may vary. Therefore, to compare the methods, the same set of circles with the same initial position was used. Table 1 compares the algorithms in terms of execution time (Time) in seconds and the obtained minimization results (Value) – the length of the ribbon. The first column shows different data sets, with the number of circles in brackets, that were proposed in[6] and are widely used in publications to compare optimization results[7].

It was discovered that when the number of circles grows large, the SLSQP method often produces unstable or unrealistic results because it is a local solver that becomes highly sensitive to initialization, scaling, and constraint handling. For small problems it behaves reasonably, but at larger scales the numerical gradients and quadratic models degrade. The solver may converge to infeasible or extreme points unless carefully bounded and restarted from multiple initial positions.

Trust-constr and COBYLA runs slower than SLSQP but sometimes has better resul. However, for 100 circles the run exceeded the 15-minute time limit.

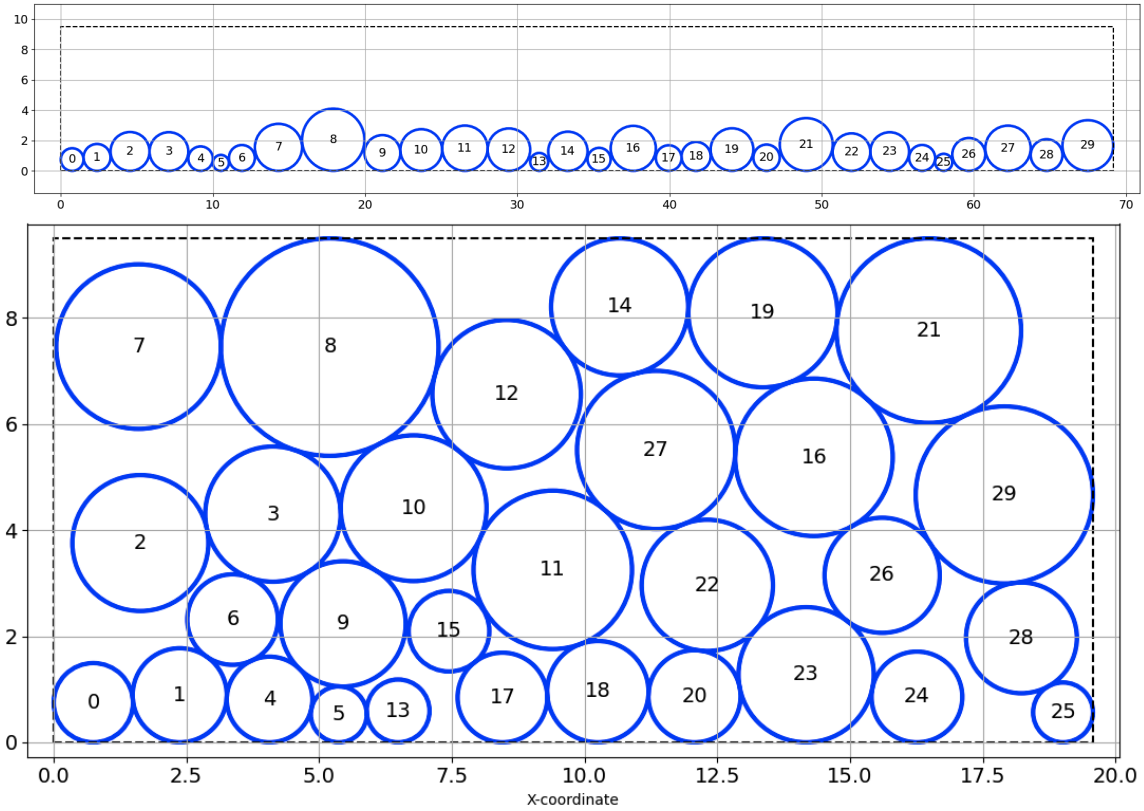

Figure 1: Circle positions before and after the algorithm runs.

Table 1
Algorithm comparison

Name	Strip Size	SLSQP		trust-constr		COBYLA		Gradient projection	
		Value	Time	Value	Time	Value	Time	Value	Time
SY1(30)	9.5	21.4258	1.6478	19.5692	33.1259	19.6811	140.0507	19.7650	1.7729
SY2(20)	8.5	16.9872	0.3370	17.0380	0.9754	15.9530	24.3653	17.1761	0.3252
SY3(25)	9	18.4978	0.7869	16.2645	18.0203	16.3038	83.8553	16.6905	0.8318
SY4(35)	11	26.6269	2.9732	26.3147	20.2244	25.8357	419.5545	26.6390	3.5827
SY5(100)	15	124.028	75.547	-	-	-	-	39.7569	796.7323
SY12(50)	9.5	36.6170	9.8393	33.4974	183.6629	-	-	36.4932	6.4808

5. Conclusion

In this article, we implemented a gradient projection method to find the local minimum in the circle packing problem. The results were compared with the local optimization methods of the scipy.optimize package in Python. It has been found that the implemented method works faster than standard methods, which means that it has potential for use as an auxiliary method in global searches.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-5 in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] S. Yakovlev, O. Kartashov, System Analysis and Classification of Spatial Configurations, in: 2018 IEEE First International Conference on System Analysis & Intelligent Computing (SAIC), IEEE, 2018, pp. 1–4. URL: https://ieeexplore.ieee.org/document/8516760/. doi:10.1109/SAIC.2018.8516760.

[2] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (2007) 1109–1130. URL: https://linkinghub.elsevier.com/retrieve/pii/S037722170600292X. doi:10.1016/j.ejor.2005.12.047.

[3] H. Akeb, M. Hifi, S. Negre, An augmented beam search-based algorithm for the circular open dimension problem, Computers & Industrial Engineering 61 (2011) 373–381. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835211000647. doi:10.1016/j.cie.2011.02.009.

[4] Z. Fu, W. Huang, Z. Lü, Iterated tabu search for the circular open dimension problem, European Journal of Operational Research 225 (2013) 236–243. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221712007680. doi:10.1016/j.ejor.2012.10.022.

[5] R. Taşpınar, B. Kocuk, Discretization-Based Solution Approaches for the Circle Packing Problem, 2023. URL: http://arxiv.org/abs/2401.00217. doi:10.48550/arXiv.2401.00217, arXiv:2401.00217 [math].

[6] Y. Stoyan, G. Yaskov, Packing unequal circles into a strip of minimal length with a jump algorithm, Optimization Letters 8 (2014) 949–970. URL: http://link.springer.com/10.1007/s11590-013-0646-1. doi:10.1007/s11590-013-0646-1.

[7] S. Yakovlev, O. Kartashov, K. Korobchynskyi, B. Skripka, Numerical Results of Variable Radii Method in the Unequal Circles Packing Problem, in: 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), IEEE, 2019, pp. 1–4. URL: https://ieeexplore.ieee.org/document/8779288/. doi:10.1109/CADSM.2019.8779288.

[8] S. Yakovlev, O. Kartashov, O. Yarovaya, On Class of Genetic Algorithms in Optimization Problems on Combinatorial Configurations, in: 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), IEEE, Lviv, 2018, pp. 374–377. URL: https://ieeexplore.ieee.org/document/8526746/. doi:10.1109/STC-CSIT.2018.8526746.

[9] R. Sharma, M. K. Rai, R. Khanna, Structure optimization: Configuring optimum performance of randomly distributed mixed carbon nanotube bundle interconnects, International Journal of Circuit Theory and Applications 51 (2023) 3949–3967. URL: https://onlinelibrary.wiley.com/doi/10.1002/cta.3605. doi:10.1002/cta.3605.

[10] R. Sharma, M. K. Rai, R. Khanna, Pragmatic structure optimization: Achieving optimal crosstalk delay and gate oxide reliability of randomly mixed CNT bundle interconnects, Micro and Nanostructures 195 (2024) 207983. URL: https://linkinghub.elsevier.com/retrieve/pii/S2773012324002322. doi:10.1016/j.micrna.2024.207983.

[11] R. Sharma, M. K. Rai, R. Khanna, Crosstalk Delay and Reliability Analysis of Carbon Nanotube On-Chip Interconnects: Modelling and Optimization using Metaheuristic Algorithm, IETE Journal of Research 71 (2025) 603–615. URL: https://www.tandfonline.com/doi/full/10.1080/03772063.2024.2423263. doi:10.1080/03772063.2024.2423263.

[12] packomania.com, 2025. URL: https://www.pacomania.com, accessed on October 10, 2025.

[13] M. Minoux, Mathematical Programming: Theory and Algorithms, John Wiley & Sons, 1986.

[14] P. E. Gill, W. Murray, M. H. Wright, Practical Optimization, SIAM, 2019.

[15] Numpy documentation, 2025. URL: https://numpy.org/doc/stable/, accessed on October 10, 2025.

[16] Scipy.optimize documentation, 2025. URL: https://docs.scipy.org/doc/scipy/tutorial/optimize.html, accessed on October 10, 2025.