Particle Swarm Algorithm for Weighted Rectangle Placement

Yi-Chun Xu
Institute of Intelligent Vision and Image Information
and
School of Electrical Engineering and Information Technology
China Three Gorges University
China
xuyichun@tom.com

Ren-Bin Xiao
School of Electrical Engineering and Information Technology
China Three Gorges University
China
rbxiao@163.com

Martyn Amos
Department of Computing and Mathematics
Manchester Metropolitan University
United Kingdom
M.Amos@mmu.ac.uk

Abstract

In this paper we present a new algorithm for a layout optimization problem: this concerns the placement of rectangular, weighted objects inside a circular container, the two objectives being to minimize imbalance of mass and to minimize the radius of the container. This problem carries real practical significance in industrial applications (such as the design of satellites), as well as being of significant theoretical interest. Previous work has dealt almost exclusively with purely circular objects, but here we deal with the much more realistic case where objects are rectangular. We present a particle swarm-based solution and compare it with the best published algorithm for this problem. Experimental results show that our approach out-performs this existing method in terms of both solution quality and execution time.

1. Introduction

The *Layout Optimization Problem* (LOP) concerns the physical placement of instruments or pieces of equipment in a spacecraft or satellite. Because these objects have mass, the system is subject to additional constraints (beyond simple Cartesian packing) that affect our solution. The two main constraints that we handle in this paper are (1) the space occupied by a given collection of objects (envelopment), and (2) the non-equilibrium (i.e. imbalance) of the system. The rest of the paper is organized as follows: In Section 2 we first present a detailed description of the problem, and describe previous related research. In Section 3 we discuss in detail how to measure and thus optimize object overlap. In Section 4 we describe the initial compaction algorithm, and in Section 5 we describe our own particle swarm local search method. We then give the results of numerical experiments in Section 6, which confirm that our method out-performs the previous best known algorithm for this problem. We conclude with a discussion of future work.

2. Definition of the problem

The LOP was proposed by Feng *et al.* [2] in 1999, and has significant implications for the cost and performance of devices such as satellites and spacecraft. It concerns the *two dimensional* physical placement of a collection of *objects* (instruments or other pieces of equipment) within a spacecraft/satellite "cabinet", or *container*. Previous work on this

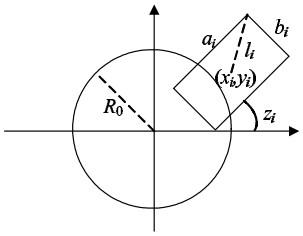

Figure 1. Illustration of notation

problem [5, 9, 11, 14] has almost always modeled objects as circles in order to simplify the packing process. However, in real-world applications, objects are generally rectangular in shape, and modeling them as circles leads to expensive wastage of space.

A search of the relevant literature yields only a single paper [12] dealing with the placement of *rectangular* objects in this context. The solution described is similar in principle to previous work, in that the authors transform the problem to one with a single objective and then solve it using a genetic algorithm (GA). However, the execution time of these algorithms is generally high, and we seek a quicker method.

In this paper we present a rapid algorithm for the LOP with rectangular objects. Our method derives from the intuitive idea that we should first compact the objects tightly, giving a "rough" initial solution. It is generally probable that a *good* solution lies near the initial solution, so we then apply a local search technique to find it. To obtain the initial compact layout, we use a technique introduced in [3, 10, 13]. We use an *energy function* to quantify the degree of overlap between objects, and then optimize this function to yield a compact layout. Finally, to perform the local search we use particle swarm optimization [1, 4, 7, 8].

In what follows, we assume that each object is rectangular, with *dimensions*, and *mass*. The container region is assumed to be circular. Suppose there are n rectangular objects to be placed, with dimensions (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) and masses $m_1, m_2, \ldots, m_n$. Assume the center of mass and shape is located at the same point in each rectangle. All objects are to be placed in a circular container, whose radius is R_0. We set the Cartesian origin to the center of the container. Let vector X =(x_1, y_1, z_1,x_2, y_2, z_2,..., x_n, y_n, z_n) denote a layout, where (x_i, y_i) is the center of the rectangle i, and z_i is the angle between the edge a_i and the positive direction of the x-axis (Figure 1).

The *static imbalance* of the system may be defined as follows:

$$B = \sqrt{(\sum_{i=0}^{n} m_i x_i)^2 + (\sum_{i=0}^{n} m_i y_i)^2} \tag{1}$$

From (1), it is easy to know $B \geq 0$, and B=0 when $\sum_{i=1}^{n} m_i x_i = 0$ and $\sum_{i=1}^{n} m_i y_i = 0$. This means that the center of mass of the system should be located at the origin (0, 0) if we require the imbalance of system be minimal. Then, for a layout X, the *envelopment radius*, denoted by envelop(X), is the longest distance from the center of mass to any of the corner points of any object. This may be formalized as:

$$\begin{aligned} R &= \min_X \text{ envelop}(X) \\ &= \min_X \max_{i=1,2,..,n} \max(\text{d}(A_i, o'), \text{d}(B_i, o'), \\ &\text{d}(C_i, o'), \text{d}(D_i, o')) \end{aligned} \tag{2}$$

where o' is the center of mass of the system, A_i, B_i, C_i, D_i are the corner points of rectangle i, and d(P_1, P_2) returns the Euclidian distance between point P_1 and point P_2.

How to deal with the constraints is a key issue in optimization. In our method, we let the *overlap* constraints be an objective of optimization. We use a function overlap(X, r) to quantify the overlap of a layout X, where r is the radius of the container. The overlap objective can thus be written as:

$$V = \min_X \text{ overlap}(X, R_0) \ = \ 0 \tag{3}$$

3. Measurement and optimization of overlap

For convenience, we use overlap(i, j) to denote the overlap between rectangles i and j, overlap(i,r) to denote the overlap between object i and the container, and overlap(i) to denote the total overlap on object i:

$$\begin{aligned} &\text{overlap}(X, r) = \sum_{i=1}^{n} \text{overlap}(i) \\ &= \sum_{i=1}^{n} (\sum_{j=1}^{n} \text{overlap}(i, j) + \text{overlap}(i, r)) \end{aligned} \tag{4}$$

3.1. Determining existence of overlap

Before *measuring* overlap, we require very some simple rules to first judge whether or not overlap *exists*. Suppose we have two rectangles $\square ABCD$ and $\square A'B'C'D'$. The sufficient and necessary conditions for no overlap include all the following:

1. $\square ABCD$ is not *inside* $\square A'B'C'D'$, or vice versa.
2. None of the edges of $\square ABCD$ intersect any of the edges of $\square A'B'C'D'$.

3.2. Overlap definitions

It is reasonable to try to *precisely* quantify overlapping area, but, in practise, this is complex and computationally expensive. More importantly, it is often also difficult to find

an object *movement* by which overlap may be reduced. Let l_i denote the distance from the center of rectangle i to a corner point of i. Denote the distance between rectangles i and j by $d_{i,j}$, and denote the distance between rectangle i and the container center by$d_{i,o}$. We now give our definitions of overlap(i, j) and overlap(i, r):

$$\text{overlap}(i,j) = \begin{cases} l_i + l_j - d_{i,j}, \text{ if overlap exists between } i \text{ and } j \\ 0, \text{ otherwise} \end{cases} \tag{5}$$

$$\text{overlap}(i,r) = \begin{cases} d_{i,o}, \text{ if overlap exists between } i \text{ and the container} \\ 0, \text{otherwise} \end{cases} \tag{6}$$

where $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, and $d_{i,o} = \sqrt{x_i^2 + y_i^2}$. It is obvious that overlap(i,j) $\geq$0 and overlap(i,r) $\geq$0.

3.3. Moving rectangles to reduce overlap

From (5) and (6), it is easy to see that, if overlap exists between i and j, the reduction of overlap(i, j) requires us to enlarge $d_{i,j}$, the distance from i to j. If overlap exists between i and the container, we should shorten $d_{i,o}$, the distance from i to the origin. By the strategy of gradient descent, we obtain the motion direction of rectangle i:

$$\begin{aligned} \frac{\partial \text{overlap}(i)}{\partial x_i} &= -\sum_{j=1}^{n} \frac{2(x_i - x_j)}{\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}} + \frac{2x_i}{\sqrt{x_i^2+y_i^2}} \\ \frac{\partial \text{overlap}(i)}{\partial y_i} &= -\sum_{j=1}^{n} \frac{2(y_i - y_j)}{\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}} + \frac{2y_i}{\sqrt{x_i^2+y_i^2}} \end{aligned} \tag{7}$$

So the iteration steps to reduce the overlap(i) are:

$$\begin{aligned} x_i &:= x_i + \alpha \frac{\partial \text{overlap}(i)}{\partial x_i}, \\ y_i &:= y_i + \alpha \frac{\partial \text{overlap}(i)}{\partial y_i}. \end{aligned} \tag{8}$$

The step length α is first set to 1. If overlap(i) does not decrease, l set α = 0.618α and repeat. This loop continues until either overlap(i) is reduced or α is arbitrarily small enough.

4. Compaction algorithm

The purpose of the compaction algorithm (CA) is to obtain an *initial layout* without overlaps, where the objects are compacted reasonably tightly. The algorithm first moves the objects to get a layout without overlap. It then reduces the radius of the container and searches once again for a feasible layout. This process is continued until the algorithm cannot find a feasible layout within a container of a certain radius. At this point, we say that a tightly compacted layout has been reached. The main steps are described as follows:
function compaction

- Predefined constant δ is the compaction step length, ε is a small positive number

1. Generate a random layout X. Set $r = R_0$ and *last_overlap* = overlap(X).
2. Set *this_overlap* = move(X, r) (reduce overlap)
3. If *this_overlap* = 0, let $r = r - \delta$ and goto 2 (compact)
4. If $|this\_overlap - last\_overlap| > \varepsilon$, goto 2 (Cannot compact any more, so make the overlap = 0)
5. While *last_overlap* $> \varepsilon$, *last_overlap* = move(X, R_0).
6. Return X

end compaction
function move (X, r)

1. for i = 1 to n, {
2. if overlap(i) >0 {
3. Get the motion direction by (7) and set α = 1
4. Get the new position of rectangle$i$ by (8);
5. If overlap of i increases and $\alpha > \varepsilon$, set α = 0.618α and go to 4
6. Update the position of i } }
7. Return overlap(X, r)

end move

From the above description, it is clear that the CA has two major shortcomings: First, only *translation* movements of objects are considered, and *rotations* are ignored. If we rotate a rectangle, it is perfectly possible that the layout may improved. Secondly, the imbalance objective is not considered. We now show how to overcome these shortcomings with a local search process.

5. Particle swarm local search

Particle swarm optimization (PSO) [1, 4, 7, 8] is suitable for problems where the objective function is not differentiable. Like evolutionary algorithms, it uses a population, or group of *particles*, to search for the optimum. A particle regards a solution to the problem as its position and the optimum as the target position. Particles use the notion of *velocity* to alter their position and move to to the target. The velocity of a particle is determined by its inertia, the best position in its history, and the best position in the whole group. The "best positions" are evaluated by a fitness function, which usually is the objective function. The main stages in PSO are:

1. Set up k particles, each with a random initial position p and initial velocity v = 0;

2. For each particle, update its velocity and position by (9) and (10):

$$v := C_1 * v + C_2 * rand() * (pbest - p) \\ +C_3 * rand() * (gbest - p) \quad (9)$$

$$p := p + v \quad (10)$$

3. Repeat Step 2 for a predefined m cycles and output the details of the best particle

In (9) and (10), p and v denote the position and the velocity, rand() is a function to return a random floating point number between [0,1], *pbest* denotes the best position of the particle and *gbest* denotes the global best position of the group. C_1, C_2 and C_3 are three constants that can influence the convergence speed of PSO if they meet a constraint [1]. In this paper we set $C_1 = 0.729$ and $C_2 = C_3 = 1.49445$, as in [8].

5.1. Applying PSO after compaction

Although PSO can often show good convergence, it is prone to becoming trapped in local minima. Although techniques such as mutation have been tried in the past in order to alleviate this problem, it does not yield satisfactory solutions when applied globally to layout problems [5]. In this paper we do not depend on PSO for the global search. Instead, much of the work is done with the compaction algorithm, and we only apply PSO as a local search tool to its output once a reasonable layout has been produced (although we do test PSO on its own for purposes of comparison).

The general local search starts from an initial feasible solution and searches the neighborhood for a better solution [6]. After we obtain the feasible layout X_0 generated by CA, we use it as a starting point to apply a local search method that we name particle swarm local search (PSLS). Unlike the standard PSO, which starts from random positions, PSLS initializes the positions of particles as follows:

$$p[1] = X_0, \\ p[i] = X_0 + rand(), i = 2, 3, \ldots, k \quad (11)$$

In (11), X_0 is set as the position of a particle. Then, during the iteration steps, the "*gbest*" particles are known to be better than X_0. The fitness function of PSLS is described as:

$$\text{fitness}\,(X) = \\ \text{envelop}(X) + \begin{cases} L, \text{if overlap}(X) > \varepsilon, \\ 0, \text{otherwise.} \end{cases} \quad (12)$$

where L is a very large constant, and ε is a very small positive number. The purpose of using L instead of overlap(X) is to *prevent* overlap – if a small overlap appears, the fitness value will increase a lot.

6. Numerical experiments

We use four examples to test our algorithm, three of which are taken from [12]. The relatively small size of the existing test cases may be due to reasons of computational feasibility – the reported computational time for a nine object example is more than three hours. We therefore design a "large-scale" example that has twenty objects as the fourth test case. The dimensions and masses of the rectangles are listed in Table 1. The first three test cases were used to measure the performance of the GA, PSO alone, and CA-PSLS, and the fourth was used to measure both particle-based algorithms. Both PSO and CA-PSLS used 30 particles, and ran for 5000 and 4500 cycles respectively.

The computational results are listed in Table 2, where the radii of envelopment, imbalance and the elapsed machine time are used to compare the performance of the algorithms. Results for radius and imbalance are given as the best result obtained over 10 runs. In Table 2, our algorithm in this paper is denoted as CA-PSLS, PSO alone as PSO and the algorithm in [12] as GA (genetic algorithm). Because our tests were run on a 1.83G/512M computer and the GA in [12] was run on a 586/166M CPU, we simply use $t' = \frac{1830}{166} \times t$ to convert run time (of course, the comparison is imprecise, but it serves for the purposes of illustration). Graphical representations of our best solutions for each case are shown in Figure 2.

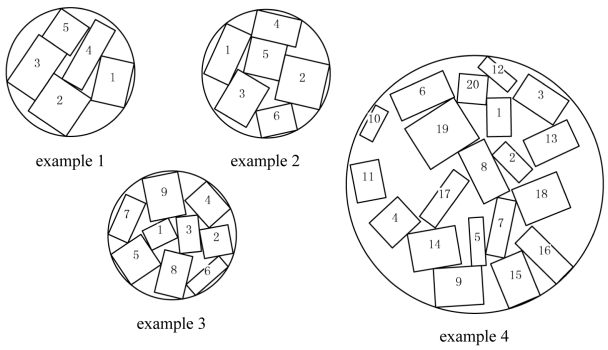

Figure 2. Results for 4 test cases

The results show that both particle-based algorithms consistently outperform the GA in terms of envelopment radius and imbalance, and offer a significant improvement in terms of machine time. For small problem instances, the compaction algorithm offers little advantage over "pure"

Table 1. Sizes and masses of four test cases

Example 1 (5 objects)	Sizes	(8,6), (8,8), (10,6), (12,4), (6,6)
	Masses	12, 16, 15, 12, 9
Example 2 (6 objects)	Sizes	(8,6), (8,8), (10,6), (10,8), (10, 10) (12,6)
	Masses	12, 16, 15, 20, 25, 18
Example 3 (9 objects)	Sizes	(8,6), (8,8), (10,6), (10,8), (10,10), (12,6), (12,4), (12,8), (12,10)
	Masses	12, 16, 15, 20, 25, 18, 12, 24, 30
Example 4 (20 objects)	Sizes	(8,5), (4,8), (10,6), (7,8), (10,3), (12,6), (12,4), (12,6), (8,10), (7,3), (8,6), (8,3,) (10,6), (10,8), (10,7), (12,5), (12,4), (10,8), (12,10), (6,6)
	Masses	10, 8, 15, 14, 7.5, 18, 12, 18, 20, 5.25, 12, 6, 15, 20, 17.5, 15, 12, 20, 30, 9

Table 2. Results of numerical experiments

	Algorithm	Envelopment	Imbalance	Time (s)
1	GA	11.801	< 10	4718
	PSO	11.333	0	77
	CA-PSLS	11.598	0	77
2	GA	15.002	< 10	5229
	PSO	14.049	0	121
	CA-PSLS	14.617	0	121
3	GA	18.851	< 10	10980
	PSO	17.734	0	342
	CA-PSLS	18.343	0	287
4	GA	—	—	—
	PSO	30.289	0	1852
	CA-PSLS	26.660	0	1786

PSO, but when the problem size is increased, the benefit of its inclusion quickly becomes apparent.

7. Conclusions

Since evolutionary algorithms generally start from a random population and their evolutionary processes are also stochastic, they often perform badly when applied to constrained layout optimization problems. In this paper we have described an alternative particle-based algorithm for the placement of weighted rectangles within a containing circle. This method significantly out-performs the only known algorithm for this problem, a method which is evolutionary in nature.

Our algorithm moves objects and then compacts them to get a reasonable feasible solution, then uses this feasible solution as the starting point for particle swarm optimization. Although the results obtained are promising, much more work is required, in terms of both testing it against larger "real world" problem instances, and further refining the underlying heuristics.

References

[1] M. Clerc and J.Kennedy. The particle system - exploration, stabilty, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):53–58, 2002.

[2] E. Feng, X. Wang, X. Wang, and H. Teng. An algorithm of global optimization for solving layout problems. *European Journal of Operational Research*, 114:430–436, 1999.

[3] W. Huang and R. Xu. Two personification strategies for solving circles packing problem. *Science in China (Ser. E)*, 29(4):347–353, 1999.

[4] J. Kennedy and J. Eberhart. Particle swarm optimization. In *Proc. IEEE Int'l. Conf. on Neural Networks*, pages 1942–1948, 1995.

[5] N. Li, F. Liu, and D. Sun. A study on the particle swarm optimization with mutation operator constrained layout optimization. *Chinese Journal of Computers*, 27(7):897–903, 2004.

[6] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Dover Publications, 1998.

[7] Y. Shi and R. Ebehart. A modified particle swarm optimizer. In *Proc. IEEE Int'l. Conf. on Evolutionary Computation*, pages 69–73, 1998.

[8] Y. Shi and R. Ebehart. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. Congress on Evolutionary Computation*, pages 84–88, 2000.

[9] F. Tang and H. Teng. A modified genetic algorithm and its application to layout optimization. *Journal of Software*, 10:1096–1102, 1999.

[10] H. Wang, W. Huang, Q. Zhang, and D. Xu. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141:440–453, 2002.

[11] Y. Yu, J. Cha, and X. Tang. Learning based GA and application in packing. *Chinese Journal of Computers*, 24(12):1242–1249, 2001.

[12] J. Zhai, E. Feng, Z. Li, and H. Yin. Non-overlapped genetic algorithm for layout problem with behavioral constraints. *Journal of Dalian University of Technology*, 39(3):352–357, 1999.

[13] D. Zhang and A. Deng. An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Computers & Operations Research*, 32:1941–1951, 2005.

[14] C. Zhou, L. Gao, and H. Gao. Particle swarm optimization based algorithm for constrained layout optimization. *Control and Decision*, 20(1):36–40, 2005.