Lauri Jokinen

Cyclic Placement Method for Capsule Packing Problem

School of Science

Bachelor's thesis
Espoo 31.8.2018

Thesis supervisor and advisor:

Prof. Harri Ehtamo

Aalto University
School of Science

AALTO-YLIOPISTO
PERUSTIETEIDEN KORKEAKOULU

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä: Lauri Jokinen

Työn nimi: Syklinen sijoittelumenetelmä kapseleiden pakkausongelmassa

Päivämäärä: 31.8.2018 Kieli: Englanti Sivumäärä: 5+27

Koulutusohjelma: Teknillinen fysiikka ja matematiikka

Vastuuopettaja: Prof. Harri Ehtamo

Tässä kandidaatintyössä esitetään algoritmi kahden tasossa olevan kapselin leikkauksen pinta-alan laskentaan sekä menetelmä kapseleiden pakkausogelmaan neliössä. Tässä ongelmassa tavoitteena on pakata kapseleita mahdollisimman paljon neliön sisään siten, etteivät kapselit ole päällekkäin. Algoritmi minimoi annetusta alkutilanteesta kapseleiden välisiä, päällekkäisiä pinta-aloja, kunnes näiden pinta-alojen summa on mahdollisimman pieni. Esitetyssä menetelmässä liikutetaan yhtä kapselia kerrallaan pitäen muut kapselit paikoillaan.

Menetelmän tehokkuus perustuu siihen, että liikuttamalla yhtä kapselia suuri osa päällekkäisistä pinta-aloista ei muutu. Optimoitaessa yhden kapselin paikkaa pitää laskea ainoastaan siihen kapseliin vaikuttavat pinta-alat. Kaikkien kapseleiden paikat täytyy optimoida kylläkin useita kertoja, jotta kapselit asettuvat optimaalisille paikoille. Vaikka parannamme vain yhden kapselin paikkaa kerrallaan, tämä vähentää laskennan määrää merkittävästi vaihtoehtoiseen menetelmään verrattuna.

Työssä esitettyä menetelmää verrataan vaihtoehtoiseen menetelmään, jossa kaikkien kapseleiden paikkoja ja asentoja optimoidaan samanaikaisesti. Syklinen menetelmä toimii pienissä ongelmissa (5 kapselia) noin 65 % nopeammin kuin vaihtoehtoinen menetelmä ja tämä suhteellinen laskentateho paranee kapseleiden määrän kasvaessa.

Avainsanat: Kapselin pakkausongelma, optimointi, gradienttimenetelmä, syklinen menetelmä

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
BACHELOR'S THESIS

Author: Lauri Jokinen

Title: Cyclic Placement Method for Capsule Packing Problem

Date: 31.8.2018 Language: English Number of pages: 5+27

Degree programme: Engineering Physics and Mathematics

Supervisor and instructor: Prof. Harri Ehtamo

In this thesis we present an algorithm for calculating an intersection area of two capsules on the plane and a cyclic placement algorithm for a packing problem where capsules are packed into a square box. The cyclic placement method finds a local optimum for a given initial setup by minimizing the overlapping areas of capsules. In the method a single capsule is moved at a time and all the other capsules are kept fixed in their places.

Efficiency of the cyclic placement method is based on the fact that when moving a single capsule, most of the overlapping areas between the other capsules remain unchanged regardless of the position of the chosen capsule. Thus, we need to calculate fewer overlapping areas in order to move a capsule to a somewhat better location. On the other hand, we need to optimize the locations of all the capsules several times to finally reach the overall optimum for all the capsules.

In this thesis the cyclic placement method is compared to an alternative method where places and orientations of all the capsules are optimized simultaneously. In small packing problems (5 capsules) cyclic placement method is approximately 65% faster than the alternative method and the computational efficiency improves with more capsules.

Keywords: Capsule packing problem, Optimization, Gradient method, Cyclic placement method

Preface

This thesis was mostly put together during the summer 2018, but after that it took several months to mould it to its final form. I'd like to thank my supervisor and advisor, Professor Harri Ehtamo, for his patience with me and my sloppy terminology. I'd like to, and will, thank the wonderful people of the Polytech Choir, my family and most importantly Loviisa, for their support.

Servinkuja, 27.1.2019 Lauri Jokinen

Contents

1 Introduction

In a packing problem considered here, the goal is to fit as many objects as possible into a certain space, such as a square box or a cube. Such packing problem is usually studied in two or three dimensions, but other dimensions are also possible. Packing problems have various applications in several engineering areas such as logistics, medicine and materials science. In this thesis we'll look at a two dimensional packing problem where we pack a square room, e.g. a lift car, with capsules of the same size.

We approach the packing problem with minimization of the total overlapping areas of all objects. Capsules can be used in many applications, for example to model humans from above. Ellipses are also commonly used for this purpose [1], but capsules have simpler shapes for packing problems, because overlapping areas are easier to calculate for two capsules than for two ellipses. Capsules are first put randomly into the room allowing them to overlap each other freely, and also intersect the square walls, and then this overlapping is minimized.

Large optimization problems are usually decomposed to a number of smaller subproblems that can be computed faster than the full problem. Cyclic algorithms (or coordinate descent algorithms) are one of the first and simplest methods to optimize problems sequentially [2][3]. Time has shown somewhat conflicting results with cyclic methods, and sometimes alternative and more sophisticated methods easily surpass the efficiency of cyclic methods [2]. Yet, cyclic algorithms are used in various practical problems with good results, for example in neural networks [4] to simplify optimization problems.

A cyclic method in a packing problem allows us to reduce the objective function to a form that is faster to evaluate. When moving a single object in the box, most of the overlapping areas (or volumes if the problem is three dimensional) between the other capsules remain the same, except for the overlapping areas with the capsule under consideration. Thus, when replacing the object we can handle only these overlapping areas at a time. The new position is probably not the final optimal location for the object – we'll have to optimize all of the objects' locations several times in cycles, until no capsule cannot be moved to a better position, i.e., a local optimum is found.

In this thesis we present a way of calculating overlapping areas between capsules and compare the cyclic placement method against a benchmark method, in which all the capsules' position parameters are optimized simultaneously with gradient method, as is also done in [1]. The algorithm is implemented with *Matlab* software and its features and functions, mainly *Optimization Toolbox* and *Polygons* are used for the simulations. Vector and matrix notation is used as much as possible to make the problem easier to handle and to extend results to three dimensions.

2 Background

In the optimization literature various packing problems have been studied a lot, especially in the field of integer optimization. Ruokokoski studied a lift car packing problem [1]. In particular, he modelled travellers as ellipses. The results can be used, for example, when studying people flow capacity for large buildings. The problem can be extended by adding people with suitcases or shopping carts. In Ruokokoski's paper there was a problem with ellipses occasionally overlapping each other very clearly. In this paper we implement repelling factors between the objects (see Chapter 3.3) which solves this problem. Ruokokoski approached the packing problem with optimizing all the capsules' locations simultaneously, that resulted in long calculations – especially for larger simulations.

A cyclic method is often coupled with a pattern search, such as an acceleration step which helps coping with discontinuous objective functions [3], Chapter 8.5, or the method of Hooke and Jeeves which adds some gradient descent spice to the method [3].

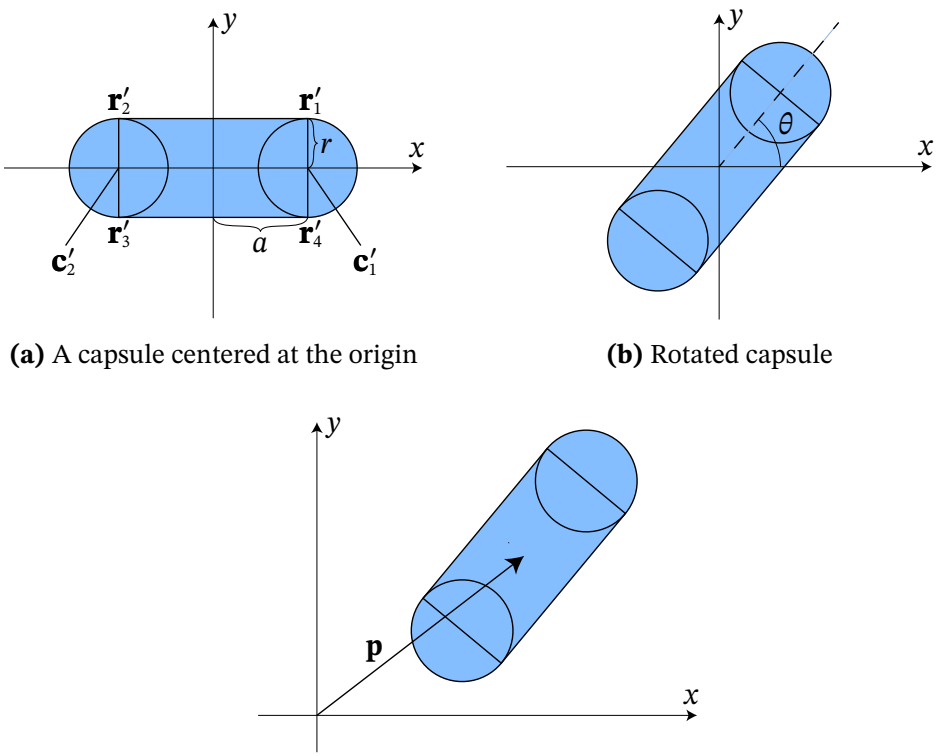
In a paper by A. Honkela, H. Valpola and J. Karhunen [4] coordinate descent was applied to a problem in neural networks. The paper's approach was very similar to that of ours and the results showed a 60%-85% reduction in the convergence times compared to an alternative, Bayesian learning method. They coupled the cyclic method very successfully with a pattern search, the method of Hooke and Jeeves. Not many recent papers exist on optimization with cyclic method – and so far there is no paper, where a cyclic method is applied to a packing problem.

3 Materials and methods

3.1 Mathematical definition of a capsule

A capsule centered at the origin can be defined with two parameters (also illustrated in Figure 1): half of the width of the rectangle $a \in \mathbb{R}_+$, and circles' radii $r \in \mathbb{R}_+$. The location of such capsule in the plane can be defined by three parameters: the angle of the capsule with respect to the x-axis $\theta \in \mathbb{R}$, and a translation $\mathbf{p} \in \mathbb{R}^2$. We shall write

$$\mathbf{s} = \begin{bmatrix} \mathbf{p} \\ \theta \end{bmatrix}. \tag{1}$$



(a) A capsule centered at the origin

(b) Rotated capsule

(c) Rotated and then displaced capsule

Figure 1: Parameters and transformations of a capsule

We will place the capsule at the origin, and then we'll rotate and displace the capsule. The rectangle's coordinates are as follows (see Figure 1):

$$\mathbf{r}'_1 = \begin{bmatrix} a \\ r \end{bmatrix}, \quad \mathbf{r}'_2 = \begin{bmatrix} -a \\ r \end{bmatrix}, \quad \mathbf{r}'_3 = \begin{bmatrix} -a \\ -r \end{bmatrix}, \quad \mathbf{r}'_4 = \begin{bmatrix} a \\ -r \end{bmatrix}.$$

The radius of the circles is r and their center points are

$$\mathbf{c}'_1 = \begin{bmatrix} a \\ 0 \end{bmatrix}, \quad \mathbf{c}'_2 = \begin{bmatrix} -a \\ 0 \end{bmatrix}.$$

Let a rotation matrix be

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

We'll rotate the capsule with an angle θ and then displace the capsule with $\mathbf{p}$, see Figures 1b and 1c. The final coordinates for the rectangle's corners are,

$$\mathbf{r}_i = R(\theta)\mathbf{r}'_i + \mathbf{p}, \quad 1 \leq i \leq 4, \tag{2}$$

and the center points of the circles are

$$\mathbf{c}_i = R(\theta)\mathbf{c}'_i + \mathbf{p}, \quad 1 \leq i \leq 2. \tag{3}$$

For further calculations we need an indicator whether a given point is in the capsule or not. The point $\mathbf{q}$ is in the capsule $\mathbf{s}$, if it's in the rectangle, or in the circles. First we map the point $\mathbf{q}$ to the origin state of the capsule $\mathbf{s}$ by first displacing it by $-\mathbf{p}$ and then rotating it with $R(-\theta)$. Let this point be $\mathbf{q}' = R(-\theta)(\mathbf{q} - \mathbf{p})$. Now, the point $\mathbf{q}'$ is in the rectangle of the capsule iff $[(-a \leq q'_1 \leq a) \wedge (-r \leq q'_2 \leq r)]$. The point is in the right circle iff $[(q'_1 - a)^2 + (q'_2)^2 \leq r]$, and the point is in the left circle iff $[(q'_1 + a)^2 + (q'_2)^2 \leq r]$. All combined, point $\mathbf{q}$ is in capsule $\mathbf{s}$ iff

$$\begin{aligned} &\big[\,(-a \leq q'_1 \leq a) \wedge (-r \leq q'_2 \leq r)\,\big] \\ \vee&\big[\,(q'_1 + a)^2 + (q'_2)^2 \leq r\big] \\ \vee&\big[\,(q'_1 - a)^2 + (q'_2)^2 \leq r\big], \end{aligned} \tag{4}$$

where $\mathbf{q}' = R(-\theta)(\mathbf{q} - \mathbf{p})$.

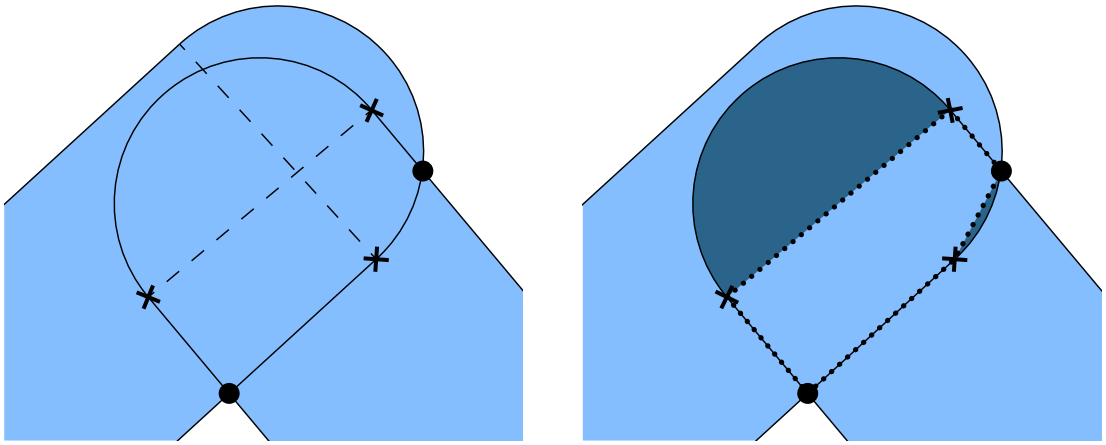
3.2 Calculating overlapping area of two capsules

Let's find the overlapping area for two capsules of the same size, i.e., with the same a and r, $\mathbf{s}_1 = \begin{bmatrix}\mathbf{p}_1^T & \theta_1\end{bmatrix}^T$ and $\mathbf{s}_2 = \begin{bmatrix}\mathbf{p}_2^T & \theta_2\end{bmatrix}^T$. We'll calculate the intersection area in two steps: first we'll find a convex polygon that covers a part of the intersection area so that only feasible circle segments (i.e., segments included in both capsules) are left out. Second we find and calculate the areas for the circle segments. Capsules are convex sets and an intersection of two convex sets is convex [6]. In particular, an intersection area of two rectangles is a convex polygon.

First we need to calculate the *intersection points of two capsules' contours*, let's call these points ipc's. Equations for the ipc's are provided in Appendix A. An example of the ipc's is shown in Figure 2a, where ipc's are illustrated as black spots. The two capsules can totally coincide, or they may share joint contours. If this is the case, we reject the ipc's at these regions. In Figure 3 there are further examples that can occur. Math for these features are included in Appendix A. Note, that the number of ipc's can be 1,2,3 or 4.

We also need to define some auxiliary points, denoted in Figure 4 as crosses. Let's call these points *fixed capsule points*, or fcp's. The fcp's consist of rectangle corner points, which are needed to calculate a correct intersection polygon. Feasible fcp's are in both capsules, see Equation 4. The possible fcp's on $\mathbf{s}_1$ are,

$$R(\theta_1)\begin{bmatrix}\pm a \\ \pm r\end{bmatrix} + \mathbf{p}_1, \text{ with every combination of } \pm \text{ -signs.}$$



(a) Black spots are ipc's and crosses are fcp's.

(b) Convex polygon (dotted line) and segments (dark areas). In this case both of the segments are feasible.

Figure 2: Calculating the intersection area of two capsules

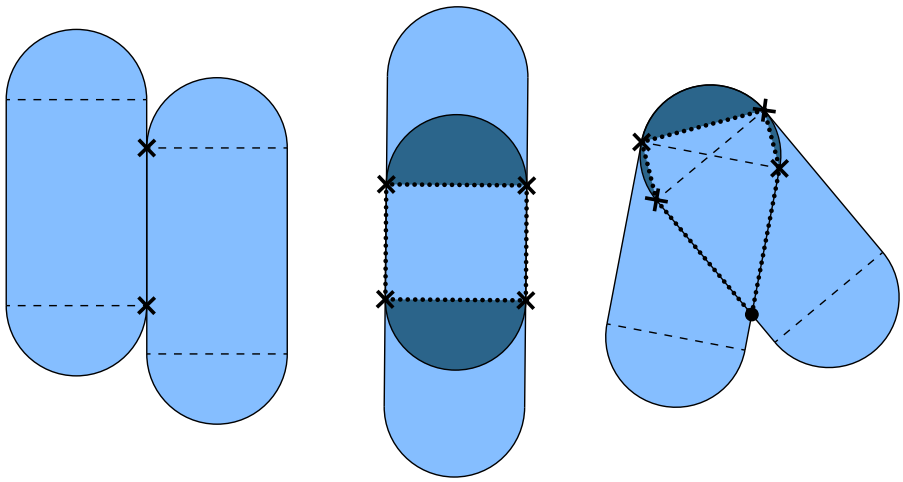


Figure 3: Different kinds of shared contours between the capsules with the corresponding convex polygons and segments.

And with capsule $\mathbf{s}_2$ possible fcp's are

$$R(\theta_2)\begin{bmatrix}\pm a\\ \pm r\end{bmatrix} + \mathbf{p}_2, \text{ with every combination of } \pm \text{-signs.}$$

An example of feasible fcp's is illustrated in the Figure 2a as crosses.

Next we find the smallest convex polygon containing all ipc's and fcp's. I used *Matlab*'s own algorithm `convhull` [6] to find the convex hull. An example of this hull is shown in Figure 2b. The area inside the convex polygon can be calculated with *Matlab*'s command `polyarea` [7].

Now all that is left are the areas of the feasible segments. Let's choose two adjacent

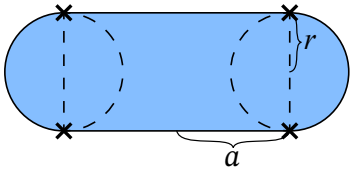
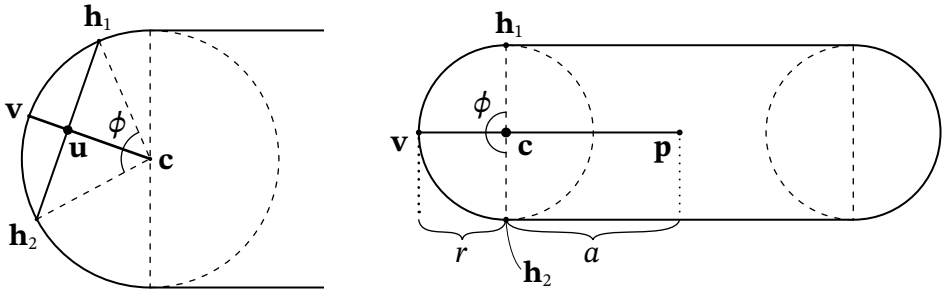


Figure 4: A capsule with fixed capsule points (fcp's) denoted with crosses.

points, $\mathbf{h}_1$ and $\mathbf{h}_2$, defined by black spots and/or crosses. Let's find, if a feasible segment exists between the points and if the segment's area should be included to the total intersection area. For the two points, two conditions must be true. First, both points must lie on a same circle on either of the capsules. Let $\mathbf{c}$ denote the center of the capsule circle. Second, the segment region must be included in both of the capsules, see Equation (4). To find whether this is true, we only need to know if a single point on the segment arc is inside or outside the capsules.



(a) Case, where $|\mathbf{u}-\mathbf{c}| \neq 0$ **(b)** Case, where $|\mathbf{u}-\mathbf{c}| = 0$

Figure 5: Calculating the top point $\mathbf{v}$ for the segment arc.

Let's solve for the segment arc's top point $\mathbf{v}$, see Figure 5a. If we start from the circle's center point $\mathbf{c}$, we need a vector that has the length of the circle's radius r, and direction from the center point $\mathbf{c}$ to the middle point of the points $\mathbf{h}_1$ and $\mathbf{h}_2$. Let the middle point be $\mathbf{u} = (\mathbf{h}_1 + \mathbf{h}_2)/2$. The direction vector is thus $-\mathbf{c} + \mathbf{u}$. So the top point in the middle of the segment's arc is

$$\mathbf{v} = \mathbf{c} + r\frac{\mathbf{u}-\mathbf{c}}{|\mathbf{u}-\mathbf{c}|}, \quad |\mathbf{u}-\mathbf{c}| \neq 0. \tag{5}$$

Note, that the denominator in (5) cannot be zero. If the denominator is zero, then

$$|\mathbf{u}-\mathbf{c}| = 0 \quad \Longleftrightarrow \quad \frac{1}{2}(\mathbf{h}_1 + \mathbf{h}_2) = \mathbf{c},$$

which means that $\mathbf{h}_1$ and $\mathbf{h}_2$ are on the opposite sides of the point $\mathbf{c}$. This can be true only when $\mathbf{h}_1$ and $\mathbf{h}_2$ are fcp's, because we only have half circles. Since this is the only case where the problem occurs, we can solve it with an if-statement. If we begin from the point $\mathbf{p}$, we want a vector of length $a + r$ and a direction of $-\mathbf{p} + \mathbf{c}$, see Figure 5b. The final coordinates

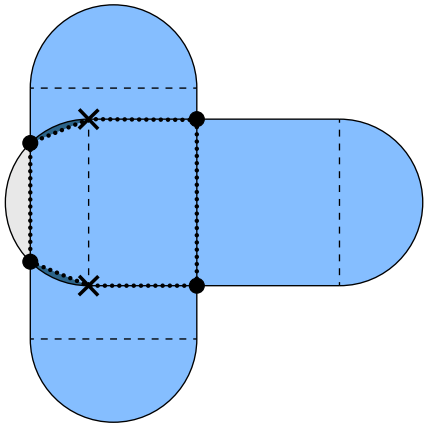


Figure 6: An example of a non-feasible segment marked with light gray.

for $\mathbf{v}$ are

$$\mathbf{v} = \begin{cases} \mathbf{p} + (a+r)\dfrac{\mathbf{c}-\mathbf{p}}{|\mathbf{c}-\mathbf{p}|}, & \text{when } |\mathbf{u}-\mathbf{c}| = 0 \\ \mathbf{c} + r\dfrac{\mathbf{u}-\mathbf{c}}{|\mathbf{u}-\mathbf{c}|}, & \text{when } |\mathbf{u}-\mathbf{c}| \neq 0. \end{cases}$$

If $\mathbf{v}$ is in both of the capsules, we'll accept this segment, otherwise it will be excluded from the calculations. In Figure 2b all of the center points of all the segments are in both of the capsules and thus all the segments are feasible; see also Figure 6, where there's a non-feasible segment.

Finally we can calculate the area of the segment [8]:

$$A_{\text{seg}} = \frac{1}{2}r^2\left(\phi - \sin\phi\right), \qquad \phi = 2\arcsin\frac{|\mathbf{h}_1 - \mathbf{h}_2|}{2r},$$

where ϕ is a central angle of the circle, see Figure 5. When we do this for all feasible segments, the resulting areas, and the area of the convex polygon, sum to the total intersecting area of the two capsules.

3.3 Objective functions

Let a set of capsules be $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, and $N = \{1, \ldots, n\}$. Let the box B be a square with sides of length $b > 0$. Corner points of B are

$$\frac{1}{2}\begin{bmatrix}\pm b \\ \pm b\end{bmatrix}, \text{with every combination of } \pm \text{-signs.}$$

For the cyclic placement method we need an objective function f_m, $m \in N$. It will be used to optimize location and orientation of a single capsule, $\mathbf{s}_m$. The function has three parts. The first part gives the overlapping areas of all the other capsules with the chosen capsule $\mathbf{s}_m$:

$$\sum_{i \in N \setminus \{m\}} A(\mathbf{s}_i, \mathbf{s}_m), \tag{6}$$

where $m \in N$, and $A(\mathbf{s}_i, \mathbf{s}_m)$ gives the intersection area of capsules $\mathbf{s}_i$ and $\mathbf{s}_m$. The second part gives the area of the capsule $\mathbf{s}_m$ that is outside of B,

$$A(\mathbf{s}_m) - A(B, \mathbf{s}_m), \tag{7}$$

where $A(\mathbf{s}_m)$ is the area of the capsule $\mathbf{s}_m$, and $A(B, \mathbf{s}_m)$ is the intersecting area of square B and capsule $\mathbf{s}_m$. The third part of the function f_m makes capsules repel each other. The

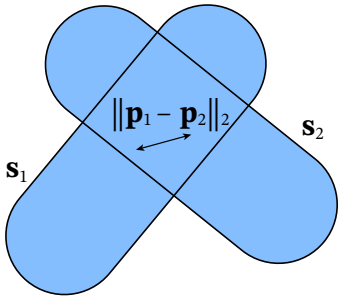


Figure 7: A case of two intersecting capsules. The norm of $\mathbf{p}_1 - \mathbf{p}_2$ is the distance between the center points $\mathbf{p}_1$ and $\mathbf{p}_2$ of the two capsules. This distance should increase, if we minimize the penalty term.

repelling factor to be minimized is interpreted as a penalty term. It is defined as follows,

$$\sum_{i \in N \setminus \{m\}} \frac{\gamma}{d(\mathbf{s}_i, \mathbf{s}_m) + 1}, \tag{8}$$

where the function $d(\mathbf{s}_i, \mathbf{s}_m)$ gives the distance between the center points of the capsules $\mathbf{s}_i$ and $\mathbf{s}_m$, see the Figure 7:

$$d(\mathbf{s}_i, \mathbf{s}_m) = \|\mathbf{p}_i - \mathbf{p}_m\|_2.$$

We multiply Equation (8) by a penalty parameter γ, which we take small because the overlapping and overflowing areas are more important to minimize. In the simulations, and for the rest of this thesis, a value of $\gamma = 10^{-6}$ is used.

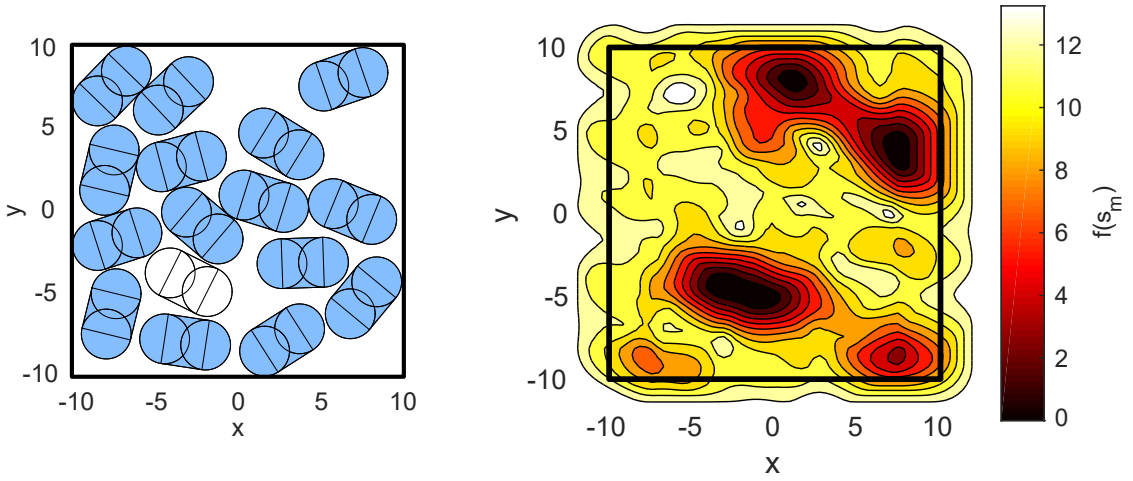
Finally we define the objective function f_m as the sum of functions defined in (6), (7) and (8):

$$f_m(\mathbf{s}_m) = \sum_{i \in N \setminus \{m\}} \left(A(\mathbf{s}_i, \mathbf{s}_m) + \frac{\gamma}{d(\mathbf{s}_i, \mathbf{s}_m) + 1} \right) + A(\mathbf{s}_m) - A(B, \mathbf{s}_m). \tag{9}$$

An example of a contour map of $f_m(\mathbf{s}_m)$ as a function of $\mathbf{p}_m$, with θ_m fixed, is shown in Figure 8b for a randomly chosen capsule $\mathbf{s}_m$ shown in Figure 8a. The function is smooth and no discontinuities arise, thus this function will work as a good objective function in optimization.

To shorten the notation, we'll define two functions:

$$\begin{aligned} G_1(\mathbf{s}_i, \mathbf{s}_m) &= A(\mathbf{s}_i, \mathbf{s}_m) + \frac{\gamma}{d(\mathbf{s}_i, \mathbf{s}_m) + 1}, \\ G_2(\mathbf{s}_m) &= A(\mathbf{s}_m) - A(B, \mathbf{s}_m), \end{aligned} \tag{10}$$



(a) A setup with 16 capsules, where, in this case, no capsules overlap. The contours on the right are calculated with respect to the white capsule, $\mathbf{s}_m$.

(b) Contour map of the objective function $f_m(\mathbf{s}_m)$ as a function of $\mathbf{p}_m$ of capsule $\mathbf{s}_m$. The capsule's angle θ_m is kept fixed as in the Figure on the left.

Figure 8: A plot of the objective function $f_m(\mathbf{s}_m)$ in the box B which is drawn as a black square.

so that,

$$f_m(\mathbf{s}_m) = \sum_{i \in N \setminus \{m\}} G_1(\mathbf{s}_i, \mathbf{s}_m) + G_2(\mathbf{s}_m). \tag{11}$$

The function f_m gives the sum of the overlapping areas between a chosen capsule and all the other capsules. We also need a function that gives areas for combinations of every two capsules. This function will be used as an objective function for the benchmark method, with which the cyclic placement method is compared. This function can be used to minimize the total overlapping area of the capsules. Let's start by summing f_m with respect to every capsule:

$$\begin{aligned}\sum_{j=1}^{n} f_j(\mathbf{s}_j) &= \sum_{j=1}^{n} \left[\sum_{i \in N \setminus \{j\}} G_1(\mathbf{s}_i, \mathbf{s}_j) + G_2(\mathbf{s}_j) \right] \\ &= \sum_{i,j \in N,\, i \neq j} G_1(\mathbf{s}_i, \mathbf{s}_j) + \sum_{j=1}^{n} G_2(\mathbf{s}_j).\end{aligned}$$

The first sum of this expression has duplicate terms, because $G_1(\mathbf{a}, \mathbf{b}) = G_1(\mathbf{b}, \mathbf{a})$. Hence we can replace $i \neq j$ with $i < j$. Let's call the resulting function $F(S)$, where $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$:

$$F(S) = \sum_{i,j \in N,\, i<j} G_1(\mathbf{s}_i, \mathbf{s}_j) + \sum_{k=1}^{n} G_2(\mathbf{s}_k) \tag{12}$$

Here f_m and F have many convenient properties; e.g., a change of $\mathbf{s}_m$ affects the value of $f_m(\mathbf{s}_m)$ the same amount as it affects the value of $F(S)$, more formally written in the following theorem.

Theorem 3.1. *Let's assume that we have a set of capsule locations $S_0 = \{\boldsymbol{s}_1, \dots, \boldsymbol{s}_{n-1}\}$. If we add an arbitrary capsule $\boldsymbol{s}_n$ to the set S_0, we get a set S. If we add an arbitrary capsule $\boldsymbol{s}'_n$ to the set S_0, we get a set S'. For these two sets it holds that $F(S) - F(S') = f_n(\boldsymbol{s}_n) - f_n(\boldsymbol{s}'_n)$.*

Proof. Using Equation (12) we can form $F(S_0)$ as

$$F(S_0) = \sum_{i,j \in N_0,\, i<j} G_1(\mathbf{s}_i, \mathbf{s}_j) + \sum_{k=1}^{n-1} G_2(\mathbf{s}_k),$$

where $N_0 = \{1, \dots, n-1\}$. The first sum's indices of $F(S_0)$ are listed in Table 1 beneath the

Table 1: Indices of the first sum in Equation (12). Indices under the double line is used with S_0 and the whole Table is used for sets S and S'.

j	$i < j, i \in \mathbb{Z}_+$
n	$\{1, 2, \dots, n-3, n-2, n-1\}$
$n-1$	$\{1, 2, \dots, n-3, n-2\}$
$n-2$	$\{1, 2, \dots, n-3\}$
$\dots$	$\dots$
3	$\{1, 2\}$
2	$\{1\}$

double line. When we add a new capsule $\mathbf{s}_n$ to the system, we'll make use of the expression for $F(S_0)$ and add missing terms whose indices are listed above the double line in Table 1. Together these indices give the whole sum. For the second sum, we only need to add one term, $G_2(\mathbf{s}_n)$, to complete the sum. Thus,

$$\begin{aligned} F(S) &= F(S_0) + \sum_{i=1}^{n-1} G_1(\mathbf{s}_i, \mathbf{s}_n) + G_2(\mathbf{s}_n) \\ &= F(S_0) + \sum_{i \in N \setminus \{n\}} G_1(\mathbf{s}_i, \mathbf{s}_n) + G_2(\mathbf{s}_n) \\ &= F(S_0) + f_n(\mathbf{s}_n), \end{aligned} \tag{13}$$

where $N = \{1, \dots, n\}$. With Equation (13) we can write

$$\begin{aligned} F(S) - F(S') &= F(S_0) + f_n(\mathbf{s}_n) - F(S_0) - f_n(\mathbf{s}'_n) \\ &= f_n(\mathbf{s}_n) - f_n(\mathbf{s}'_n), \end{aligned}$$

proving the claim. □

In order to calculate a gradient with respect to a single capsule, we need three partial derivatives, i.e., with respect to $\mathbf{p}$ and θ. Let's take θ as an example:

$$\frac{\partial F(S)}{\partial \theta_m} = \lim_{h \to 0} \frac{F(S') - F(S)}{h},$$

where S' is the same as S, except for $\theta'_m = \theta_m + h$. Using Theorem 3.1 we have,

$$\frac{\partial F(S)}{\partial \theta_m} = \lim_{h \to 0} \frac{F(S') - F(S)}{h} = \lim_{h \to 0} \frac{f(\mathbf{s}'_m) - f(\mathbf{s}_m)}{h} = \frac{\partial f_m(\mathbf{s}_m)}{\partial \theta_m}.$$

We can construct the partial derivatives for $\mathbf{p}$ the same way. Now we can write

$$\nabla F(S) = \begin{bmatrix} \nabla f_1(\mathbf{s}_1)^T & \nabla f_2(\mathbf{s}_2)^T & \cdots & \nabla f_n(\mathbf{s}_n)^T \end{bmatrix}^T, \tag{14}$$

where,

$$\nabla f_m(\mathbf{s}_m) = \begin{bmatrix} \frac{\partial f_m(\mathbf{s}_m)}{\partial p_{m,x}} & \frac{\partial f_m(\mathbf{s}_m)}{\partial p_{m,y}} & \frac{\partial f_m(\mathbf{s}_m)}{\partial \theta_m} \end{bmatrix}^T.$$

Thus, we can use whichever function in our gradient method to obtain the same results. Furthermore, if we're optimizing the position of a single capsule $\mathbf{s}_m$ with gradient method, we can use which ever objective function, $f_m(\mathbf{s}_m)$ or $F(S)$, to obtain the same results.

3.4 Optimization methods

3.4.1 Theoretical performance

Let's consider calculating the gradient in Equation (14). Let's assume that functions G_1 and G_2 (see Equations (10)) are evaluated in constant times, $T_{G1} > 0$ and $T_{G2} > 0$. We can calculate the time to evaluate f_m in Equation (11):

$$T_{f_m} := \sum_{i \in N \setminus \{m\}} T_{G1} + T_{G2} = (n-1)T_{G1} + T_{G2}. \tag{15}$$

We approximate the gradient numerically. Gradient of f_m has three components, so we need to sample f_m in four different locations,

$$\begin{bmatrix} p_{m,x} \\ p_{m,y} \\ \theta_m \end{bmatrix}, \begin{bmatrix} p_{m,x} + h \\ p_{m,y} \\ \theta_m \end{bmatrix}, \begin{bmatrix} p_{m,x} \\ p_{m,y} + h \\ \theta_m \end{bmatrix}, \begin{bmatrix} p_{m,x} \\ p_{m,y} \\ \theta_m + h \end{bmatrix}.$$

Thus, the evaluation of $\nabla f_m(\mathbf{s}_m)$ takes the computation time

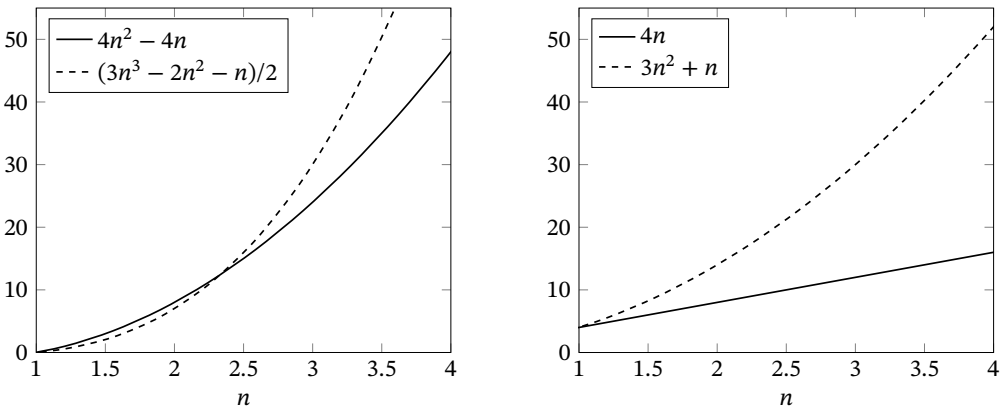
$$T_{\nabla f_m} := 4T_{f_m} = 4\big((n-1)T_{G1} + T_{G2}\big) = (4n-4)T_{G1} + 4T_{G2},$$

and the computing time for all the n capsules takes the time

$$nT_{\nabla f_m} = (4n^2 - 4n)T_{G1} + 4nT_{G2}.$$

With Equation (12) we can calculate the computation time for $F(S)$:

$$T_F := \sum_{i,j \in N,\, i<j} T_{G1} + \sum_{k=1}^{n} T_{G2} = \frac{n^2 - n}{2} T_{G1} + nT_{G2}.$$



(a) The coefficient for T_{G1}. **(b)** The coefficient for T_{G2}.

Figure 9: Coefficients for T_{G1} and T_{G2}. Dashed line is with $T_{\nabla F}$ and solid line for $nT_{\nabla f_m}$.

To calculate the gradient using Equation (12) for F, as was done in [1], rather than using the gradient given in (14), we need to calculate the initial state plus $3n$ calculations to cover all the variables of all the capsules. That results in a total time of,

$$T_{\nabla F} := (1 + 3n)T_F = \frac{3n^3 - 2n^2 - n}{2} T_{G1} + (3n^2 + n)T_{G2}.$$

As seen in Figure 9, as n is large, the computing time $nT_{\nabla f_m}$ is clearly smaller than $T_{\nabla F}$, which makes f_m better for optimization. Let's construct a measure about how large $nT_{\nabla f_m}$ is compared to $T_{\nabla F}$. Let's just divide them:

$$\frac{nT_{\nabla f_m}}{T_{\nabla F}} = \frac{(4n^2 - 4n)T_{G1} + 4nT_{G2}}{(3n^3 - 2n^2 - n)T_{G1}/2 + (3n^2 + n)T_{G2}}.$$

Let's define $g > 0$ as $T_{G1} = gT_{G2}$. Then we have,

$$\begin{aligned} t(n, g) := \frac{nT_{\nabla f_m}}{T_{\nabla F}} &= \frac{(4n^2 - 4n)\cancel{T_{G2}}g + 4n\cancel{T_{G2}}}{(3n^3 - 2n^2 - n)\cancel{T_{G2}}g/2 + (3n^2 + n)\cancel{T_{G2}}} \\ &= \frac{(4n^2 - 4n)g + 4n}{(3n^3 - 2n^2 - n)g/2 + 3n^2 + n} \\ &= \frac{8g(n - 1) + 8}{(3n + 1)(gn - g + 1)}. \end{aligned}$$

A key observation is that

$$\lim_{n\to\infty} t(n, g) = 0,$$

which means that $n\nabla f_m$ is faster to evaluate than ∇F, when n is large, regardless of the value of g.

Let's find minimum and maximum for $t(n, g)$, with respect to g. The equation $\partial t(n, g)/\partial g = 0$ does not have solutions that meet the conditions $g, n > 0$. However, we

can study the endpoints $g \to 0^+$, and $g \to \infty$:

$$\lim_{g \to 0^+} t(n, g) = \frac{4}{3n + 1}, \tag{16}$$

$$\lim_{g \to \infty} t(n, g) = \frac{8}{3n + 1}. \tag{17}$$

The function $t(n, g)$ reaches its minimum at $n \geq 1$ when $g \to 0$, and maximum when

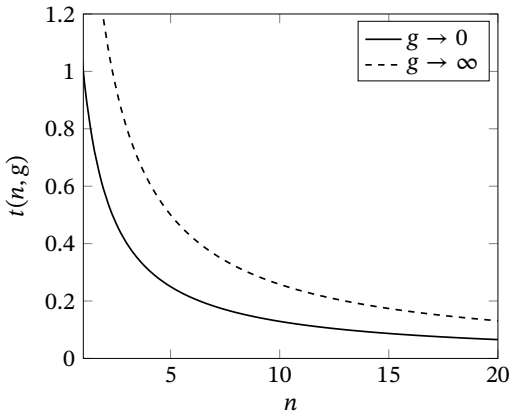


Figure 10: Minimum and maximum for $t(n, g)$ with respect to g.

$g \to \infty$, shown in Figure 10. This means that the true value for $t(n, g)$ lies between these boundaries. A value for g can be measured to make the approximations more explicit. Note that the theoretical performance does not apply exactly for comparing the cyclic placement method and benchmark method, since in cyclic placement method we move the capsules into new places along the evaluation of the gradient. Nevertheless, this is still a significant result and can give us a hint of the difference in the algorithms' performance.

3.4.2 Cyclic placement method

Here, cyclic placement method means that we're optimizing only one capsule $\mathbf{s}_m$ at a time and fix other capsules $\mathbf{s}_{i \in N \setminus \{m\}}$ to their current positions. The cyclic method applied here differs from the commonly known cyclic coordinate method [3] slightly, as we minimize not only one parameter of the objective function, but three parameters ($\mathbf{p}$ and θ) at once using gradient method.

One full cycle of iterations to the set of all the capsules $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ is in pseudo code as follows.

0. Given information: set of capsules S with initial positions in the square box, number of capsules n and the box side length b.

1. Set $m = 1$.

2. $\mathbf{s}_m \leftarrow \mathbf{s}'_m$, where $f_m(\mathbf{s}'_m) < f_m(\mathbf{s}_m)$. Here a local minimum for f_m is found with gradient method.

3. $m \leftarrow m + 1$. If $m \leq n$, go to step 2. Otherwise the cycle is complete.

The convergence of this method is ensured, if the inequality in Step 2 holds, i.e., the gradient method converges.

A cyclic method is sometimes coupled with a pattern search, such as acceleration step, or the method of Hooke and Jeeves [3], which makes the method rather effective. However, neither of these methods benefit the cyclic placement algorithm considered here. Optimal step sizes can be calculated for different pattern searches and results are very close to the original step size. On top of that, the method of Hooke and Jeeves also uses line search, which in this case requires relatively heavy computing. Therefore we prefer the pure cyclic method with no pattern searches, which seems to be the most effective method for our purposes.

3.4.3 Refining the gradient methods in Matlab

Gradient methods are used with both the cyclic method and the benchmark method. We'll be using algorithms from *Matlab*'s *Optimization Toolbox*, which has many options for the algorithms. We use the constrained minimization function (`fmincon`) coupled with an `active-set` algorithm. *Matlab*'s unconstrained optimization method, quasi-Newton method, was also tested, but in our case, it isn't faster in any way. Another down side with the quasi-Newton method is that we can't easily constrain the capsules inside the box – the method would need a modification to the objective function to prevent capsules of wandering far beyond the box's boundary.

In *Matlab*, it is possible to set iteration limits for the algorithms. A large limit could be set, because *Matlab*'s optimization algorithms use pattern searches that uses previous iterations for its advantage. On the other hand we want the algorithms to return the result as fast as possible which would leave us with a low iteration limit. Also, with the cyclic method we don't want to waste time finding a precise optimal place for a single capsule in the middle of the iterations, because it will probably change in the next iteration anyway. Optimal limits were calculated for both of the algorithms by empiric testing. For the cyclic placement method, the minimization in the Step 2 in Chapter 3.4.2, the iteration limit is set to 3 and for the benchmark method the iteration limit happens to be the same, 3.

3.5 Comparison of the algorithms

We are interested of the speed of the algorithms of finding local minima. We also let the algorithms stop to differing local optimum points. To decide if a setup is at it's local minimum, we need to see if the gradient of the objective function is close to zero. Let $\varepsilon > 0$. The algorithms are run, and the consumed time is measured, as long as inequality

$$\sum_{m=1}^{n} \left(\nabla f_m^T(\mathbf{s}_m) \cdot \nabla f_m(\mathbf{s}_m) \right) > \varepsilon \tag{18}$$

holds. Due to Theorem 3.1, we can use f_m, $1 \leq m \leq n$, in Equation (18) for both of the algorithms. We use a value of

$$\varepsilon = 3 \cdot 10^{-4} n^3 / b^2,$$

because this value resulted in consistent results with different box sizes and different number of capsules in the system. Evaluation of the statement in Equation (18) requires many calculations of overlapping areas and so comparison of the algorithms may be unfair if the result of another algorithm is checked more frequently than the other. Therefore we stop the timer during the gradient's evaluation.

To compare different setup sizes we'll simulate the algorithms with various box sizes $b = 20, 40, 60$ and different number of capsules $n = 5, 10, 20, 40$. Capsule size will be constant with $a = 2.5$ and $r = 3$. Initial setups will be generated with a pseudo-random generator with

$$\mathbf{s}_i = \begin{bmatrix} (X_1 - 1/2)\, b \\ (X_2 - 1/2)\, b \\ 2\pi X_3 \end{bmatrix}$$

for all $i \in N$ and where $X_{1,2,3}$ are random numbers uniformly distributed between 0 and 1. The generated setups should fulfill the inequality in the Equation (18), and we will generate new setups, until we find a setup where this statement holds. The random configuration is then optimized by both of the algorithms. Because high-level programming languages, like *Matlab*, are not always consistent with computing times, we'll choose randomly the order of the algorithms. All simulations will be run on a HP Z240 desktop computer with Intel® Xeon(R) CPU E3-1230 v5 @ 3.40GHz × 8 processor, 31.3GB of memory and Ubuntu 16.04 LTS. Version of *Matlab* used is 2018a and computation times are measured with *Matlab*'s command `cputime` [9].

4 Results

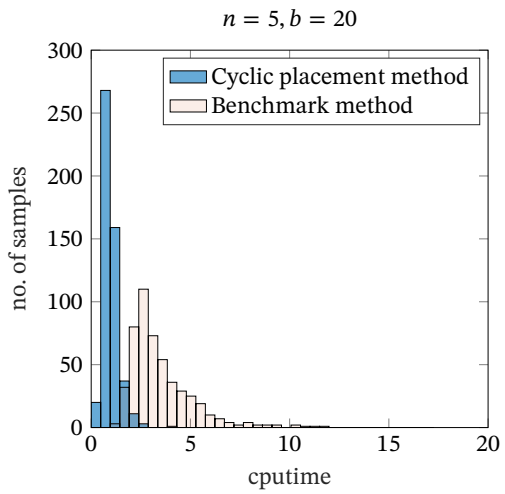
When we run both algorithms with a random initial setup, we get a data point (t_f, t_F), where t_f is the `cputime` for the cyclic placement method and t_F for the benchmark method. To get simple numerical results, we'll fit a slope $t_f = \alpha t_F$ to the data, with least squares method. If $\alpha < 1$, the cyclic placement method is, on average, faster than the benchmark method and vice versa.

Examples of three simulation scenarios are seen in Figures 13-15. In the latter two figures, one can see how the algorithms cope with impossible packing scenarios: the capsules are spread in the box overlapping each other evenly from at least three directions.

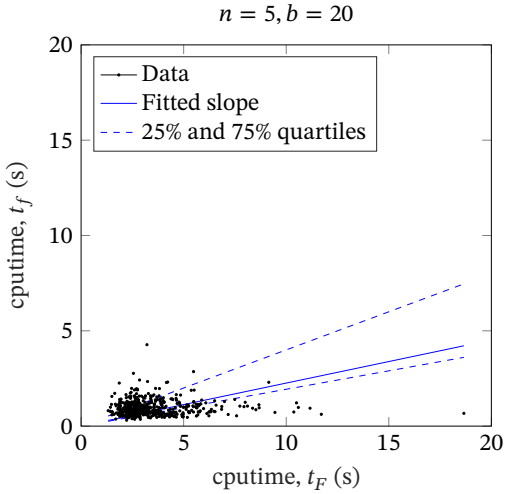
The simulations were run at least 50 times for each setting. A case with $b = 20$ and $n = 40$ was left out because of excessive computing times. Also, the box is quite overfilled with capsules. The results are in favor of the cyclic placement method, as seen in Tables 2, 3 and Figure 12. According to the results, the cyclic placement method is faster when the number of capsules is larger and/or the box is smaller. A case with five capsules and a box with a side length of 20 is studied more closely in Figure 11. As seen in Table 2, the confidence intervals are relatively small compared to the mean values which indicates good sensitivity of the results.

A theoretical minimum for the gradient evaluation times, see Equation (16), is also shown in Figure 12, and the results somewhat follow it. The theoretical model is based only on the evaluation speed of the gradients, and has an impossible assumption, $g \to 0$, but yet it is quite close to the obtained results.

The resulting objective function values from the simulations can be seen in Appendix B, where we can see that the algorithms produce somewhat similar results. Up to four regions are visible (e.g., case of $n = 20$ and $b = 60$), which is due to finding, or not finding, a solution for the packing problem. For two algorithms, that makes a total of four regions. Some problems that we simulated are impossible (e.g., $n = 20$ and $b = 20$), resulting in one region, while other scenarios are rather easy for the algorithms to solve (e.g. $n = 5$ and $b = 60$). Because of the repelling factors, see Chapter 3.3, the objective function can never be exactly zero.



(a) Histogram of computation times for both of the algorithms.



(b) Computation times for both algorithms. Fitted line $t_f = \alpha t_F$ and 25% and 75% quartiles

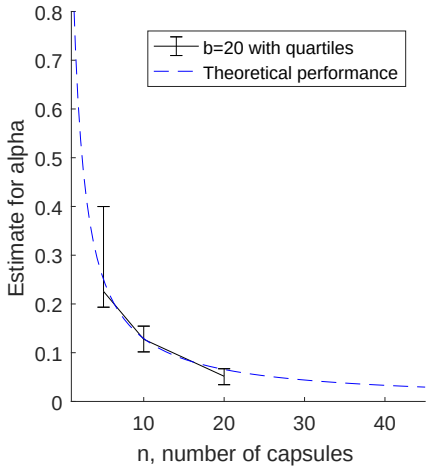
Figure 11: Computing times for both of the algorithms when $n = 5$ and $b = 20$. Sample size is 500 for each of the algorithms.

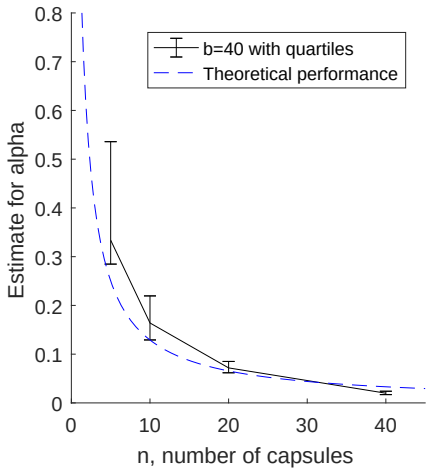
Table 2: Estimations for α in different setups.

Estimate of α with 95% confidence interval		b, size of box 20	40	60
n, number of capsules	5	0.226±0.021	0.334±0.027	0.327±0.032
	10	0.128±0.011	0.164±0.019	0.197±0.021
	20	0.0516±0.0092	0.0718±0.0056	0.0811±0.0094
	40	–	0.0199±0.0012	0.0314±0.0026

Table 3: Quartiles for α.

25% and 75% quartiles for α		b, size of box 20	40	60
n, number of capsules	5	0.193, 0.4	0.285, 0.54	0.302, 0.57
	10	0.102, 0.15	0.129, 0.22	0.144, 0.29
	20	0.0344, 0.067	0.0618, 0.085	0.0645, 0.11
	40	–	0.017, 0.024	0.0267, 0.036





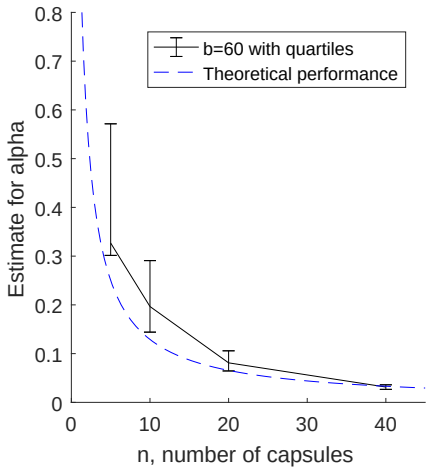
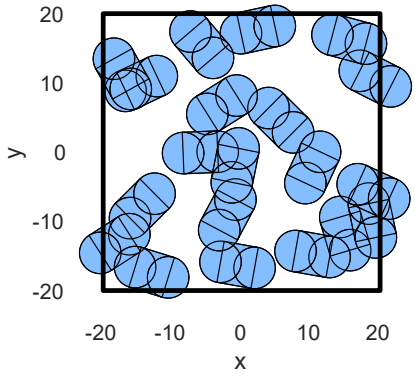
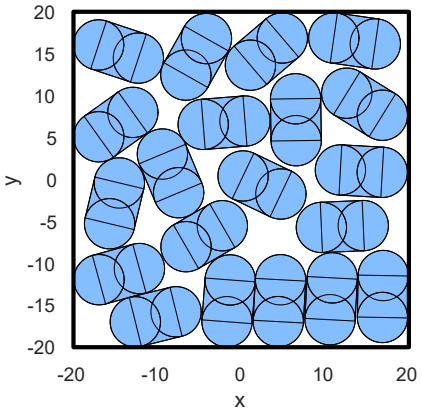


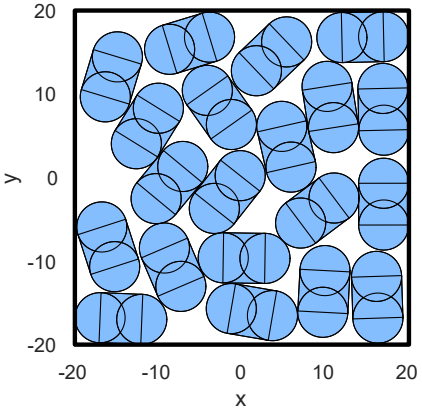
Figure 12: Data plotted from the Tables 2 and 3. Theoretical performance is from Equation (16).



(a) Initial, random setting. This is now optimized by both of the algorithms.

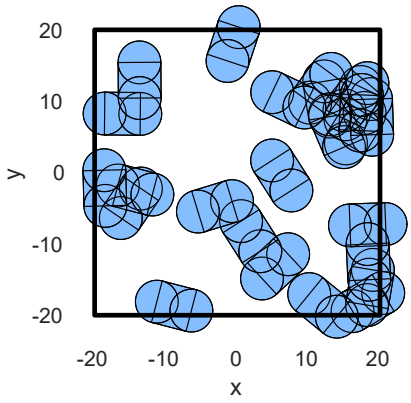


(b) Optimized setting with the benchmark algorithm.

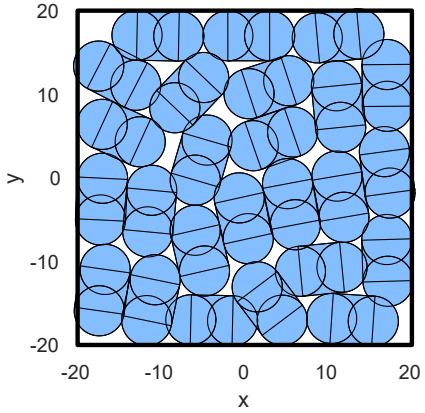


(c) Optimized setting with the cyclic placement algorithm.

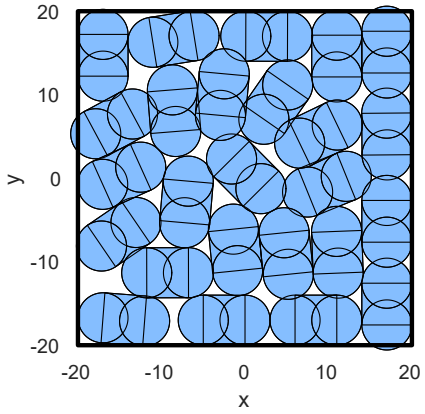
Figure 13: A sample where the number of capsules (n) is 20 and size of the box (b) is 40. In these cases local optima are found where no capsules overlap.



(a) Initial, random setting.

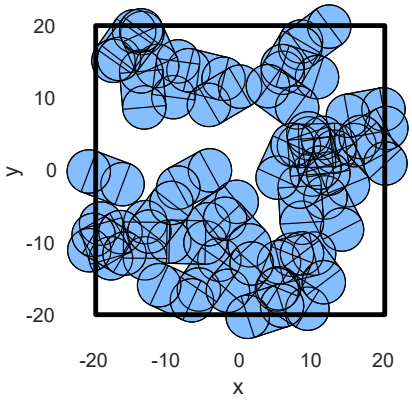


(b) Optimized setting with the benchmark algorithm.

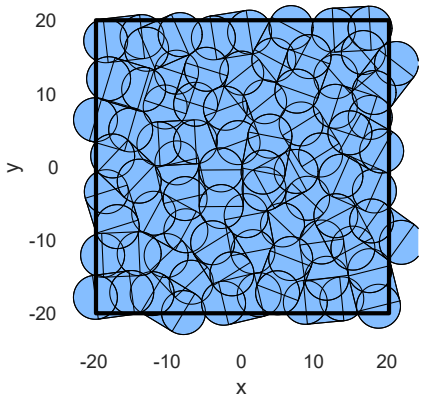


(c) Optimized setting with the cyclic placement algorithm.

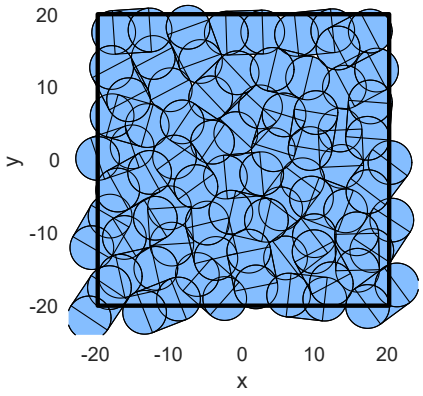
Figure 14: $n = 25$ and $b = 40$. In this case the capsules overlap somewhat in the local optima.



(a) Initial, random setting.



(b) Optimized setting with the benchmark algorithm.



(c) Optimized setting with the cyclic placement algorithm.

Figure 15: $n = 40$ and $b = 40$. In this case the capsules overlap extensively in the local optima.

5 Summary

In this thesis we presented an algorithm for packing capsules with cyclic placement method and it was compared to a benchmark method algorithm that optimizes the places of all the capsules simultaneously. Both algorithms work as expected and a valid local minimum is always found by both algorithms. Results show consistency, and that the cyclic placement method is clearly faster than the benchmark method. Nevertheless, it should be noted that if the gradient for the benchmark method had been calculated as in Equation (14), the method could have taken less time compared to the cyclic placement method.

Overlapping area is a general way of telling how much two shapes overlap and this way the cyclic placement method can be implemented easily, for example, to an ellipse packing problem. In fact, the idea of the cyclic placement method is quite general and the method can be very effective with other optimization purposes as well, where the objective function is possible to separate into factors that can be optimized separately.

The ineffectiveness of pattern searches (see Chapter 3.4.2) in the cyclic placement method is a little surprising – in many cases pattern searches are very effective and step sizes can be even around 100 times the regular step size [4], but in this problem they were entirely ineffective. Many other heuristics exist for packing problems and they are not handled in this thesis. It would be interesting to compare the cyclic placement method against other sophisticated algorithms.

References

[1] Mirko Ruokokoski, *Determining the number of passengers that can be fitted in a standard-sized lift car*, KONE Corporation, 2015.

[2] Stephen J. Wright, Mathematical Programming, *Coordinate descent algorithms*, June 2015, Volume 151, Issue 1, pp 3–34. Accessed at https://link.springer.com/article/10.1007/s10107-015-0892-3#citeas.

[3] Mokhtar S. Bazaraa, Hanif D. Sherali, C. M. Shetty, *Nonlinear Programming, Theory and Algorithms*, 1993.

[4] Antti Honkela, Harri Valpola, Juha Karhunen, *Accelerating Cyclic Update Algorithms for Parameter Estimation by Pattern Searches*, 2003 Kluwer Academic Publishers. Accessed at https://link.springer.com/content/pdf/10.1023/A:1023655202546.pdf at 30.7.2018.

[5] Wikibooks, *Convexity/The intersection of convex sets is convex*, https://en.wikibooks.org/wiki/Convexity/The_intersection_of_convex_sets_is_convex (Accessed on 8.8.2018).

[6] MathWorks Documentation, *convhull, Convex hull*, https://se.mathworks.com/help/matlab/ref/convhull.html (Accessed on 12.7.2018).

[7] Wolfram Mathworld, *Polygon Area*, http://mathworld.wolfram.com/PolygonArea.html (Accessed on 12.7.2018).

[8] Wolfram Mathworld, *Circular Segment*, http://mathworld.wolfram.com/CircularSegment.html (Accessed on 12.7.2018).

[9] MathWorks Documentation, *cputime, Elapsed CPU time*, https://se.mathworks.com/help/matlab/ref/cputime.html (Accessed on 12.7.2018).

A Intersection points with circles and line segments

A.1 Intersection point of two line segments

Let's define two line segments $\mathbf{p}$ and $\mathbf{q}$ as

$$\begin{cases} \mathbf{p} = \mathbf{p}_1 t + \mathbf{p}_2(1-t), & 0 \le t \le 1 \\ \mathbf{q} = \mathbf{q}_1 s + \mathbf{q}_2(1-s), & 0 \le s \le 1. \end{cases} \tag{A1}$$

For an intersection point of these line segments it holds:

$$\begin{aligned} & \mathbf{p} = \mathbf{q}, \\ \iff \quad & \mathbf{p}_1 t + \mathbf{p}_2(1-t) = \mathbf{q}_1 s + \mathbf{q}_2(1-s) \\ \iff \quad & t(\mathbf{p}_1 - \mathbf{p}_2) + \mathbf{p}_2 = s(\mathbf{q}_1 - \mathbf{q}_2) + \mathbf{q}_2 \\ \iff \quad & t(\mathbf{p}_1 - \mathbf{p}_2) + s(\mathbf{q}_2 - \mathbf{q}_1) = \mathbf{q}_2 - \mathbf{p}_2, \end{aligned}$$

or in matrix form,

$$\begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_2 & \mathbf{q}_2 - \mathbf{q}_1 \end{bmatrix} \begin{bmatrix} t \\ s \end{bmatrix} = \mathbf{q}_2 - \mathbf{p}_2.$$

The inverse of a 2×2 matrix is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, \tag{A2}$$

so that t and s are

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_2 & \mathbf{q}_2 - \mathbf{q}_1 \end{bmatrix}^{-1} (\mathbf{q}_2 - \mathbf{p}_2).$$

If requirements $0 \le t \le 1$ and $0 \le s \le 1$ aren't fulfilled or the inverse matrix in Equation (A2) does not exist (i.e., $ad - bc = 0$), the intersection point does not exist. If the values of t and s are feasible, then they can be substituted to Equation (A1), which gives the coordinates of intersection point.

A.2 Intersection points of two circles

Let's find the intersection points for two circles with a center points $\mathbf{c}_1$ and $\mathbf{c}_2$ and radii r_1 and r_2. Set $d = |\mathbf{c}_1 - \mathbf{c}_2|$. If $d = 0$, we return no intersection points, even if the circles are identical, otherwise we'll transform $\mathbf{c}_2$. First we'll subtract $\mathbf{c}_1$ from $\mathbf{c}_2$, and then we'll rotate the resulting vector so that it is on the x-axis. The rotation is accomplished with $R(-\alpha)$, where

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \quad \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} = \frac{-\mathbf{c}_1 + \mathbf{c}_2}{d}.$$

Now, we have a simplified system of equations

$$\begin{cases} x^2 + y^2 = r_1^2 \\ (x-d)^2 + y^2 = r_2^2 \end{cases},$$

for which the solutions for the intersection points, say $\mathbf{p}_{\pm}$, are (solved with *Mathematica*):

$$\mathbf{p}_{\pm} = \frac{1}{2d}\begin{bmatrix} d^2 + r_1^2 - r_2^2 \\ \pm\sqrt{2d^2\left(r_1^2 + r_2^2\right) - d^4 - \left(r_1^2 - r_2^2\right)^2} \end{bmatrix}.$$

Now, the intersection points of the original circles are $\mathbf{q}_{\pm} = R(\alpha)\mathbf{p}_{\pm} + \mathbf{c}_1$. Complex results are rejected.

A.3 Intersection points of a circle and a line segment

Let's find the intersection points for a line segment between points $\mathbf{r}_1, \mathbf{r}_2$, and a circle with center point $\mathbf{c}$ and radius r. Let's first subtract $\mathbf{c}$ from the points $\mathbf{r}_1$ and $\mathbf{r}_2$, and then rotate them with $R(-\beta)$, so that the line segment becomes parallel with the x-axis. These transformations are accomplished with,

$$\begin{cases} \mathbf{r}_1' = R(-\beta)(\mathbf{r}_1 - \mathbf{c}) \\ \mathbf{r}_2' = R(-\beta)(\mathbf{r}_2 - \mathbf{c}) \end{cases}, \text{where} \quad \begin{bmatrix} \cos\beta \\ \sin\beta \end{bmatrix} = \frac{-\mathbf{r}_1 + \mathbf{r}_2}{|-\mathbf{r}_1 + \mathbf{r}_2|}.$$

Denote $y_0 = \mathbf{r}_{1,y}'$. We then have a circle $x^2 + y^2 = r^2$, and a line of $y = y_0$, $x \in \mathbb{R}$, for which the points of intersection are,

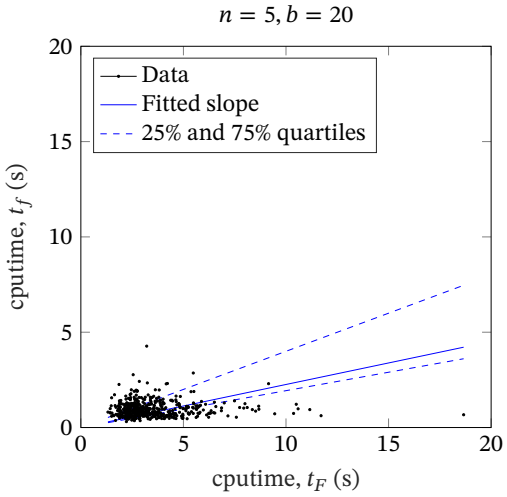
$$\mathbf{p}_{\pm} = \begin{bmatrix} \pm\sqrt{r^2 - y_0^2} \\ y_0 \end{bmatrix}.$$

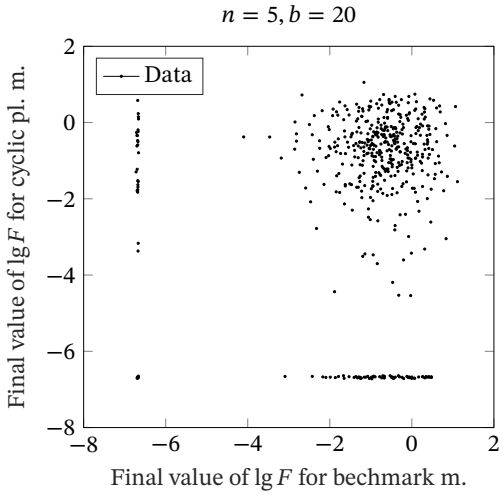
If

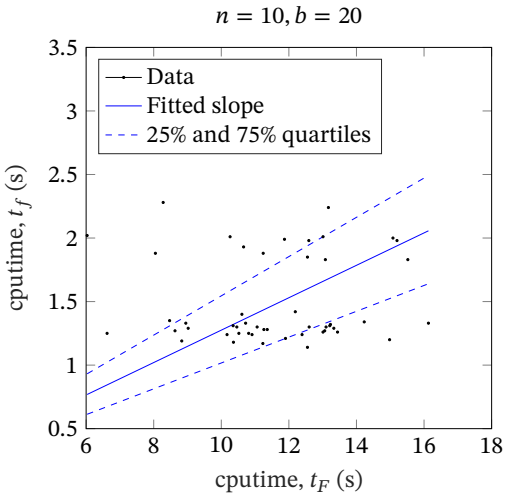
$$\min\{\mathbf{r}_{1,x}', \mathbf{r}_{2,x}'\} \leq \mathbf{p}_{\pm,x} \leq \max\{\mathbf{r}_{1,x}', \mathbf{r}_{2,x}'\},$$

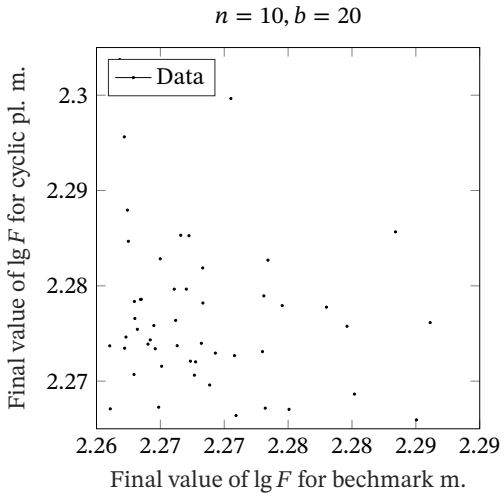
we accept the intersection point. After transformations back, the final intersection points are $\mathbf{q}_{\pm} = R(\beta)\mathbf{p}_{\pm} + \mathbf{c}$.

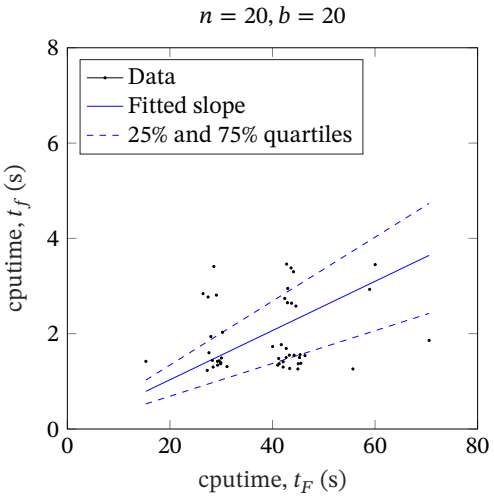
B Simulation data and fitting

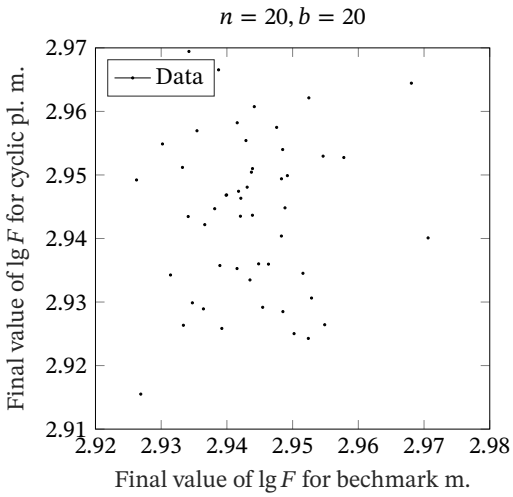


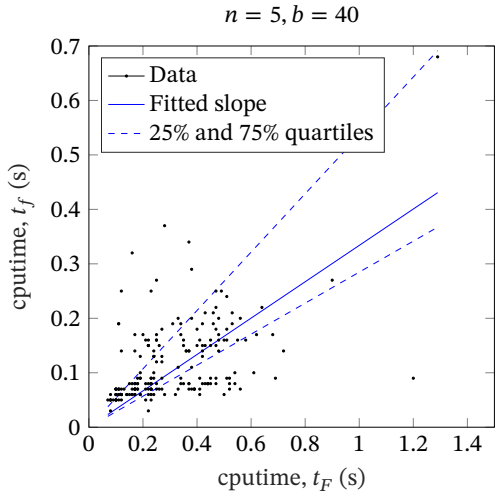


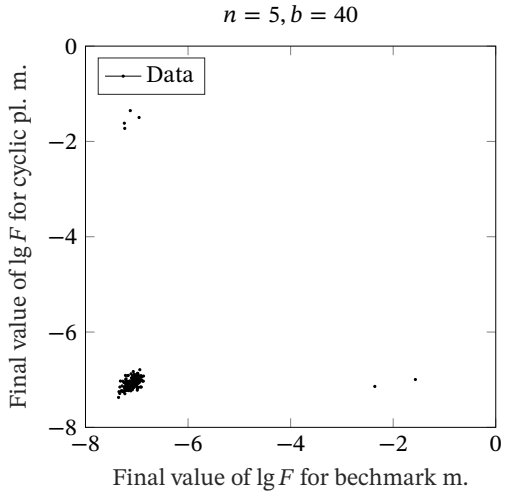


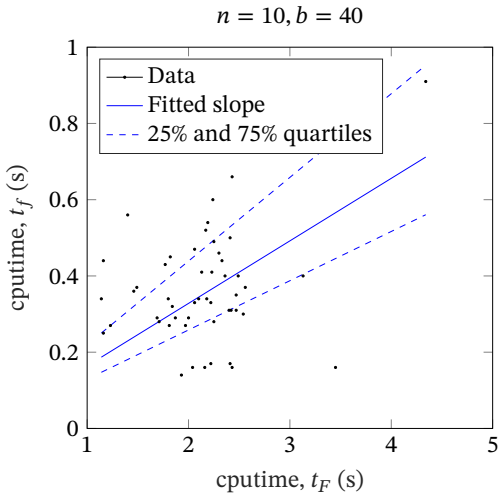


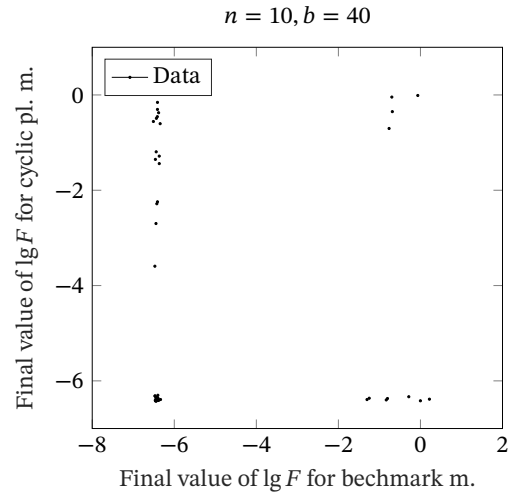

$n = 20, b = 20$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

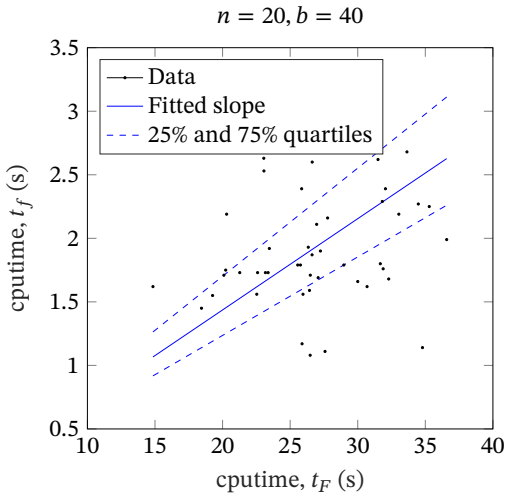

$n = 20, b = 20$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

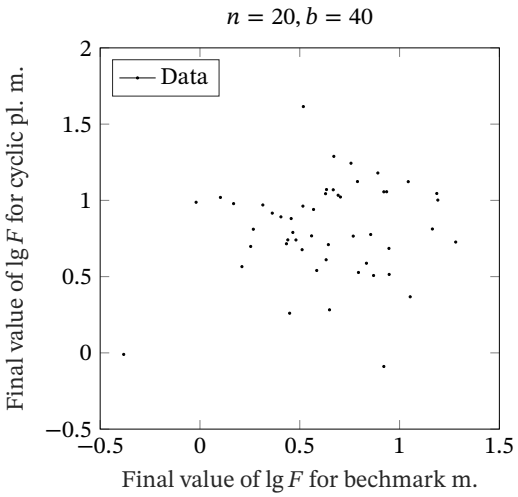

$n = 5, b = 40$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

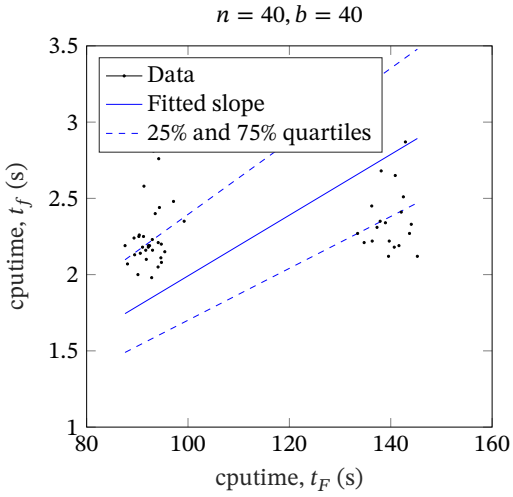

$n = 5, b = 40$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

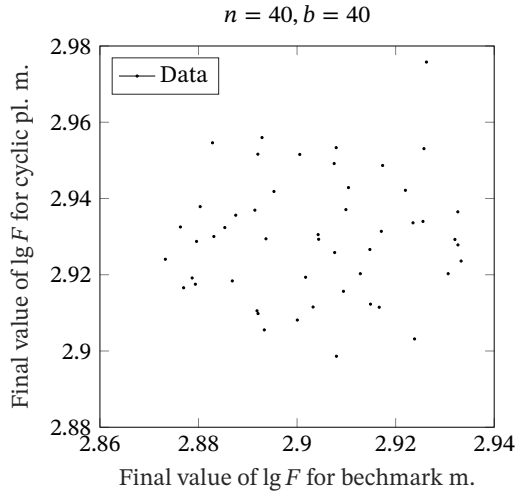

$n = 10, b = 40$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

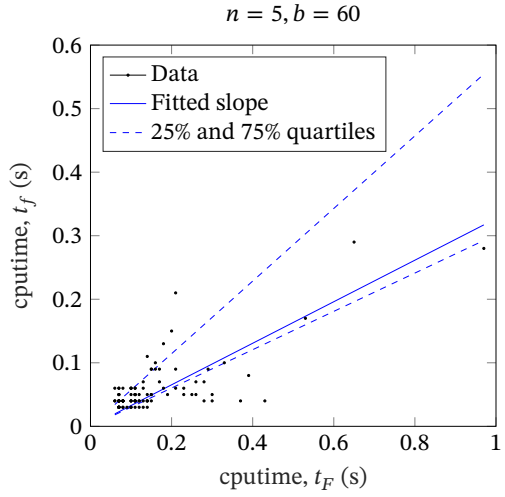

$n = 10, b = 40$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

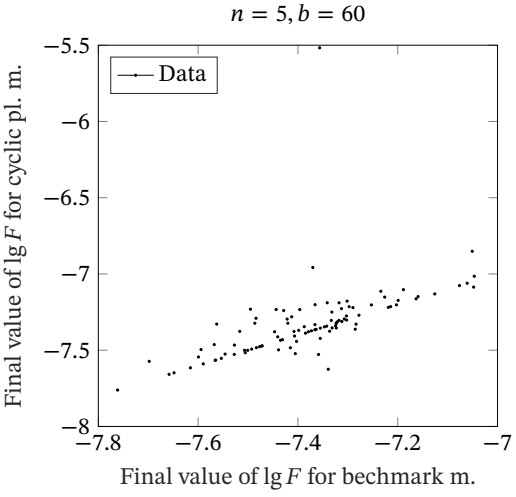

$n = 20, b = 40$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

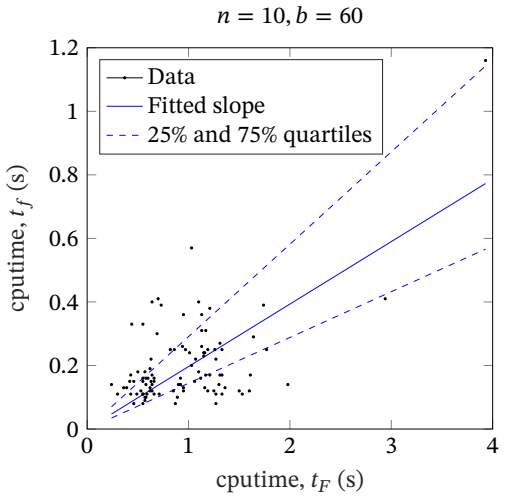

$n = 20, b = 40$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

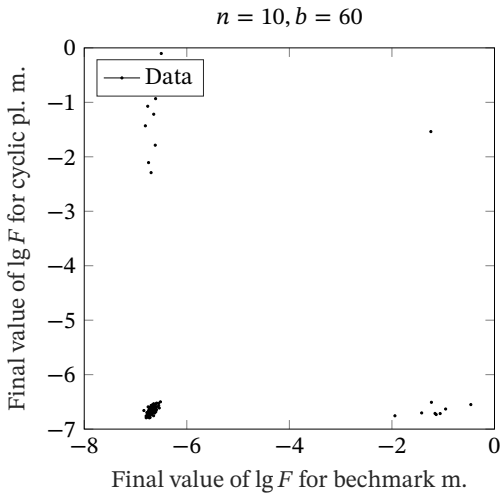

$n = 40, b = 40$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

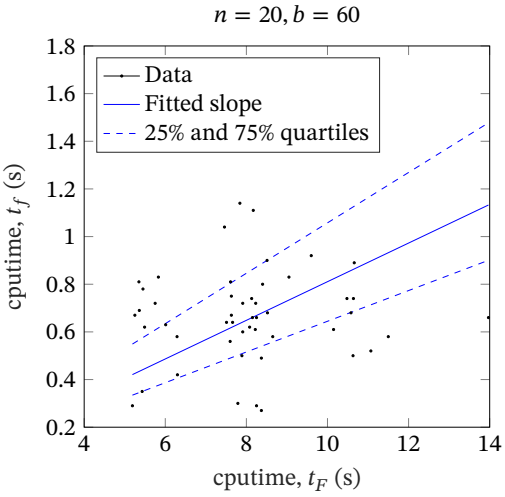

$n = 40, b = 40$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

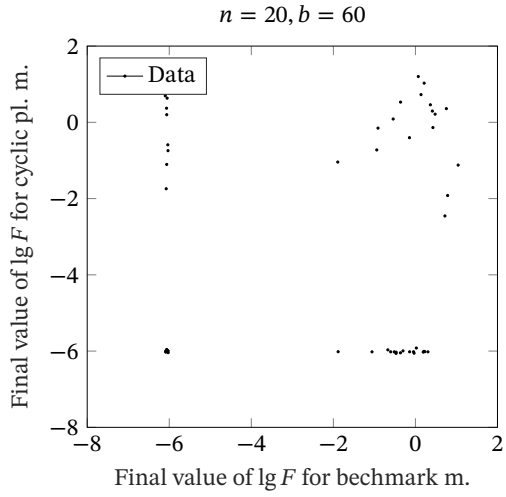

$n = 5, b = 60$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)

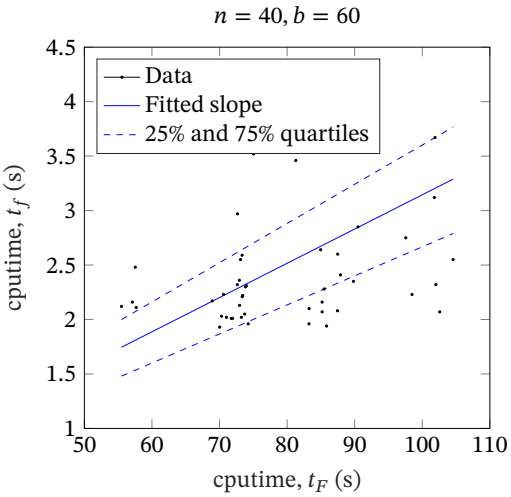

$n = 5, b = 60$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.

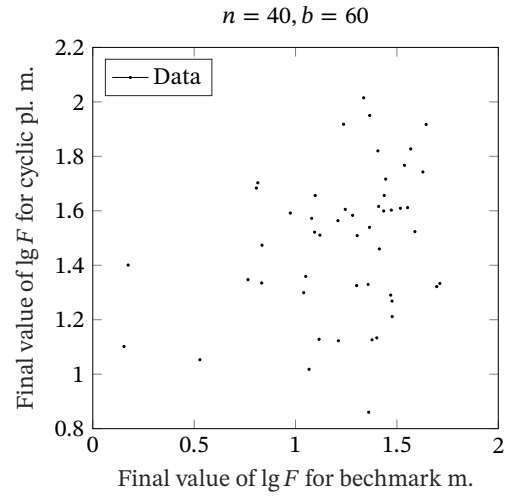

$n = 10, b = 60$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)


$n = 10, b = 60$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.


$n = 20, b = 60$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)


$n = 20, b = 60$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.


$n = 40, b = 60$
Data
Fitted slope
25% and 75% quartiles
cputime, t_f (s)
cputime, t_F (s)


$n = 40, b = 60$
Data
Final value of lg F for cyclic pl. m.
Final value of lg F for bechmark m.