Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 2. pp. 255–262.

Circle Covering and its Applications for Telecommunication Networks[*]

Endre Palatinus, Balázs Bánhelyi

University of Szeged
Institute of Informatics

Abstract

Circle covering is the dual problem of circle packing. We have used an interval arithmetic algorithm to test whether a given setting of circles completely covers the unit square. A new branch-and-bound–based method was developed for this problem. We also examined our algorithm in a parallel environment, and it proved to be well scalable. We applied our method to a telecommunication-related problem: covering Hungary with terrestrial radio signals.

Keywords: Circle covering, interval arithmetic, reliable computing, parallel computing.

1. Introduction

The circle packing problem has attracted much attention in the last century, and a variant called packings of equal circles in a square receives attention even nowadays [7]. The objective of it is to give the densest packing of a given number of congruent circles with disjoint interiors in a unit square.

However, its dual problem, the circle covering has not been exhaustively studied so far. We aim to find the "sparsest" covering of the unit square with a given number of congruent circles with overlapping interiors allowed. By sparsest we mean the total covering of the square with congruent circles of minimal radii (see [6]).

The main difficulty of the problem is the uncertainty of our computations caused by the finite precision of computers. To overcome this, we have used interval arithmetic to test whether a given setting of circles covers the unit square completely. We developed a branch-and-bound–based method for the previously mentioned

[*]This research was partially supported by the TÁMOP-4.2.2/08/1/2008-0008 program of the Hungarian National Development Agency.

problem. Our main focus, however, is to examine our method in a parallel environment, when the phases of our method can be executed concurrently on several CPU cores.

To test the efficiency of our method we also applied it to a telecommunication-related problem: We would like to find an optimal covering of Hungary with TV-stations for terrestrial signal given the positions of the broadcasting antennas. Similar problems were solved without reliable computing in [3].

2. The Branch-and-Bound–Based Parallelised Testing Method

The following algorithm is a simplified B&B method, which can reliably prove a given property of all the points of a given multidimensional interval utilising multiple CPU cores in order to reduce the running time [2].

It takes a multidimensional interval as its input. It has two constants: ϵ, which is the minimal allowed size of the subintervals, and $thread\_max$, the maximal number of concurrently running threads. Its output is true, if all points of the given interval have the given property, otherwise it returns a subinterval that contradicts the considered property.

ALGORITHM 1. *The parallelised checking routine*

1. step: Increase the number of running threads by 1.

2. step: Push the input interval into the stack.

3. step: Take out an interval, v of the stack.

4. step: Compute the widest side of the interval v.

5. step: Determine whether the interval v has the property considered using the reliable procedure.

6. step: If it does not, then:
If the widest side of the interval v is smaller then ϵ, then:

- Print v and continue with Step 8.

else:

- Subdivide v along its widest side, and push the two subintervals into the stack
- If the number of concurrently running threads is less than $thread\_max$, then start a new thread with one of the intervals in the stack.

7. step: If the stack is not empty, continue with Step 3.

8. step: Decrease the number of running threads by one.

9. step: Wait until all threads started by the current thread terminate.

10. step: If there is an interval printed which does not have the property considered, or any of the threads have found an interval not satisfying the considered property, then the algorithm terminates.

11. step: Print that the interval has the considered property and terminate.

The input interval does not have the considered property in the following cases:

- There is a subinterval printed which does not have the property considered
- Any of the threads have found a subinterval smaller than ϵ not satisfying the considered property.

Otherwise the interval has the considered property. In this case we have split the interval into subintervals all having the desired property.

The correctness of the non-parallelised version of this algorithm is proved in [1]. The previous algorithm terminates after a finite number of iteration steps with positive answer, if the considered condition is fulfilled. The B&B method produces independent subproblems which makes the parallelisation possible. It is easy to see that this parallelised technique considers all subintervals, so this method is also correct mathematically. We expected an increase in performance proportional to the number of CPU cores due to the multi-threaded execution.

There were a number of implementation issues to consider. Every thread waits in suspended mode until all of its child threads terminate, therefore they are not considered running threads. The procedure is deadlock-proof, since we applied locks on the mutually used variables, which store the number of concurrently running threads and denote if any intervals not satisfying the desired property have been found. We used interval arithmetic to overcome the uncertainty of our computations, thus it is computationally reliable. Where interval arithmetic was needed, we have used the programming language C-XSC (see [5]) in our code, which has a widely recognised C++ class library for extended scientific computing.

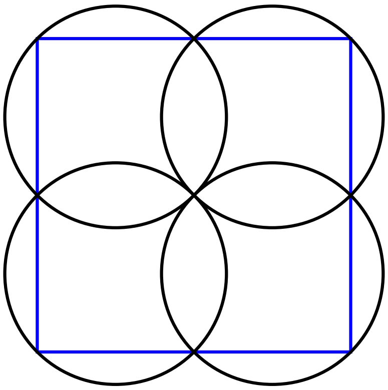
3. The Circle Covering Problem

The problem is to arrange some congruent circles to cover the unit square with minimal overlapping. This is an optimization problem, where the aim is to find a covering arrangement with minimal circle radii for a given number of circles. Optimal solutions have been found (see [4]) only for small numbers, and their correctness have been proved mathematically. With larger number of circles the amount of arrangements increases drastically, and this makes it very challenging to find the optimal solution and also to mathematically prove its optimality.

It makes sense to write computer programs to solve this problem, but the multiple symmetry of the arrangements and the uncertainty of the results of arithmetic operations performed by the computer impose great difficulties. One should not use real arithmetic in computations testing if a given arrangement fully covers the unit square. To overcome these we applied the B&B-based parallelised testing method introduced in Section 1.

The testing method applied to the circle covering problem works as follows: The input interval is the unit square itself. The desired property of an interval is to be covered completely by the circles. This is not always decidable due to the finite precision of our arithmetic, therefore all circles are considered to be open sets on the plane. This way an interval is covered by a circle, if $\sup(x - center) < \inf(r^2)$ holds for every point of the interval, where x denotes a point of the interval, *center* denotes the center of the circle, and r denotes the radius of the circle.

To study the efficiency of the method we used some simple test cases: n^2 circles on a regular $n \times n$ grid, where every circle touches its diagonal neighbours. Actually they have two intersections very near to each other because the radii are increased by $\epsilon_r > \epsilon$, otherwise it would not be possible to determine whether the point where they touch each other is covered by any of the circles. Some examples of such arrangements can be seen on Figure 1.


(a) $n = 2$

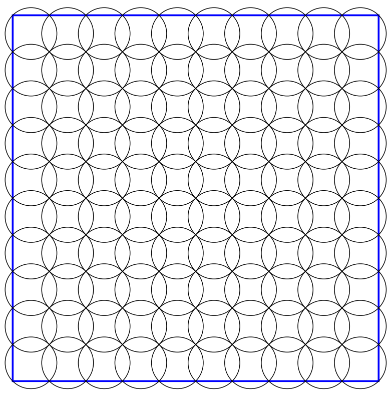
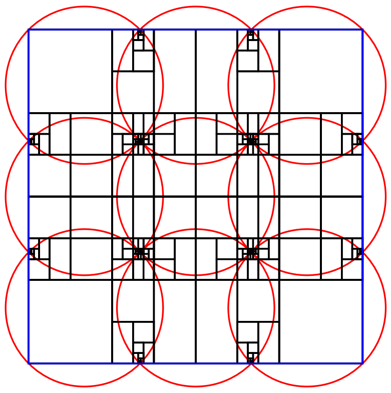

(b) $n = 10$

Figure 1: Test cases for the circle covering problem.

Illustrations of the results of the algorithm on our simple test cases can be seen on Figure 2. The blue boxes represent the unit square, and the black boxes show the subintervals created by the B&B method. In the very special case, when n is a proper power of 2, the smallest subintervals created are those that are the inscribed squares of the circles. As mentioned before, the radii are increased by ϵ_r, therefore all circles completely cover their inscribed squares and these intervals are not split further. This makes these special test cases very easy to solve, since the run times are very small. On the other hand, when n is not a power of 2, the B&B method does not create any subintervals, which are inscribed squares of the circles, since it always divides the intervals along their widest sides into two equal parts. In this case more subintervals are created near the intersections of the circles, since these points have a neighbourhood of the order of ϵ_r, which is covered by the circles having the intersection there.


(a) $n = 3$

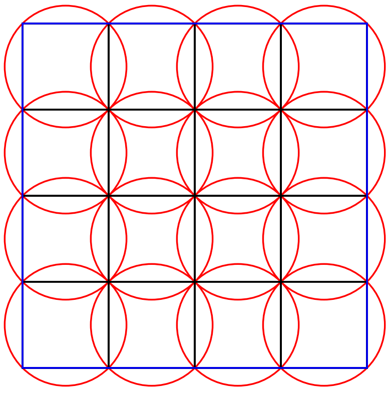

(b) $n = 4$

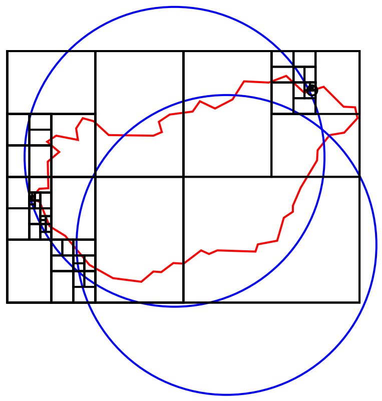
Figure 2: The results of the B&B testing method.

4. Covering Hungary with Terrestrial Radio Signals

Deciding whether a unit square is covered by a circle or not was a relatively easy task, and not of particular industrial or practical interest. However, covering more complex objects, especially concave ones makes the problem harder and more realistic. Therefore we will look at a telecommunication related problem, where our previous results can be utilised: We are given the borders of Hungary and the positions of the main broadcasting antennas. The goal is to decide if every point of Hungary is covered or not with the given radio signal.

Let us assume that all antennas consume the same amount of power, so we can utilise some of the ideas of the previous solution. The testing method applied to this problem works as follows: The input interval is any interval containing Hungary. The desired property of an interval is either to be covered completely by the circles, or not to have any common points with Hungary. The latter condition is the only novelty compared to the previous solution. Here we applied the basic geometrical algorithm for deciding whether a point is inside a polygon or not. The only modification needed was to replace points with intervals. The results of the testing method on two test cases can be seen on Figure 3. In the first one there are two radio stations which cover a large part of Hungary. The first interval found by the algorithm which is inside Hungary but not covered by any of the circles is near the upper right corner of the picture. In the second case there are five radio stations which fully cover Hungary.

Now that we have a proper testing method, we can apply a simple optimizer algorithm to find a near optimal size of the radii of the circular areas covered by the radio signals:


(a) Not covered

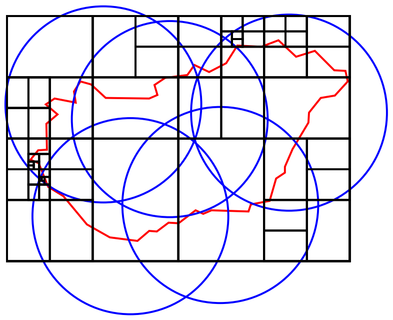

(b) Fully covered

Figure 3: The running of the B&B testing method.

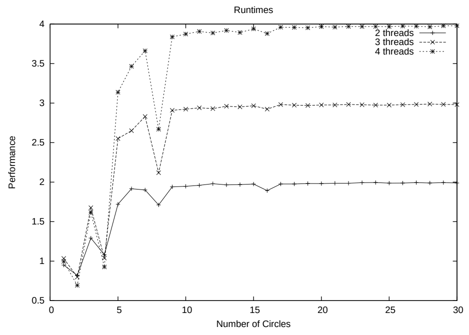
- Start with an initial solution.
- Let $delta\_circle\_radius = circle\_radius/2$.
- **While** $delta\_circle\_radius > 1$
  - Check whether the current set of circles cover Hungary.
  - **If** they do, then save the current setting and decrease the radii by $delta\_circle\_radius$.
  - **Otherwise** increase the radii by $delta\_circle\_radius$.
  - Let $delta\_circle\_radius = delta\_circle\_radius/2$.

5. Performance

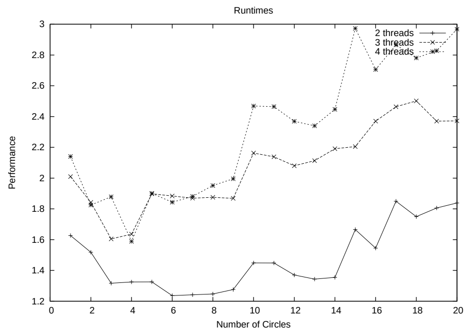
Our original goal was to examine our algorithm in a parallel computing environment, when the phases of the B&B method can be executed concurrently on several CPU cores. We performed tests on a Sun computer with 4 CPU cores, therefore the maximal number of concurrently running threads was 4. What we expected was that with larger circle counts the number of subintervals created during the testing method increases and multi-threaded execution improves its performance accordingly.

If we take a look at the performance benchmarks of the unit square covering problem on Figure 4, we can clearly see that from 11×11 circles on the increase in execution speed is proportional to the number of available CPU cores. Note that the performance of the multi-threaded execution is compared to the single-threaded case. As mentioned before, when n is a suitable power of 2, the runtime is smaller, therefore the increase in performance is negligible when n is less than 11.

The test results of the second problem, covering Hungary with terrestrial radio signals, do not show a clear connection between the number of threads and the increase in performance. One reason for that is that the number of circles means the number of broadcasting radio stations included in the testing, which are selected in a fixed order from a list. We did not experiment with alternative selections, therefore these are not considered to be optimal solutions in the general case. Another reason is the involvement of the simple optimiser mentioned before, which decreases performance as well. However, we could still benefit from the multi-threaded execution, when the number of circles increases.



(a) Covering the unit square



(b) Covering Hungary

Figure 4: The performance benchmarks.

In the future we would like to develop solutions to some generalisations of the circle covering problem, when different radius sizes would be allowed or when even the centers of the circles could be changed, too. The first setting would make the solution of the covering problem of Hungary more optimal, and the latter one could be used to improve the solutions of the unit square covering problem presented in [4].

References

[1] Bánhelyi, B., T. Csendes, and B.M. Garay: A Verified Optimization Technique to Locate Chaotic Regions of Hénon Systems, *Journal of Global Optimization,* **35** (2006), 145–160.

[2] Casado, L.G., J. A. Martinez, I. Garcia, and E.M.T. Hendrix :, Branch-and-Bound interval global optimization on shared memory multiprocessors, *Optimization Methods Software,* **23** (2008), 689–701.

[3] Das, G.K., S. Das, S.C. Nandy, and B.S. Shina: Efficient algorithm for placing a given number of base station to cover a convex region, *J. Parallel Distrib. Comput.* **66** (2006), 1353–1358.

[4] Friedman, E.: Circles Covering Squares, *http://www2.stetson.edu/ efriedma/circovsqu/* (2005)

[5] Hofschuster, K., W. Wedner: C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing, *Preprint BUGHW-WRSWT 2001/1, Universität Wuppertal,* (2001)

[6] Nurmela, K.J. and P.R.J. Östergard: Covering a square with up to 30 equal circles, *Helsinki University of Technology, Laboratory for Theoretical Computer Science Research Reports,* **62** (2000)

[7] Szabó, P.G., M.Cs. Markót, T. Csendes, E. Specht, L.G. Casado, and I. García: New Approaches to Circle Packing in a Square – With Program Codes. *Springer, Berlin,* 2007.

Balázs Bánhelyi
6701 Szeged, Hungary, P.O. Box 652.
e-mail: banhelyi@inf.u-szeged.hu