A Two-Level Search Strategy for Packing Unequal Circles into a Circle Container

Wen Qi Huang[1], Yu Li[2], Bernard Jurkowiak[2], Chu Min Li[2], and Ru Chu Xu[1]

[1] HuaZhong Univ. of Science and Technology Wuhan 430074, China
{wqhuang, Xu}@hust.edu.cn
[2] LaRIA, Université de Picardie Jules Verne, 5 Rue du Moulin Neuf, 80000 Amiens, France
{yli, jurkowia, cli}@laria.u-picardie.fr

Abstract. We propose a two-level search strategy to solve a two dimensional circle packing problem. At the first level, a good enough packing algorithm called $A1.0$ uses a simple heuristic to select the next circle to be packed. This algorithm is itself used at the second level to select the next circle to be packed. The resulted packing procedure called $A1.5$ considerably improves the performance of the algorithm in the first level, as shown by experimental results. We also apply the approach to solve other CSPs and obtain interesting results.

1 Introduction

Given a set of n circles of different radii $r_1, ..., r_n$ and a circle container, the two dimensional (2D) circle packing problem consists in finding the minimal radius r_0 of the container so that all the n circles can be packed into the container without overlap. If we find an efficient algorithm to solve this problem for a fixed circle container, we can solve the original problem by using some search strategies (e.g. dichotomous search) to reach the minimal radius of the container.

No significant published research appears to exist addressing this problem, except [2]. On the contrary, a lot of algorithms are proposed in the literature for packing equal circles into a circle container (see e.g. [4, 3]).

In this paper we propose a greedy approach called two-level search strategy to solve this problem.

2 The Two-Level Search Strategy

Consider a 2D Cartesian coordinate system. The coordinate of the center of the container is (0,0) and the coordinate of the center of the ith circle center is denoted by (x, y). We call *configuration* a pattern (layout) where m (≥ 2) circles have been already placed inside the container without overlap, and $n - m$ circles remain to be packed into the container.

A *legal action* (i, x, y) is the placement of circle i inside the container at position (x, y) so that circle i does not overlap any other circle and is tangent with 2 circles in the container (note that one of the 2 circles may be the container itself). There may be several legal actions for circle i.

Let (i, x, y) be a legal action, u and v be the two circles in the container and tangent with circle i if i is placed at (x, y). The degree λ of this action is defined as $\lambda = (1 - \frac{d_{min}}{r_i})$, where d_{min} is the minimal distance from circle i to other circles in the container (excluding u and v but including the container) and r_i is the radius of the circle i.

$$d_{min} = \min_{j \in \mathcal{M}, j \neq u, j \neq v} (|\sqrt{(x - x_j)^2 + (y - y_j)^2} - r_j| - r_i)$$

where $\mathcal{M}$ is the set of circles already placed inside the container (note that the container is itself included in $\mathcal{M}$).

```
Procedure CirclePacking(I)
Begin
  for k:=1 to n-1 do
    for l:=k+1 to n do
      Generate an initial configuration C using circles k and l;
      Generate the legal action list L;
      if (CirclePackingCore(C, L) returns a successful
          configuration)
      then stop with success;
  Stop with failure;
End.

Procedure CirclePackingCore(C, L)
Begin
  while (there are legal actions in L) do
    Compute the benefit of each legal action
    Select the legal action (i, x, y) with the maximum benefit;
    Modify C by placing circle i at (x, y);
    Modify L;
  Return C;
End.
```

Fig. 1. A generic circle packing algorithm

Our packing procedures *A1.0* and *A1.0Core* are respectively *CirclePacking* and *CirclePackingCore* shown in figure 1. At every iteration, *A1.0Core* places the circle with the largest λ and re-calculate the degree of all existing legal actions. If all circles are placed in the container without overlap, *A1.0Core* stops with success. If none of the circles outside the container can be placed into container without overlap ($\mathcal{L}$ is empty), it stops with failure.

However, given a configuration, *A1.0* only looks at the relation between the circles already inside the container and the circle to be packed. It doesn't examine the relation between the circles outside the container.

In order to more globally evaluate the benefit of a legal action and to overcome the limit of *A1.0*, we compute the benefit of a legal action using *A1.0Core* itself in the *CirclePackingCore* procedure to obtain our main packing algorithm called *A1.5*, the *CirclePackingCore* procedure becoming *A1.5Core* in this case.

In other words, *A1.5Core* is *CirclePackingCore* calling the subprocedure *BenefitA1.5* shown in Figure 2 to compute the benefit of legal actions.

```
Procedure BenefitA1.5(i, x, y, C, L )
Begin
  let C' and L' be copies of C and L;
  Modify C' by placing circle i at (x,y) and modify L';
  C'=A1.0Core(C',L') ;
  if (C' is successful) then return C'; else return density(C');
End.
```

Fig. 2. Subprocedure of Algorithm $A1.5$ for computing the benefit of a legal action

Given a copy $\mathcal{C}'$ of the current configuration $\mathcal{C}$ and a legal action (i, x, y), *BenefitA1.5* begins by placing the circle i in the container at (x, y) and calls *A1.0Core* to reach a final configuration. If *A1.0Core* stops with success then *BenefitA1.5* returns a successful configuration, otherwise *BenefitA1.5* returns *the density* (the ratio of the total surfaces of the circles inside the container to the surface of the container) of a failure configuration as the benefit of the legal action (i, x, y). In this manner, *A1.5* evaluates all existing legal actions and chooses the best.

Roughly speaking, under the current configuration, *the first level of the strategy* consists in choosing the best action (i, x, y) by using a simple heuristic. *The second level* uses the first level itself to select the next legal action. We call this approach *two-level search strategy*.

3 Experimental Results

A1.0 and *A1.5* are implemented in C and executed on an Athlon XP2000+ CPU under Linux system.

Given a set of circles we respectively use *A1.0* and *A1.5* to find the minimum container radius (r_{min}) so that the set of circles can be packed into the container without overlap.

Table 1 shows the minimum container radius of the two hard instances illustrated in figures 3 and 4 found by *A1.0* and *A1.5*. It is clear that *A1.5* is substantially more powerful. The resolution of other instances in our experimentation using *A1.0* and *A1.5* exactly gives the same conclusion. *A1.5* is also substantially better than the approach presented in [2].

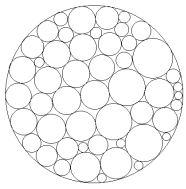

Fig. 3. Instance 1 ($n = 50, r_{min} = 380.00$)

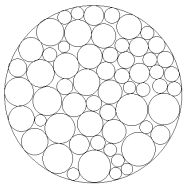

Fig. 4. Instance 2 ($n = 60, r_{min} = 522.93$)

Table 1. Minimum container radius found by *A1.0* and *A1.5* and runtime (in seconds) used by *A1.0* and *A1.5* to find the successful configuration with the container radius r_{min}.

Instance	*A1.0*		*A1.5*		Instance	*A1.0*		*A1.5*	
	r_{min}	*time*	r_{min}	*time*		r_{min}	*time*	r_{min}	*time*
1	381.05	2712s	380.00	5396s	2	527.57	4216s	522.93	6615s

4 Solving CSPs with the Two-Level Search Strategy: First Results

Constraint Satisfaction Problems (CSPs) involve the assignment of values to variables subject to a set of constraints.

Formally, a CSP is defined by a triplet (X, D, C) where X is a set of n variables $\{X_1, X_2, \ldots, X_n\}$, D is a set of n finite domains $\{D_1, D_2, \ldots, D_n\}$ with each D_i is a set of possible values $\{v_{i1}, \ldots, v_{ik_i}\}$ for X_i, and C is a set of m constraints between variables $\{C_1, C_2, \ldots, C_m\}$. A solution is an assignment of values to all variables such that all constraints are satisfied.

The circle packing problem can be considered as a CSP : X is the set of circles to be packed into the circle container, D_i is the set of all legal actions associated with circle i and C is the set of constraints saying that all circles should be placed in the container without overlap.

Consequently assigning a value (i, x, y) to X_i means the execution of the legal action (i, x, y) placing a circle i at position (x, y). $A1.0$ is just a search procedure where the benefit of an assignment $X_i = (i, x, y)$ for every unassigned variable X_i and every $(i, x, y) \in D_i$ is defined by degree λ. *A1.5* is similar except that the benefit of an assignment is evaluated using *A1.0*.

More generally, the two-level search strategy can be used to solve satisfiable CSPs or Max-CSP. The key issue is the definition of the benefit of a variable assignment $X_i = v$. At the first level, we use the value $0 - c$ as the benefit of $X_i = v$, where c is the total number of values in the domain of other unassigned variables conflicting with $X_i = v$. At the second level, the search procedure at the first level is itself used to evaluate the benefit of $X_i = v$, i.e., after assigning v to X_i, the search procedure at the first level runs until a final configuration, where either all variables are assigned, or the domain of each unassigned variable is empty. We use the number of assigned variables in the final configuration as the benefit of $X_i = v$.

Intuitively, we should choose an assignment to leave as many rooms as possible to other unassigned variables. Our current heuristic at the first level is very simple but not precise enough. However, with a quick implementation, this heuristic already gives us the first interesting results to solve Round Robin and n-queens problems. These two well-known CSPs appear to be quite hard for backtracking methods.

In the n-queens problem, one has to place n queens on a $n \times n$ chessboard so that no two queens attack each other. Although specific methods are known to solve this problem easily, it has been used extensively to test constraint satisfaction algorithms.

As in a backtracking approach, we define n variables with domain $\{1, 2, \cdots, n\}$, one for each queen to be placed. Intuitively, queen X_i is to be placed in some column in

ith row. We then define constraints stating that there are no two queens in any column or diagonal.

The search procedure at the first level solves some n-queens instances, but it is the search procedure at the second level that solves this problem up to 600 queens in less than 10 hours on a Athlon XP2000+ PC.

In the n teams (n even) Round Robin scheduling problem, one has to place the $(n-1)n/2$ meetings (all teams meet each other exactly once) on a $n/2$ row and $n-1$ column matrix, such that every team occurs exactly once in each column and no more than twice in each row. See [1] for a formal definition of this problem.

The Round Robin problem is challenging for integer programming or standard constraint satisfaction techniques, because of its explosive combinatorics. Linear programming is not able to find a solution for $n \geq 14$. Using the powerful C++ constraint programming library ILOG SOLVER, Gomes et al. [1] built a deterministic backtrack-style CSP engine able to find a solution for $n = 14$. Then the CSP engine has been reinforced by randomization with restart in choosing branching variables to find a solution for $n = 16$ in 1.4 hours and for $n = 18$ in 22 hours. Note that Gomes et al.'s results in [1] for this problem were among the best until 1999.

We use $(n-1)n/2$ variables, one for each meeting. At the first level, no solution is found even for $n = 6$. However, we are able to solve this problem at the second level for n up to 18 in one hour.

5 Conclusion

We have proposed a new heuristic and a two-level search strategy, from which an effective algorithm called *A1.5* is designed for packing unequal circles into a circle container. At the first level *A1.0* uses the new heuristic to select the next legal action. Then at the second level *A1.5* uses *A1.0* itself to select the next legal action. Experimental results show the effectiveness of this strategy.

The two-level search strategy can also be used to solve other CSPs. The first results obtained using simple heuristics are very encouraging. We are searching for new and more precise heuristics.

References

1. Gomes C.P., Selman B., Kautz H. *Boosting Combinatorial Search Through Randomization*, In proceedings of AAAI'98, 1998.
2. Huang W.Q., Li Y. Xu R.C., *Local Search Based on a Physical Model for Solving a Circle Packing Problem,* In proceedings of the 4th Metaheuristics International Conference (MIC'2001), Porto, Portugal, July 16-20, 2001.
3. Melissen H., *Densest packing of eleven congruent circles in a circle,* Geom. Dedicata 50 (1994) 15-25.
4. Reis G.E., *Dense packing of equal circles within a circle,* Math. Mag. 48 (1975) 33-37.