SOME APPLICATIONS OF CONTINUOUS VARIABLE NEIGHBOURHOOD SEARCH METAHEURISTIC (MATHEMATICAL MODELLING)

A Thesis Submitted for the Degree of

Doctor of Philosophy

by

Rima Sheikh Rajab

School of Information System, Computing and Mathematics

Brunel University

Abstract

In the real world, many problems are continuous in nature. In some cases, finding the global solutions for these problems is difficult. The reason is that the problem's objective function is non convex, nor concave and even not differentiable. Tackling these problems is often computationally too expensive. Although the development in computer technologies are increasing the speed of computations, this often is not adequate, particularly if the size of the problem's instance are large. Applying exact methods on some problems may necessitate their linearisation. Several new ideas using *heuristic* approaches have been considered particularly since they tackle the problems within reasonable computational time and give an approximate solution.

In this thesis, the variable neighbourhood search (VNS) metaheuristic (the framework for building heuristic) has been considered. Two variants of variable neighbourhood search metaheuristic have been developed, *continuous variable neighbourhood search* and *reformulation descent variable neighbourhood search.* The GLOB-VNS software (Drazić et al., 2006) hybridises the Microsoft Visual Studio C++ solver with variable neighbourhood search metaheuristics. It has been used as a starting point for this research and then adapted and modified for problems studied in this thesis. In fact, two problems have been considered, *censored quantile regression* and *the circle packing problem.* The results of this approach for censored quantile regression outperforms other methods described in the literature, and the near-optimal solutions are obtained in short running computational time. In addition, the reformulation descent variable neighbourhood search variant in solving circle packing problems is developed and the computational results are provided.

Contents

List of Figures

List of Tables

Acknowledgments

I present my best thanks to Allah (my God), whose response always helped me and given me the power to complete my work.

I would like to express my deep gratitude to my supervisor, Professor Dr Nenad Mladenović, for supporting my research over the years. I also want to thank Dr Mladenović for his inspiration, competent guidance and encouragement were of tremendous value for my scientific career. Besides, many thanks to Dr Milan Drazić for providing the C++ code `GLOB`.

I would also like to thank the School of Information Systems, Computing and Mathematics (SISCM) at Brunel University for the extensive support during my PhD research years and for providing me with everything I need for my research.

Finally, many thanks to my family, my husband Raed Alsalem and my son Ebaa for their supprort during these years. Also, I want to pass my thanks to my parents (my mother and father) for their huge support and love.

Related Publications

- **Published papers**

  R. S. Rajab, M. Drazić and N. Mladenović. Reformulation descent for solving circle packing problems. A Proceedings volume from the Information Control Problems in Manufacturing International Symposium, 2011. ISBN: 978-86-403-1168-7, pages 397-400.

- **Papers submitted for publication**

  R. S. Rajab, M. Drazić, N. Mladenović and K. Yu. Fitting Censored Quantile Regression by Variable Neighbourhood Search. Submitted to Statistics and Computing 2011.

Introduction

0.1 Optimisation problem

An optimisation problem P can be defined as

$$\min\{f(x) \mid x \in X, X \subseteq S\} \tag{1}$$

where S represents the solution space, and X denotes the feasible set. x and f, where $f : S \to R$, are the feasible solution and the real valued function respectively. x is called a feasible solution for (1) if $x \in X$, and it is infeasible if $x \in S$ but $x \notin X$. The optimisation problem P is called an infeasible optimisation problem if there is no feasible solution x. Otherwise, P is a feasible one.

If S is a finite but large set, or infinite but enumerable, the P in (1) is called a combinatorial or discrete optimisation problem. If $S = \mathcal{R}^n$, P is called a continuous optimisation problem. The formulation (1) is defined as the minimisation problem. The maximisation problem can be defined easily by using $\max f(x) = -\min(-f(x))$.

The solution x' is a local minimum for the problem (1), if there exists $\delta > 0$ such that for all feasible solutions x with $\| x' - x \|_2 \leq \delta$, satisfy

$$f(x') \leq f(x) \qquad \forall x \in X. \tag{2}$$

The local maximum is given if there exists $\delta > 0$ such that for all feasible solutions x with $\| x' - x \|_2 \leq \delta$, satisfy

$$f(x') \geq f(x) \qquad \forall x \in X. \tag{3}$$

The local minimum is a global minimum $x^* \in X$ for (1) if

$$f(x^*) \leq f(x) \qquad \forall x \in X. \tag{4}$$

For a maximisation problem, the global maximal solution or global maximum satisfies the condition

$$f(x^*) \geq f(x) \qquad \forall x \in X. \tag{5}$$

The real world problems come from industry, transportation and management, where optimisation models may be discrete or continuous, as mentioned above.

0.2 Continuous optimisation

In the case of a constrained problem, the optimisation problem (1) can be formulated as

$$\min f(x) \tag{6}$$

subject to

$$g_i(x) \geq 0, \qquad \forall i = 1, \ldots, m \tag{7}$$

$$h_i(x) = 0, \qquad \forall i = 1, \ldots, p \tag{8}$$

where the functions f, h_i and g_i are continuous ones, and $x \in X$.

This problem is called a convex optimisation problem, if the objective function $f(x)$ and the constrained functions $g_i(x)$, $\forall i = 1, \ldots, m$ are convex functions and the feasible solution set X is a convex set. Besides, the $h_i(x)$, $\forall i = 1, \ldots, p$ are affine functions (Roberts and Varberg, 1973).

Definition 1 *The function $h(x)$, where $h : R^m \to R^n$ is called an affine function if there is a linear function $L : R^m \to R^n$ and a vector $b \in R^n$ such that*

$$h(x) = L(x) + b \quad \forall x \in R^n \tag{9}$$

The optimisation problem (1) is called an unconstrained optimisation problem if it does not have any constraint (7 and 8), i.e., if $X = R^n$.

0.2.1 Unconstrained methods

As above, the optimisation problem (1) is an unconstrained problem, and it is convex if the objective function $f(x)$ is a convex function. Note that $X = R^n$ is a convex set.

There are three ways to solve convex unconstrained optimisation problem: by *direct search methods*, *first-order methods* or *second-order methods*. Details of each method are discussed below.

Direct search methods

The direct search methods solve the problem without using derivatives. If the gradient of f is not available, some direct search methods attempt to estimate it. The f gradient is determined by evaluating its value at several points (Zhao et al., 2009). Generally the direct search methods select one by one a sequence of points in X. These points converge to the local optimum of $f(x)$. The first point is chosen by the analyst due to the information of the problem. If there is not enough information, it can be chosen randomly. Then each other point is generated by some routine or strategy. *Nelder-Mead method* (Nelder and Mead, 1965) is an example of the direct search methods.

Nelder-Mead method (or downhill simplex method) does not require the function derivative, it just needs the function evaluations. It was first introduced in (Nelder and Mead, 1965). The simplex is a geometrical figure. It consists of a $n + 1$ $(x_1, x_2, \ldots, x_{n+1})$ vertices in n dimension. If any point has been taken as an origin, then the rest of n points are defined as the vector directions, where the n-dimension vector space will be extended. For keeping the shape of the simplex unchanged, it chooses only one point in the directions.

If the initial solution has been chosen randomly, the n points can be generated by this formula

$$x_i = x_0 + \lambda e_i \tag{10}$$

where e_i's are n unit vectors and λ is a constant. The λ is guessed by the user and it depends on the problem's length scale (Press et al., 1989). The initial solution x_0 then changes through a sequence of geometry transformation (reflection, expansion, contraction

and multi-contraction) (Zhao et al., 2009).

The Nelder-Mead algorithm is started by choosing the worst point of the objective function (see Figure 1). This point is called high, then one generates another point due to the worst point (see Figure 2a). This operation is called a reflection. This point is given by

$$x_r = (1 + \alpha)\bar{x} - \alpha x_{n+1} \tag{11}$$

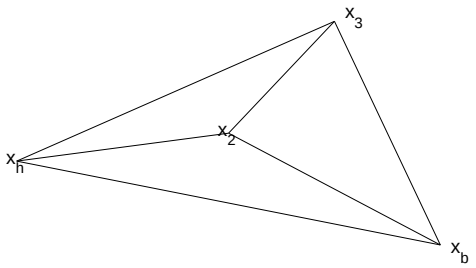


Figure 1: Initial simplex, where x_h represents the highest point and x_b represents the lowest point

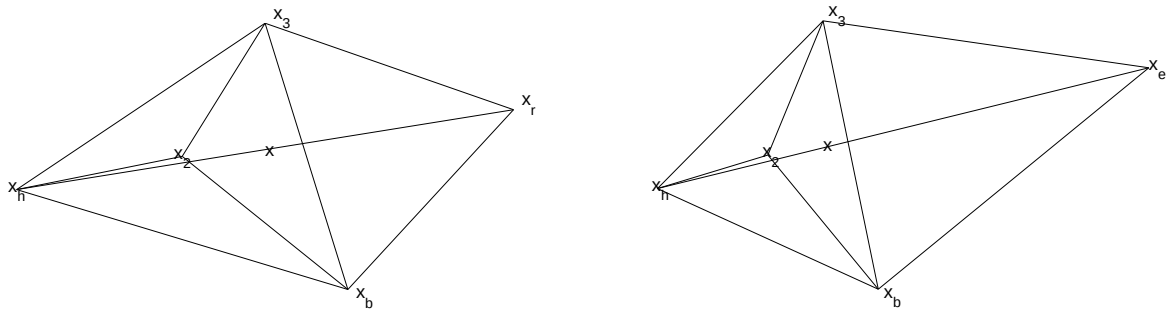
where α is constant and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. If the reflection point is not better than the other points, the algorithm reflects again with the new worst point. Otherwise, the simplex expands in this direction. The expansion operation can be seen in Figure 2b and it is given by

$$x_e = (1 - \beta)\bar{x} + \beta x_r \tag{12}$$

where β is also a constant. The contraction happens, if the reflected point is as good as the worst point. It's formula is given by

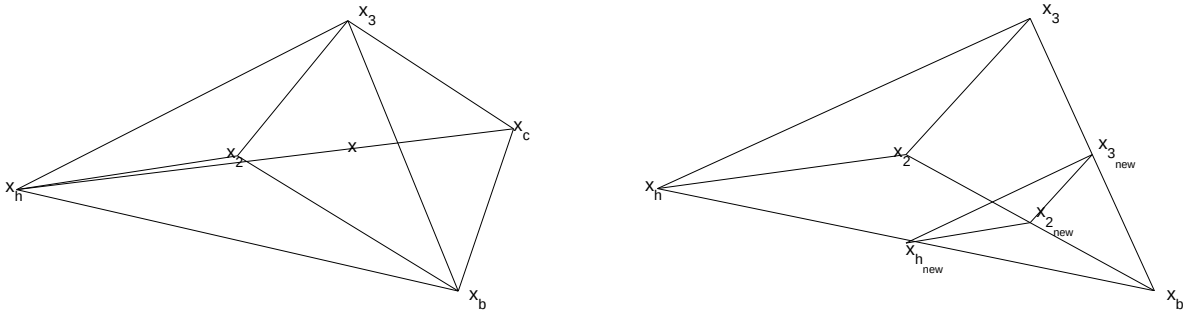
$$x_c = (1 - \gamma)\bar{x} + \gamma x_n \tag{13}$$

where γ is also a constant. It is illustrated in Figure 3a. However, if the worst point is better than the contracted point, the multi-contraction is applied. Moreover, each rejected point x_i



(a) A Reflection (b) A Expansion

Figure 2: The reflection and expansion steps in Nelder-Mead method



(a) A Contraction (b) A Multi-contraction

Figure 3: The contraction and multi-contraction steps in Nelder-Mead method

in the simplex will be exchanged by $\frac{x_i+x_1}{2}$ for each contraction step, where x_1 is a low point in the simplex (see Figure 3b).

The Nelder-Mead algorithm is given in Algorithm 1 (Press et al., 1989)

function `Nelder-Mead`(X, f);

1 **Order.** Order the $n+1$ vertices of X to satisfy $f(x_1) \leq f(x_2) \ldots \leq f(x_{n+1})$.

2 **Reflect.** Compute the *reflection point* x_r as $x_r = \bar{x} + \alpha(\bar{x} - x_{n+1})$, where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the centroid of the n best points (all vertices except for x_{n+1}).
If $f(x_1) \leq f(x_r) < f(x_n)$, accept the reflected point x_r, terminate the iteration.

3 **Expand.** If $f(x_r) < f(x_1)$, calculate the *expansion point* $x_e = \bar{x} + \beta(x_r - \bar{x})$.
If $f(x_e) \leq f(x_r)$, accept the expanded point x_e and terminate; otherwise accept x_r and terminate the iteration.

4 **Contract.** If $f(x_r) \geq f(x_n)$, perform a *contraction* between $\bar{x}$ and the better of x_{n+1} and x_r.

(a) *Outside.* If $f(x_r) < f(x_{n+1})$, then *outside contraction*: $x_c = \bar{x} + \gamma(x_r - \bar{x})$.
If $f(x_c) \leq f(x_r)$, accept x_c and terminate; otherwise go to *Multi-contract* step.

(b) *Inside.* If $f(x_r) \geq f(x_{n+1})$, then *inside contraction*: $x_c = \bar{x} - \gamma(\bar{x} - x_{n+1})$.
If $f(x_c) \leq f(x_{n+1})$, accept x_c and terminate; otherwise go to *Multi-contract* step.

5 **Multi-contract.** Evaluate f at the n points $v_i = x_1 + \delta(x_i - x_1), i = 2, \ldots, n+1$.
The (unordered) vertices at the next iteration consist of $V = \{x_1, v_2, \ldots, v_{n+1}\}$; set $X = V$.

Algorithm 1: Nelder-Mead Algorithm

First-order methods

The First-order methods are known as gradient methods (Simmons, 1975a). These methods attempt to find the answers for two questions during the search:

- In what direction do we move next?
- How far?

To give more details about answering these questions, let us suppose that x_i is the latest member in a given sequence of points, and $f(x_i)$ is the objective function, where $x \in R^n$. The next point x_{i+1} is given by

$$x_{i+1} = x_i + \theta_i s_i \tag{14}$$

where $s_i \in R^n$ is the direction of the gradient vector evaluated at x_i, and it is given by

$$s_i = \bigtriangledown f(x_i) \tag{15}$$

where the equation (15) gives the direction to find the local maximum. However, if we need to find the local minimum, s_i is given by

$$s_i = - \bigtriangledown f(x_i) \tag{16}$$

To answer the question of "how far?", we should find $\theta_i \geq 0$. Moreover, the function $f(x_i)$ will move in a gradient direction until it starts to decrease. The desired θ_i step length is the smallest positive θ_i, which is satisfied by the equation

$$\frac{dg(\theta_i)}{d\theta_i} = 0 \tag{17}$$

This should be accomplished by maximising the function

$$g(\theta_i) \equiv f(x_i + \theta_i \bigtriangledown f(x_i)) \tag{18}$$

where θ_i, in general, determines the distance moved in the s_i direction between x_i and x_{i+1}, and it is called the step length.

The Hooke-Jeeves method is one of the first-order methods (Hooke and Jeeves, 1961). It combines both exploratory and pattern moves. The exploratory move tries to find the best point around the current one. Then, these two points are used to make a pattern move (Bath et al., 2004). The Hooke-Jeeves algorithm is given in Algorithm 2 (Babu et al., 2008).

function `Hooke-Jeeves`(X, f);

1 Choose a starting point x_0, variable increments $\triangle_i$ $(i = 1, \ldots, n)$, and a step reduction $\beta > 1$.

2 Choose termination parameter ε, $k = 0$ and choose boundary conditions for variables.

3 Apply exploratory move based on x_k.

4 **Exploratory move**. If the current solution is x_k, suppose the new solution x_{ki} is perturbed by $\triangle_i$. Then set $i = i + 1$ and $x = x_k$.

5 Calculate $f = f(x)$, $f^+ = f(x_{ki} + \triangle_i)$ and $f^- = f(x_{ki} - \triangle_i)$.

6 Find $f_{\min} = \min(f, f^+, f^-)$. Then set the corresponding $f(x_{new})$ to $f_{\min}$.

7 If $(i < n)$ then go to step 5, else go to step 8.

8 If $(x_{new} = x_k)$ then set $x_{k+1} = x_{new}$ and go to step 10. Otherwise, go to step 9.

9 If $(\triangle > \varepsilon)$ then $\triangle_i = \triangle_i/\beta$ for $i = 1, \ldots, n$, and go to step 10. Otherwise, the algorithm will terminate.

10 Set $i = i + 1$, and apply pattern move.

11 **Pattern move**. The new point will be found by jumping from the current best point x_i along the direction s_i between the previous best point x_{i-1} and the current based point x_i by using the formula $x_{i+1} = x_i + (x_i - x_{i-1}) = x_i + s_i$.

12 Apply exploratory move on x_{i+1}. Let the result be $x_{new(i+1)}$.

13 If $(f(x_{new(i+1)}) < f(x_i)$ then go to step 11. Otherwise, go to step 9.

Algorithm 2: Hooke-Jeeves Algorithm

Second-order method

The second-order method is also known as Newton's method (Bazarra et al., 1993a). It can only be applied if the function is a twice differentiable one. It is based on exploiting the quadratic approximation q of the function $f(x)$ at a given point x_i. The formula of quadratic approximation is given by

$$q(x) = f(x_i) + f^{'}(x_i)(x - x_i) + \frac{1}{2} f^{''}(x_i)(x - x_i)^2 \tag{19}$$

The new point x_{i+1} is satisfied when the quadratic approximation q is equal to zero. This leads to the next equation

$$f^{'}(x_i) + f^{''}(x_i)(x_{i+1} - x_i) = 0 \tag{20}$$

So that

$$x_{i+1} = x_i - \frac{f^{'}(x_i)}{f^{''}(x_i)} \tag{21}$$

This procedure will stop if $\mid x_{i+1} - x_i \mid < \varepsilon$ or $\mid f^{'}(x_i) \mid < \varepsilon$, where ε is a termination factor. In general, Newton's method can be given by

$$x_{i+1} = x_i - [f'(x_i)]^{-1} f(x_i) \tag{22}$$

which is a classic method for solving the nonlinear equation $f(x) = 0$, where $f : R^n \to R^n$ is a continuously differentiable function.

0.2.2 Constrained optimisation

The constrained optimisation problem is formulated as shown in (6). It can be solved by transforming the problem into a sequence of unconstrained problems. Moreover, there are three techniques for solving constrained problems, *the Lagrangian method*, *the exterior point method* or *the interior point method* (Simmons, 1975b). These techniques are built with basic strategies to transform the problem from a constrained problem to an unconstrained one. More details of each type are given below.

The Lagrangian method

The Lagrangian method can be applied to problem (6) (where $g_i(x) \leq 0$, $\forall i = 1, \ldots, m$), if the functions f, $g_i, \forall i = 1, \ldots, m$, $h_i, \forall i = 1, \ldots, p$ are twice differentiable. Also, X is nonempty set (Bazarra et al., 1993b). The Lagrangian function of the problem can be written as

$$\varphi(x, \lambda, \upsilon) \equiv f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} \upsilon_i h_i(x) \tag{23}$$

where$\lambda \in R^n$,and $,\upsilon \in R^p$ are weight factors. Moreover, if we have conditioned $\bar{\lambda}$ and $\bar{v}$, it can be defined as the *restricted Lagrangian function*

$$\phi(x) \equiv \varphi(x, \bar{\lambda}, \bar{v}) \equiv f(x) + \sum_{i=1}^{m} \bar{\lambda_i} g_i(x) + \sum_{i=1}^{p} \bar{v_i} h_i(x) \tag{24}$$

where $I = \{i : g_i(x) = 0\}$ is the index set of the binding inequality constraints at x. The dual feasibility condition is

$$\bigtriangledown f(x) + \sum_{i=1}^{m} \bar{\lambda_i} \bigtriangledown g_i(x) + \sum_{i=1}^{p} \bar{v_i} \bigtriangledown h_i(x) \tag{25}$$

The new problem (24) is an unconstrained problem, and it can be solved by one of the unconstrained methods above.

The interior point method

The Interior point method or Barrier method has been applied to nonlinear constrained problems of the form

$$\min f(x) \tag{26}$$

subject to

$$g_i(x) \leq 0, \qquad \forall i = 1, \ldots, m \tag{27}$$

where $f(x)$ and $g_i(x)$ $(\forall i = 1, \ldots, m\ )$ are continuous functions. They have first partial derivatives where there exists at least one point $\hat{x}$ that can satisfy $g_i(\hat{x}) < 0$ $(\forall i = 1, \ldots, m)$. This indicates that the interior of the feasible set is non-empty. If the feasible set X can be defined as

$$X \equiv \{x \mid g_i(x) \leq 0, \qquad \forall i = 1, \ldots, m\} \tag{28}$$

then the interior set X_0 can be defined as

$$X_0 \equiv \{x \mid g_i(x) < 0, \qquad \forall i = 1, \ldots, m\} \tag{29}$$

where the boundary of X includes all points of x that lie on X not in X_0. The barrier approach to solve problem (26) can be formulated as

$$\min C(x, r) \equiv f(x) + \frac{1}{r} B(x) \tag{30}$$

where r is a positive parameter and $B(x)$ is a barrier function. Moreover, the barrier function $B(x)$ is non positive at each point in the interior and decreases to $-\infty$ at the boundary of X. A typical barrier function and the most used can be given as

$$B(x) = -\sum_{i=1}^{m} \frac{1}{g_i(x)}, \qquad x \in X \tag{31}$$

Also, it can be written as a logarithmic utility function (Luenberger and Ye, 2008a)

$$B(x) = -\sum_{i=1}^{m} \lg[-g_i(x)] \tag{32}$$

Furthermore, the interior point method (Griva, 2004; Luenberger and Ye, 2008b) for solving problem (26) with a classical log barrier function can be written as

$$B(x, r) \equiv f(x) - r \sum_{i=1}^{m} \lg[-g_i(x)] \tag{33}$$

subject to

$$A\ x = b \tag{34}$$

where $r = r^k > 0$, $k = 1, \ldots, m$ with $r^k > r^{k+1}$, $r^k \longrightarrow 0$ is a barrier parameter, and the r^k could be predetermined. Moreover, we have

$$r^{k+1} = \lambda\ r^k, \qquad 0 < \lambda < 1 \tag{35}$$

and it can be assumed that the original problem has a feasible interior-point solution x_0, it can be satisfied that

$$A\ x_0 = b \qquad and \qquad g(x_0) < 0 \tag{36}$$

where A is a matrix with full row rank. If we have a fixed r and by using $D_i = \frac{r}{g_i}$, then the optimality conditions of problem (33) are given by

$$-D\ g(x) = r1 \tag{37}$$

$$A\ x = b \tag{38}$$

$$-A^T\ y + \bigtriangledown f(x)^T + \bigtriangledown g(x)^T\ d = 0, \tag{39}$$

where $\bigtriangledown g(x)$ is a Jacobian matrix and $D = diag(d)$, which means D is a diagonal matrix whose diagonal elements are d. If $f(x)$ and $g_i(x)$, $\forall i = 1, \ldots, m$ are convex functions, then $f(x) - r\sum_{i=1}^{m} \lg[-g_i(x)]$ is also a convex function and there is a unique minimum solution for this problem.

The exterior point method

The exterior point method is sometimes called the penalty method. Penalty techniques are used to solve problem (26), where the functions $f(x)$ and $g_i(x)$ ($\forall i = 1, \ldots, m$) are continuous ones and they have continuous first partial derivatives. The feasible set X can be formulated as

$$X \equiv \{x \mid g_i(x) \leq 0, \qquad \forall i = 1, \ldots, m\} \tag{40}$$

Then, the penalty approach to solve this problem can be written as

$$\min D(x, r) \equiv f(x) + rP(x) \tag{41}$$

where r is a positive parameter and $P(x)$ is a penalty function. The penalty function is zero for any point of X and negative at all other points of S. The most famous penalty function (Luenberger and Ye, 2008a) is given by

$$P(x) = \frac{1}{2}\sum_{i=1}^{m}[\max\{g_i(x), 0\}]^2 \tag{42}$$

0.2.3 Global methods

The global minimum for the problem in (1) is defined above in (4). If the problem is a convex problem, then the local minimum is a global minimum. However, if the problem is not a convex (nor concave) one, that means the local optimum is not a global optimum. There are therefore two possibilities to solve the optimisation problem by using exact or approximate global methods.

The exact algorithms solve the problems exactly. They guarantee to find the optimal solution with a proof of its optimality. The most exact methods used are branch-and-bound,

dynamic programming, Lagrangian relaxation based methods, branch-and-cut, branch-and-price and branch-and-cut-and-price, etc. The running time for solving the OP by exact methods increases due to the size of the problem, where exact algorithms almost fit in small and moderate size problems. However, in some cases it could take days or more to find the optimal solution even in small or moderate size problems (VoB, 2001). Furthermore, in large instance size problems, the exact methods can not prove optimality. To solve this problem, the approximate methods were introduced, which are classified as classical heuristics. The classical heuristic methods are explained in Section (0.3).

0.3 Classical heuristics

Classical heuristic is a new idea introduced in the sixties to deal with operational research problems (OR). Its name is derived from the Greek word. *Heuristic* from the verb *heuriskein*, meaning "to find". The approximate or heuristic algorithm does not guarantee the optimal solution for the input problem. It just gives a feasible solution. For instance, if $x' \in X$ is a feasible solution for the instance P of an optimisation problem (1), where the optimal solution for P is x^*, one would like x' to be identical to x^*. However, a heuristic can not prove optimality, it hopes that x' is close to x^*.

Many optimisation problems are NP-hard (Garey and Johnson, 1979). The NP-hard problems are the problems that can not be solved by a polynomial time algorithm, unless P=NP (for more details the reader is referred to Appendix A). Moreover, in some problems, which are solvable by a polynomial time algorithm, the power of that polynomial could be very large. In this situation, it needs an unreasonable time to be solved. This is another case where heuristic methods are in need.

Sometimes, using efficient heuristic algorithms may outperform using the exact algorithms with regards to the computational time. However, there is no guarantee that any optimisation algorithm performs well for any optimisation problem. The *No-Free-Lunch-Theorem* (NFL) proves this fact (Wolpert and Macready, 1997). The NFL theorem explains that each optimisation algorithm is designed for a sub-class of optimisation problems, where

it performs well in practice. However, this may not be the case for other characteristic problems.

In general, according to their application area, the classical heuristic methods are categorised as follow (Zanakis et al., 1989):

- **Construction methods**. The construction algorithms generate a feasible solution. They are obtained by adding individual components (like nodes, arcs) one at a time. The *greedy algorithms* are the most commonly used approaches. They seek maximum improvements at each step. They start from a given feasible or infeasible solution. At each iteration, the greedy algorithms choose the best move to improve that solution (VoB, 2001). Moreover, *look-ahead algorithm* is another approach. At each iteration, it estimates the sequence of possible choices and candidate of solutions. It discards all the choices or candidate solutions which may lead to a bad final solution. In general, most construction algorithms can not reach the feasible solution till the end of the search. One example of construction heuristics is the nearest neighour in travelling salesman problems.

- **Improvement methods**. They are also known as *local search methods*. The improvement methods start from a feasible (initial) solution. They are then improved by exchanges or mergers in the local search until they reach the local optimum. The feasible solution is maintained through the search. In general, for each solution x, they define a *neighbourhood* $\mathcal{N}(x)$ with all candidate solutions. Then the *move* is selected if the new solution is better than the current solution x until the local optimum is found. Sometimes there are combinations between construction and improvement methods. In this case, the construction methods find the initial solution, while the improvement methods improve it in order to find the local optimum.

- **Mathematical programming methods**. In this type of approach, there are a combination of mathematical optimisation models and an exact solution procedure. The solution is then modified to obtain an efficient heuristic to solve the problem. However, this approach is not as clear-cut as the other approaches. This design is a creative

process and gives more opportunities for developing, for instance, using the estimation procedures by incomplete branch-and-bound.

- **Decomposition methods**. This approach attempts to solve the problems by dividing them into a sequence of manageable smaller problems. The output of one will be the input to the next one, then inductively merging these solutions. The final solution of the problem is decomposed into a number of discrete steps, where in most cases, it is a one pass procedure.

- **Partitioning methods**. This approach is similar to the decomposition method. However, it divides the problem into subproblems. Each subproblem is solved independently, and the solution of the problem is given by merging the solutions of subproblems.

- **Relaxation methods**. This approach is the opposite of restriction, as it increases the solution space to obtain a manageable problem. Some methods are multistage. The first stage utilizes a relaxation approach to decompose a problem, where the initial solution is almost infeasible, and the feasible solution is found in the next stage.

As previously mentioned the heuristic methods were introduced in the late 1940s. Each approach was established to solve the specific structure of problems, and as a result the heuristics were called special heuristics. In the last three decades a more general heuristic methodology was introduced. It is called *metaheuristics*. The next section will give more details about metaheuristic, as well as its definition and classification.

0.4 Metaheuristics

It was introduced by Glover. Its name is derived from two Greek words, *heuristics* and the suffix *meta* means "beyond, in the upper level" (Blum and Roli, 2003). A metaheuristic can be applied to a wide structure of problems. In Osman and Laporte (1996), it is defined as "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring the search

space, learning strategies use structure information in order to find efficiently near-optimal solutions".

In general, metaheuristics have some fundamental properties, which give the metaheuristic methods their characteristics:

- Its strategy is to guide the search process.
- Its goal is to explore the solution space to find a better solution (new optimal solution) than the current one.
- The metaheuristic algorithms transform simple local search procedures to complex ones.
- Metaheuristic algorithms are approximate algorithms and most of the time they are non-deterministic.
- Metaheuristic methods are not specific for one problem.

Moreover, a good metaheuristic algorithm should have a balance between *diversification* and *intensification*.

Intensification refers to the term exploitation. Its idea is exploring the promising area from the search space to ensure that the best solutions in this area have been found. It is based on intermediate-term memory such as *recency memory*, where exploitation is a short-term memory.

Diversification refers to the term exploration. Its idea, opposite to intensification, is forcing the search to visit previously unexplored areas of the search space. Sometimes using the intensification term may lead to the loss of some good solutions, while the diversification resolves this problem. Moreover, it is based on a long-term memory of the search such as *frequency memory*.

There are many classifications for metaheuristics. Each one is related to a specific viewpoint. All these classifications are possible. The most important types of classification are given in (Blum and Roli, 2003):

- ***Natural inspired vs. non-natural inspired***. This classification is related to origin of algorithm. For instance genetic algorithms are natural inspired types, whereas Tabu search is a non-natural inspired one. This type of classification is not a meaningful one due to two reasons:
  - Most of the hybrid metaheuristics are not related to both classes.
  - Secondly, sometimes it is really difficult to decide whether the algorithm is related to one of two classes.
- ***Population based vs. single point search***. In this classification, the number of solutions is used to decide if the algorithm is population based or a single point search. If the algorithm is working in a population solution, it is a population based algorithm. Otherwise, it is single point search algorithm, where sometimes it is called a *trajectory algorithm* like Tabu search, Iterated local search and Variable neighbourhood search. The difference between two classes occurs during the search space. The single point search is described as trajectory in the search, where the population based algorithm is describing the evolution as a set of points.
- ***Dynamic vs. static objective function***. The way of using the objective function has been used to differentiate between two types. Some metaheuristics keep the objective function fixed during the search; such approaches are static objective function ones. However, in the guided local search approach the objective function is modified to escape from local minima. Moreover, the objective function is altered during the search. This approach is a Dynamic one.
- ***One vs. various neigbourhood structures***. In general, most metaheuristic approaches have one neighbourhood structure during the search. This means the landscape topology is fixed during the search. Whereas, other metaheuristics have different neighbourhood structures such as variable neighbourhood search, this methodology has allowed the change of the landscape during the search.
- ***Memory usage vs. memory-less method***. This type of approach is very important.

It depends on the usage of memory during the search, which means the search may or may not have a history. There are three known types, short-term, intermediate-term and long-term memory. The first type is focused on the most recent moves and solutions. However, the long-term one is an accumulation of parameters about the search.

Table 1 (Consoli, 2008) summarises the classification of each type of metaheuristic approach, that will be explained in Chapter 1.

Table 1: The metaheuristics classification for some local search

	SA	TS	ILS	GLS	VNS	GA
natural inspried	×	×	×	×	×	√
single solution	√	√	√	√	√	×
population based	×	×	×	×	×	√
dynamic objective function	×	×	×	√	×	×
static objective function	√	√	√	×	√	×
one neibourhood structure	√	√	√	√	×	√
various neibourhood structures	×	×	×	×	√	×
memory usage	×	√	√	√	×	√
less usage	√	×	×	×	√	×

where SA denotes simulated annealing, TS denotes tabu search, ILS denotes iterated local search, GLS denotes guided local search, VNS denotes variable neighbourhood search and GA denotes genetic algorithm.

Moreover, the exact methods can be combined with metaheuristics in two ways (Puchinger and Raidl, 2005):

- **Collaborative combinations**. The algorithm in this case exchanges the information between exact and heuristic algorithms in parallel.

- **Integrative combinations**. In this technique, there is a master algorithm and at least one integrated slave. Furthermore, the master algorithm could be an exact or a heuristic one.

0.5 Thesis overview

In this thesis, the continuous variable neighbourhood search (Mladenović and Hansen, 1997; Liberti and Drazic, 2005; Mladenović et al., 2008) based metaheuristics and the reformulation descent variable neighbourhood search (Mladenović et al., 2005) for censored quantile regression and circle packing problems respectively are presented. The major part of this thesis is devoted to the development of metaheuristics for solving censored quanitle regression, based on continuous variable neighbourhood search metaheuristic frameworks. Moreover, continuous variable neighbourhood search is applied on the Powell estimator. This function is non convex nor concave in regressor, where it is hard to solve exactly. In this thesis, the Powell estimator has been solved exactly, which has been achieved for the first time. Furthermore, continuous variable neighbourhood search with reformulation descent idea is applied to the circle packing problem with two variant containers (a circle and a square). However, the purpose of this thesis is beyond applying GLOB software, which is designed to solve box constraints continuous problems, on different types of problems. The chapters of this thesis are organised as follows.

Chapter 1 is focussed on the literature review. It gives an overview of the most famous local search based metaheuristic approaches with a single point search (like simulated annealing, tabu search and guided local search).

As this thesis is focused on variable neighbourhood search, Chapter 2 explains in detail this metaheuristic, where a brief overview of each type of variable neighbourhood search approach is provided.

Censored quantile regression models are very useful for the analysis of censored data when standard linear models are felt to be inappropriate. This problem is an econometric one. However, fitting censored quantile regression is hard numerically due to the fact that the

function that has to be minimised (Powell estimator) is not convex nor concave in regressors. In chapter 3, we suggest a different approach, i.e., we directly solve nonlinear non-convex non differentiable optimisation problems. Our method is based on a continuous variable neighborhood search approach, a recent successful technique for solving global optimisation problems. The target here is to minimise the Powell estimator function. The GLOB software (Drazić et al., 2006) is applied on three different cases from the literature (Bilias et al., 2000; Buchinsky and Hahn, 1998). The Nelder-Mead heuristic has been used as a low-level search component. Simulation results presented indicate that our new method can considerably improve the quality of censored quantile regression estimator.

Several years ago circle packing problems (CPP) in the plane have been formulated as nonconvex optimisation problems. Chapter 4 is based on (Rajab et al., October 2011) and it proposes applying the idea of reformulation descent (RD) on circle packing problems. It consists of finding a fixed number n of equal circles within different types of containers: a circle and a square, without overlap. There are two different formulations to solve the problem in the Cartesian system. The first one is maximising the radius r associated with n equal circles when the container size is fixed as a unit circle (square), assuming the container is centered at the origin. The second formulation minimises the circle container R (or the length of edge L for square container) to accommodate n unit circles. The variable neighbourhood search has been applied as a nonlinear global optimisation method to solve the problem. We apply two types of Cartesian formulations, where they switch after half of the time. This idea has been applied to find n equal circles within the circle and the square container. The VNS is applied to solve each formulation independently. The experimental run is from $n = 10$ until $n = 200$. The computer results show that our approach is comparable with some of the very best methods from the literature (Hungarian, 2009).

Finally, in Chapter 5, the results and contribution of the thesis are summarised. Suggestions on possible future innovation and development in the field of metaheuristics are discussed.

Chapter 1

Local search based metaheuristics

The most used classical heuristic methods are *local search* methods and *constructive* methods. Constrictive methods use information from the problem structure to build up a single solution. It adds components to the current solution until the feasible one is reached. The local search method attempts to find a local optimum by starting from a given initial (feasible) solution, and improves it gradually at each iteration. Moreover, the local search methods can be considered as the basic principle underlying a number of optimisation strategies, where they have been used in many applications with good empirical achievement in most cases (Johnson et al., 1988). The interest in the local search approach has increased with the rapid development of methaheuristics, and it has been used as a procedure within some methaheuristic algorithms such as a low level search strategy (component).

The algorithmic aspects of local search and high level metaheuristic methods with some applications are proposed in this chapter. In Section 1.1, the basic idea of the local search framework is discussed. A brief idea on some of the most important local search based metaheuristics are given in Section 1.2. Section 1.3 is focused on future metaheuristics, which are called hybrid metaheuristics and their classification.

1.1 Local search basic idea: iterative improvement

The main strategy of local search algorithms for solving any problem starts from a given initial solution. Then it tries to improve that solution by repeating small changes inside the selected neighbourhood. At each iteration, if the new *neighbouring solution* is better than the current one, the change is kept, otherwise another improvement will be applied until no further improvement in the objective function can be found (Papadimitriou and Steiglitz, 1998). The definition of a *neighbourhood structure* is given in Definition 2

Definition 2 *Let P be a given optimisation problem and S the solution space. A neighbourhood structure for problem P is a function $\mathcal{N} : S \rightarrow P(S)$ that assigns to every $x \in S$ a set of neighbours $\mathcal{N}(x) \subseteq S$. $\mathcal{N}(x)$ is called the neighbourhood of x, where it could be any solution $y \in \mathcal{N}(x)$.*

The definition of neighbourhood structure enables us to explain the concept of *locally optimal* solutions.

Definition 3 *A locally minimal solution (or local minimum) with respect to a neighbourhood structure $\mathcal{N}$ is a solution y such that $\forall\ x \in \mathcal{N}(y) : f(x) \leq f(y)$, where we call y a strict local minimum if $f(x) < f(y), \forall\ x \in \mathcal{N}(y)$.*

Moreover, the local optimum for a maximisation problem is defined in a similar way by adding this condition $\forall\ x \in \mathcal{N}(y) : f(x) \geq f(y)$ instead of $\forall\ x \in \mathcal{N}(y) : f(x) \leq f(y)$, and the optimal solution in this case is called a local maximum solution.

A good neighbourhood structure should satisfy the following conditions:

- For each solution x, the neighbourhood structure should be symmetric, that means $(\forall x \in S)\ y \in \mathcal{N}(x) \Leftrightarrow x \in \mathcal{N}(y)$.
- For any two solutions $x, y \in S$, the sequence of solution $x_1, x_2, \ldots, x_n \in S$ should exist and satisfy the condition $x_1 \in \mathcal{N}(x), x_2 \in \mathcal{N}(x_1), \ldots, x_n \in \mathcal{N}(x_{n-1}), y \in \mathcal{N}(x_n)$.
- Generating neighbours $y \in \mathcal{N}(x)$, for a given solution x, should be of a polynomial complexity.

- The neighbourhood size should be determined very carefully. This means the neighbourhood should not be too large, or where it can not be explored easily. A large neighbourhood leads to expensive computation. On the other hand, it should not be too small so no neighbours with better objective function could be found. The size of the neighbourhood is defined according to the optimisation problem size.

The basic algorithm for the local search method can be written as in Algorithm 4, where P is the optimisation problem and x is an initial solution from the solution local space $x \in S$.

```
Function LS (P, x)
1 P is an optimisation problem and S is a search space
2 Choose an initial solution x
3 Define neighbourhood structures N(x) ⊆ S
4 begin
5     repeat
6         x' ← Improvement_function(P, x, N(x)) // find new solution in N(x)
          x ← x'
      until No improvement
7     return x
  end
```

Algorithm 4: Basic local search

The $Improvement\_function(P, x, \mathcal{N}(x))$ is trying to find a better solution than the current solution x within the same neighbourhood $\mathcal{N}(x)$. There are two possibilities to find x by a *first improvement* heuristic or a *best improvement* heuristic.

If there is a need to completely explore the neighbourhood of $\mathcal{N}(x)$, the best choice is the best improvement heuristic. It returns with the best value of the objective function (minimum or maximum) after it completely explores the neighbourhood of $\mathcal{N}(x)$. In this case, the move is made only if a new neighbour with the lowest objective function (in minimum case) has been found. The local search is known as *steepest descent*. The algorithm of best improvement heuristic is given in Algorithm (5):

Function `Best_Improvement_function`$(P, x, \mathcal{N}(x))$

```
repeat
    x' ← x
    x' ← argmin_{y∈N(x)} f(y)
until f(x) ≥ f(x')
return x'
```

Algorithm 5: Best improvement function

In some cases, using a best improvement heuristic may be time consuming. In practice, using the first improvement heuristic is sometimes a better choice than the best improvement heuristic. The solutions $x_i \in \mathcal{N}(x)$ of the first improvement method are enumerated systematically, then the move is made when a new direction for decent has been found. The algorithm for using a first improvement function is given in Algorithm (6):

Function `First_Improvement_function`$(P, x, \mathcal{N}(x))$

```
repeat
    x' ← x; i ← 0
    repeat
        i ← i + 1
        x' ← argmin{f(x), f(x_i)}, x_i ∈ N(x)
    until (f(x) < f(x_i) or i =| N(x) |)
until f(x) ≥ f(x')
return x'
```

Algorithm 6: First improvement function

If the initial solution is found by using some constructive methods, the best improvement heuristic is slightly better than the first improvement (Hansen and Mladenović, 2006). It may be even faster. But if the initial solution has been found randomly, the better choice is to use the first improvement heuristic (Hansen and Mladenović, 2006).

Moreover, applying the best improvement strategy can guarantee that the search will achieve a local optimum, which may not be the case by using the first improvement one. If the local search heuristic has been engaged as a low level component inside the metaheuristic algorithm, the first improvement will be enough and gives good quality solutions. However, there is another possibility to use both strategies at the same time. A sample of neighbours have been generated (randomly or by using some strategy), then the best neighbour is selected from the observed sample, i.e. not the best in the whole neighbourhood (Battiti et al., 2008). The usage of best improvement vs. first improvement is discussed in details in (Hansen and Mladenović, 2006).

To sum up, the local search heuristic is a good method that can be used to find a local optimum. However, it cannot guarantee the global optimum, because when the local optimum has been found, the search process stops without being able to reach the global optimum. This phenomenon is explained in Figure 1.1 . To solve this problem, a number of metaheuristic frameworks has been developed to escape from local optimum during the search. In the next section, some of the most important metaheuristics are described.

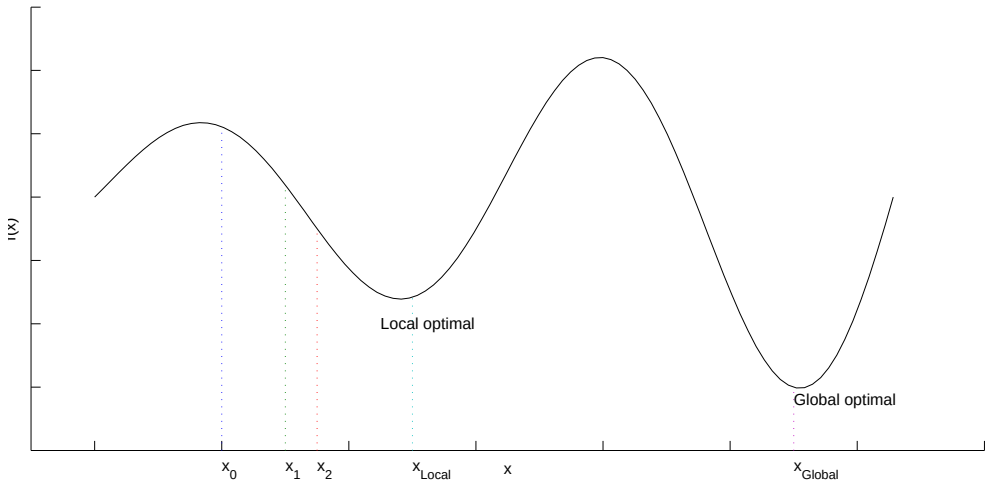


Figure 1.1: Basic idea of local search

1.2 A brief overview of some metaheuristic approaches

In this section some of the most famous metaheuristics are discussed: simulated annealing (in Subsection 1.2.1), tabu search (in Subsection 1.2.2), guided local search (in Subsection 1.2.3) and iterated local search (in Subsection 1.2.4).

1.2.1 Simulated annealing

Simulated annealing (SA) is a metaheuristic algorithm, it is used historically to address the discrete problems, and more recently continuous optimisation ones. It was independently introduced in (Cerny, 1985; Kirkpatrick et al., 1983). The main concept uses a hill-climbing move to escape from local optima in the hope of finding a global optimum, i.e. moves which worsen the objective function value. This technique has made it popular for over the past two decades.

SA is very popular in physics, where its name is derived from the process of annealing with a solid. Crystalline solids are heated and allowed to cool under a controlled cooling technique until the solid is free of crystal defects, i.e. crystal lattice configuration is achieved with its minimum lattice energy state (Nikolaev et al., 2003). Moreover, SA has established the connection between this type of *thermodynamic* behaviour and solving the optimisation problems. The details of the implementation of SA for P optimisation problem are written as Algorithm 7.

Function SA(S, x)

Set S is a search space, and T_k temperature cooling schedule
Choose an initial solution x
Define a neighbourhood structures $\mathcal{N}(x) \subseteq S$
Select an initial temperature $T = T_0 > 0$
Select the temperature change counter $k = 0$
begin
    Set $T \leftarrow T_0$
    while *termination conditions* **do**
        Generate $x' \in \mathcal{N}(x)$
        if $f(x') < f(x)$ **then**
            $x \leftarrow x'$
        else
            Find a random number $\epsilon \in [0, 1]$
            if $\epsilon < \exp(\frac{f(x')-f(x)}{T_k})$ **then**
                $x \leftarrow x'$
        Update T_k
        $k \leftarrow k + 1$
    return x
end

Algorithm 7: Simulated Annealing

As seen in Algorithm 7, the initial temperature T_0 should be defined with the neighbourhood structure $\mathcal{N}(.)$ and the specific cooling structure. Also, a termination condition is included (like maximum CPU time, maximum number of iterations or the maximum number of iterations without improvement).

At each iteration of the SA algorithm, the objective function generates two values. One is the current solution x and the other is a newly selected solution $x' \in \mathcal{N}(x)$. Afterwards, choosing an improved solution is made by a downhill move, where the temperature parameter

is decreased (or non-increased) during the search. Conversely, choosing a non-improved solution (uphill move) depends on the T temperature parameter, where the move in this case is accepted to escape from local minima.

To decide whether the new solution is accepted or not, the *Metropolis criteria* should be included. It is a method of sampling a Boltzmann distribution. It can be simply described as: a move from x_{old} to x_{new} can be accepted if $f(x_{new}) < f(x_{old})$. However, if $f(x_{new}) > f(x_{old})$, the move will be accepted with probability $\exp(\frac{f(x_{new})-f(x_{old})}{T_k})$ (Chu et al., 1999a).

In order to decide if the worse move has been taken or not, the random number ϵ is independently generated by using a uniform distribution in $[0, 1]$. Then, if $\epsilon < \exp(\frac{f(x_{new})-f(x_{old})}{T_k})$, the worse move will be accepted and the temperature will be updated by using the cooling schedule ($T_{k+1} \leftarrow T_k$).

Theoretical results on Markov chains (Aarts and Korst, 1988; Aarts et al., 2005) shows that the SA algorithm can converge to a global minimum when $k \to \infty$, under particular conditions on the cooling schedules. In more details, let P_k be a probability of finding a global minimum after k steps. We can define a $\Gamma \in R$, where $\sum_{k=1}^{\infty} \exp \frac{\Gamma}{T_k} \to \infty$ if and only if $\lim_{k\to\infty} P_k = 1$.

There are different cooling schedules like a *logarithm cooling law* and *geometric cooling law*. The logarithm cooling law can be written as

$$T_{k+1} = \frac{\Gamma}{\lg(k + k_0)} \tag{1.1}$$

where Γ and k_0 are given by the user. It guarantees the convergence of a global minimum. At the same time it is not feasible in the application because it is very slow in practice. Furthermore, the geometric cooling law is faster than the logarithm cooling one, where it can be described as

$$T_{k+1} = \alpha * T_k, \qquad \alpha \in [0, 1] \tag{1.2}$$

where α corresponds to an exponential decay of the temperature. A more robust algorithm can be obtained if the temperature is changed according to a specific iteration L, where $L \in N$ is usually found empirically. t_n is defined as

$$t_n = \alpha^k t_0 \qquad for \;\; kL \leq n < (k + 1)L, k \in N \cup \{0\} \tag{1.3}$$

In Figure 1.2 the idea of using geometric cooling law has been described .

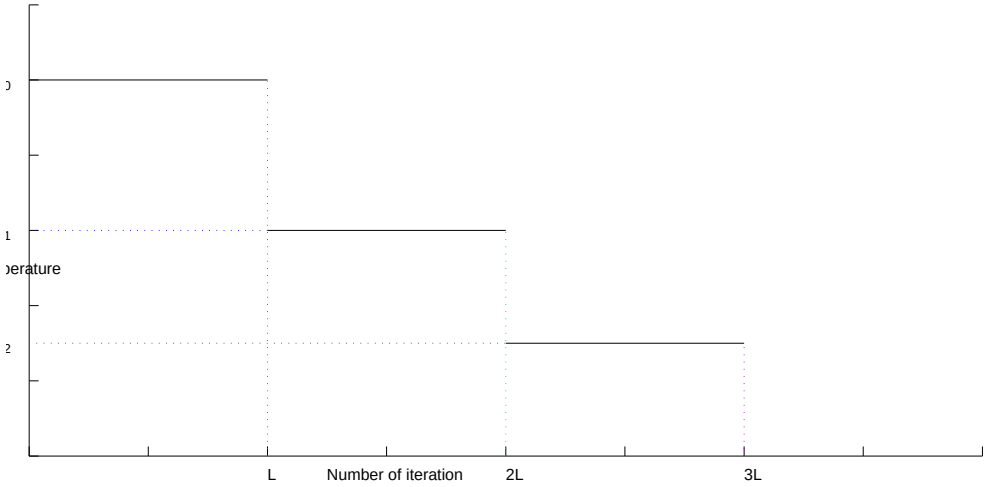


Figure 1.2: Geometric cooling scheme

There are many applications using simulated annealing for solving combinatorial problems, those described by Chu et al. (1999b); Dueck and Scheuer (1990); Kirkpatrick (1984); Osman (1993). In Aarts et al. (1988), they apply the SA algorithm to solve 100-city traveling salesman problems. They use a function of the control parameter of the cooling schedule to analyse the expectation and the variance of the cost. Also, SA is applied to solve 0-1 unconstrained optimisation problems (Chardaire et al., 1995). At a given temperature, they compute the value of the variables. This information helps to reduce the size of the problem where it allows to fix the variables as the temperature decreases.

In Romeijn and Smith (1994), a continuous simulated annealing has been used for solving the maximum of a continuous function, where a hide-and-seek strategy is implemented. This approach is applied when the objective function may be non differentiable and the feasible region may be non convex or disconnected. The difference between this approach and the discrete one is that the candidate point at each iteration of the algorithm may be generated as any point in the feasible region. It will then be either accepted or rejected according to

the metropolis criterion. This algorithm gives a competitive performance by hide-and-seek.

In Corana et al. (1987), simulated annealing is used to solve continuous functions against the Nelder-Mead simplex method and Adaptive Random search. The algorithm has adapted moves according to an iterative random search. In this case, the SA algorithm gives more reliable results than the others.

Also SA has been applied for econometrics problems. In Goffe et al. (1994), the SA algorithm, in contrast to the other three common conventional algorithms it is compared with , is less likely to fail in difficult functions. Also it can find the global optimum for four different econometrics problems. For more applications in global optimisation see Dekkers and Aarts (1991); Vanderbilt and Louie (1984). Nowadays, SA is used as a component in metaheuristics rather than applied as a basic algorithm for the search.

1.2.2 Tabu search

Over the last fourteen years, tabu search (TS) has been one of the most used metaheuristics for solving optimisation problems. It was first introduced by Glover (1986). This approach escapes from local optima by a strategy of forbidding certain moves to prevent cycling. Usually, this method gives solutions very close to optimal ones. It is among the most effective on difficult problems, and have therefore made TS very popular.

As opposed to branch and bound, TS might be called a "weak inhibition" search. Tabu generally holds a small fraction of moves, according to what is still available. These moves then become accessible after a short time. Moreover, TS keeps the ability to guide the search to escape from poor local optima, in similar to simulated annealing, by using a deterministic nature rather than a stochastic one.

As mentioned above, *tabus* are used to prevent cycling, moving from local optima and not going back. These tabus are stored in a *tabu list* (a short-term memory). It is used to avoid revisiting the most recent solutions, and forbidding any movement toward them. Tabu lists are not only used to prevent a move from being repeated, but also they prevent moves from being reversed.

First In First out (FIFO) is a technique for updating the tabu list, i.e. when the current

solution is added to a tabu list, the oldest one in the list is removed. *Tabu tenure* is the length of tabu list, where it is used to control the memory of the search process. In most cases, the length of the tabu tenure is fixed. Moreover, the small fixed length of tabu tenure cannot always prevent cycling, where the search will concentrate on a small area of the search space. In contrast, a large tabu tenure explores larger areas. For solving this problem, some methods have used varying tabu tenure during the search (see e.g. Glover (1989a,b)). On the other hand, another technique has been used, where the procedure for generating a random tabu tenure for each move has been added in a specific interval (see e.g. Gendreau et al. (1994)).

Sometimes tabus are too powerful, they may lead to the loss of some unvisited good quality solutions. For that, the *aspiration criteria* has been added. In general, the aspiration criteria is an algorithmic device. It allowes a move, even if it is tabu, if it gives a solution with a better objective value than the current best known one. The TS is described in Algorithm 8.

Function TS (P, S, x)

Set P is an optimisation problem and S is a search space
Choose an initial solution x
Define a neighbourhood structures $\mathcal{N}(x) \subseteq S$
Memorise the best solution so far x'
Define tabu list TL
Define allowed set AL
begin
    $TL \longleftarrow \emptyset$
    Move $x' \leftarrow x$, where $x' \in \mathcal{N}(x)$
    Update TL using FIFO $(TL \cup x)$
    while *termination conditions* **do**
        $AL \leftarrow \mathcal{N}(x) - TL$
        Find the best solution within AL: $x \leftarrow Update(AL)$
        if $f(x) < f(x')$ **then**
            $x' \leftarrow x$
        Update TL
    return x'
end

Algorithm 8: Tabu Search

The $Update(.)$ function tries to find a better solution from the set of solutions that belongs to the allowed list AL. There are two possible functions: first improvement function or best improvement function (as explained in Section 1.1). By using first improvement strategy, the $Update(.)$ function scans the AL and finds the first solution that is better than the current one. However, by using the best improvement one, the $Update(.)$ function completely discovers the whole allowed set and returns the solution which gives the minimum objective function value. Including the $Update(.)$ function in the algorithm makes TS more efficient to explore solutions in a dynamic neighbourhood structure with short term memory

implemented by TS. The termination conditions could be:

- A fixed number of iterations;
- A fixed amount of CPU time;
- After a fixed number of iterations without an improvement in the objective function;
- If the objective function reaches a pre-specified threshold value.

A simple TS may successfully solve difficult problems. For the most of the cases, TS should include additional elements to make the search strategy fully effective. *Intensification* is one of them, where the search should explore more portions of the search space, that could be promising areas, to make sure that the best solution is indeed found. That means the normal search should stop from time to time to perform an intensification phase. Generally intensification is based on some *intermediate-term memory*, like a *recency memory*. It records the number of consecutive iterations that various "solution elements" have been introduced in the current solution without interruption.

Many TS implementations have used an intensification strategy. However, in some cases using the normal search is enough, and there is no need to spend time in exploring more portions of the search space that have been already visited. Due to that the *diversification* strategy should be included. As opposed to intensification, diversification tends to force the search to go through previously unexplored portions of the search space. It is based on some *long-term memory*, like a *frequency memory*. It records the total number of iterations (since the search start) that various solution elements have been involved in the current solution.

There are different types of diversification, *restart diversification* and *continuous diversification*. The restart diversification tries to force a few rarely used elements in the current solution and start the search again from this point. The continuous diversification adds diversification considerations directly into the regular search process.

In mateheuristics, there are four terms to describe the usage of memory: recency, frequency, quality and influence. The first two are the most important, and have been discussed earlier. In general, quality refers to the solutions with good objective function values. That

may help in TS to give the intensive search in the most promising regions. Influence measures the degree of change in solution structure. In TS, it is an important aspect of aspiration criteria.

There are some problems where the true objective functions are quite costly to evaluate. To solve this problem, TS has used a *surrogate objective function*. It is a less demanding computational function. Besides, it is correlated to the true objective function by the identify of a small set of promising candidates. The true objective function is then computed (see Crainic et al. (1993)).

TS applications. The traditional concept of TS has been applied to combinatorial problems. Nowadays TS deals with different techniques for making the search more efficient. TS includes methods for giving more information during the search about better starting points, parallel search strategies and more powerful neigbourhood operators, for application (see Crainic et al. (1997)). Moreover, *hybridization* is an important trend in TS, and it is used in TS with other heuristics approaches such as Lagrangean relaxation (Grunert, 2002), column generation (Crainic et al., 2000) , Ant colony optimisation (Arito and Leguizamón, 2009) and Genetics Algorthims (Crainic and Gendreau, 1999; Fleurent and Ferland, 1996).

TS is also adapted with other metaheuristics approaches for solving the global optimisation problems. In Teh and Rangaiah (2003), a new version from TS has been applied, it is namely an enhanced continuous TS (ECTS). ECTS has performed better for many problems including high dimensional ones. Furthermore, ECTS has two steps. First it attempts to apply a benchmark test function having multiple minima, and then it evaluates for phase equilibrium calculations. ECTS algorithm have four stages: parameters setting, diversification, identifying the most promising area and intensification (Teh and Rangaiah, 2003). Also there are a similarity between TS and genetic algorithm (GA) in locating the global minimum, where TS converges faster than GA. For another ECTS application see Chelouah and Siarry (2000).

Another approach for TS in global optimisation is presented by Battiti and Tecchiolli (1996). They introduce a novel algorithm (C-RTS), in which reactive TS cooperates with a stochastic local minimiser. It is used for unconstrained global optimisation, where only the

function values are required. C-RTS uses an efficient memory during the search. Besides, it has a mechanism to tune the search space to be discretization by having a tree of search boxes.

1.2.3 Guided local search

Guided local search (GLS) is metaheuristic based on penalty, where it sits on the top of other local search methods. It was introduced by Voudouris and Tsang (April 1996). In GLS, a new strategy has been used by augmenting the objective function to escape from the current local optimum. In contrast, the other metaheuristics strategies use a fixed objective function, while the set of solutions and the neighbourhood structure are changed during the search (i.e. changing the search landscape to escape from the local minima). The procedure of GLS is illustrated in Figure 1.3.

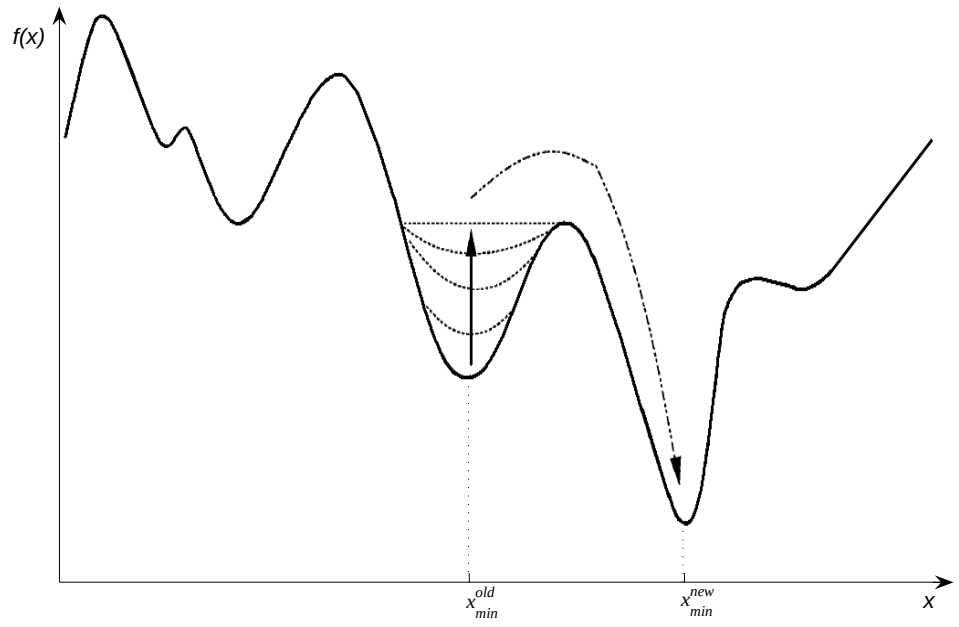


Figure 1.3: Guided Local Search

The main concept of the GLS algorithm is based on the definition of *solution feature* i. It has been used to discriminate between solutions. This solution supposes to have the

property of non-trivial solution, where not all the solutions have this property. Moreover, the constraints on feature come from the problem information and the local search heuristic, where the features cost could be constants or variables.

$I_i(x)$ is the indicator function to define if the feature i is chosen or not in the specific solution x, where it can be written as

$$I_i(x) = \begin{cases} 1 & if\ solution \text{ x } has\ property \text{ i} \\ 0 & otherwise \end{cases}$$

The *augmented cost function* is used to augment the objective function and includes the penalty p. At each iteration of the GLS algorithm, the local search modifies a penalty vector $p = (p_1, p_2, \ldots, p_n)$ to escape from a local minimum. If we suppose that we have n features and $f(.)$ the objective function (cost function), then the augmented cost function can be written as

$$f'(x) = f(x) + \lambda . \sum_{i=1}^{n} p_i . I_i(x) \tag{1.4}$$

where λ is the *regularization parameter*. The importance of the regularisation parameter comes from representing the relative effect of penalties with respect to the solution cost. At the beginning, all the penalty parameters are set to 0 (i.e. no features are constrained). Then a call is made to the local search to find a local minimum of the augmented cost function. The GLS algorithm is described in Algorithm 9.

Function GLS (S, n, λ, x)

Set the search space S, n solution features, and the regulation parameter λ

Choose an initial solution x

Define a neighbourhood structures $\mathcal{N}(x) \subseteq S$

Let p_i, where $i = 1, \ldots, n$, be the penalty parameters for the n solution features

Let c_i, where $i = 1, \ldots, n$, be the costs assigned to the n solution features

Let I_i, where $i = 1, \ldots, n$, be the indicator function to the n solution features

begin

  Generate an initial solution x

  Initialize the penalty parameters to 0, i.e. $p_i \leftarrow 0$ for all $i = 1, \ldots, n$

  while *termination conditions* **do**

    Find $f'(x) = f(x) + \lambda . \sum_{i=1}^{n} p_i . I_i(x)$, where

$$I_i(x) = \begin{cases} 1 & if \ solution \text{ x } has \ property \text{ i} \\ 0 & otherwise \end{cases}$$

    Apply a local search for $(f'(x), \mathcal{N}(x))$ to find a new solution x'

    if $f'(x) < f'(x')$ **then**

      $x' \leftarrow x$

    Compute the utility function $Util(x, i)$ for each feature i, where $i = 1, \ldots, n$, of the current candidate solution x, where the utility function is explained as

$$Util(x, i) = \begin{cases} I_i(x) . \frac{c_i}{1+p_i} & if \ solution \text{ x } has \ property \text{ i} \\ 0 & otherwise \end{cases}$$

    for *each solution feature i with maximum* $Util(x, i)$ **do**

      Penalize the solution feature i: $p_i \leftarrow p_i + 1$

  return x'

end

Algorithm 9: Guided Local Search

$c = (c_1, c_2, \ldots, c_n)$ is a vector of cost. That means for each feature i, there is a cost c_i. The cost vector may be a constant or a variable which contains zero or positive elements. The cost vector with the local minimum gives the sources of information about the problem. Moreover, a practical local minimum contains a number of features. That means if the feature f_i is exhibited in the local minimum of x, the indicator function for feature i is $I_i(x) = 1$.

Also, at a local minimum x for example, the utility of the penalising function for each feature i can be written

$$Util(x,i) = \begin{cases} I_i(x).\frac{c_i}{1+p_i} & if\ solution\ \text{x}\ has\ a\ property\ \text{i} \\ 0 & otherwise \end{cases}$$

in other words, the utility of penalising will be equal to $I_i(x).\frac{c_i}{1+p_i}$, if the feature i is exhibited in the local minimum x, otherwise it will be equal to zero. In addition, if the cost of the feature is lower (the smaller c_i), the utility of penalising will be smaller. Furthermore, if the p_i is greater, the more times it will be needed to penalise, that means the utility of penalising will be lower (Voudouris and Tsang, 2003).

There is a close relationship between GLS and Tabu Search (Voudouris and Tsang, 2003). Tabus in TS can be seen as penalties in GLS, and both ways are used to escape from local minima. Also, TS can be adopted by GLS. For instance, the idea of a tabu list and aspiration criteria have been included in later versions of GLS. However, in GLS if many penalties have been added to augment the objective function, the local search could be misguided. In Voudouris and Tsang (1998), they apply GLS to the quadratic assignment problem, where they use a limited number of penalties (resembling tabu lists), which means when the list is full, the old penalties will be overwritten.

In addition, the GLS adopts the genetic algorithm (GA) to produce a guided genetic algorithm (GGA) (Mills et al., 2003), where GGA is a hybrid of GA and GLS. In GGA, after a specific number of iterations (where this number is the parameter of GGA) without any improvement, the GLS will modify the fitness function by means of penalties, that will help GGA to focus in its search.

There are many applications for GLS. In Kilby et al. (1999); Zhong and Cole (2005),

GLS is applied to vehicle routing problems, while in Zhong and Cole (2005), it is applied to vehicle routing problem with backhauls and time windows. The main idea is to construct an initial infeasible solution and then use GLS to improve that solution to be a feasible one. This new approach gives some better solutions compared with the other previously used methods. Moreover, it is used when the customers are in clusters or distributed normally.

In Mills and Tsang (2000), they apply the GLS algorithm to solve the SAT problem. The new resulting algorithm can be easily extended to solve the weighted MAX-SAT problem. GLS is applied to solve traveling salesman problem in Voudouris and Tsang (1999), where they use the techniques of guided local search and fast local search (FLS). The FLS is applied to neighbourhood to speed up the algorithm. More GLS applications such as three dimensional bin packing problems, capacitated arc-routing and team orienteering problems are discussed by Faroe et al. (2003), Beullens et al. (2003) and Vansteenwegen et al. (2009) respectively.

1.2.4 Iterated local search -Fixed neighbourhood search

Iterated local search (ILS) is a simple and general metaheursitic. It iteratively builds a sequence of solutions generated by an embedded heuristic, which will lead to far better solutions if random trials of that heuristic have been used to find the solutions (Lourenco et al., 2003). ILS algorithm is given in Algorithm 10.

Function ILS $(x, \mathcal{N}(.), \mathcal{S})$

1 Define the neighbourhood structure $\mathcal{N}(x)$

2 **begin**

3 Apply *Generate_initial_solution* procedure to find the initial solution x, $x \leftarrow$ *Generate_initial_solution*

4 Apply *Local_Search* procedure on x to find a better solution x', $x' \leftarrow$ *Local_search*$(x, \mathcal{N}(x))$

5 **repeat**

6 Use the *Perturbation* procedure on x' to find x'', $x'' \leftarrow$ *Perturbation*$(x', history)$

7 Again apply *Local_Search* procedure on x'' to find x'^*, $x'^* \leftarrow$ *Local_search*(x'')

8 Use *Accepting_criterion* procedure to accept x^* or not, $x^* \leftarrow$ *Accepting_criterion*$(x^*, x'^*, history)$

9 Set $x \leftarrow x^*$

until *termination condition met*

end

return x

Algorithm 10: Iterated Local Search

As previously stated, the ILS algorithm is made up of four procedures:

- Generate initial solution procedure.
- Perturbation procedure.
- Accepting criterion procedure.
- Local search procedure.

As we can note the difference between ILS and multi-start method is that the multi-start method re-starts the search from a new solution to achieve the diversification, where ILS has at the beginning an initial solution only.

As shown in Algorithm 10, the ILS algorithm begins by finding the initial solution x, where starting with a good solution gives high quality solutions in reasonable time. There are two possibilities to generate the initial solution, a random restart or a greedy construction heuristic. Using a greedy initial solution over a random restart gives the search two advantages. First combining the initial solution with local search leads to good quality solutions. Additionally, local search using greedy solutions needs less CPU time, because it takes less improvement steps (i.e. the greedy solution speeds the search). Furthermore, using random restart with the short computation time will give a solution x' less efficient than a greedy heuristic one.

In order to avoid stalling in local optima and reach the global minimum, ILS uses the perturbation procedure as in the SA algorithm. The local search should not be able to undo the perturbation. Otherwise the search will fall back to visited local optima. The local search can achieve the perturbation procedure by using random moves on the neighbourhood higher than the one used before. It is still a good idea to use the perturbation procedure, which guarantees better results.

Changing the current solution using perturbation should not be too strong, because it will lead it to behave as a random restart. Also, finding better solutions are not a guarantee. On the other hand, the perturbation should not be too small, where the search may revisit the same local optima. Moreover, the *perturbation strength* may be refereed to as the number of solution components, which means an appropriate perturbation strength depends on the instance size. For example in traveling salesman problem (TSP), it is the number of edges that are changing during the tour, roughly, the strength is defined as the amount of change made on the current solution, where it may be fixed or variable.

There are many ways to determine the perturbation strength. For instance, in TSP problems it is very small and seems independent of the instance size. On the other hand, it is driven to a large size in the quadratic assignment problem (QAP). Furthermore, ILS for the QAP shows that there is not a priori single best size for the perturbation, according to that ILS algorithm adapts the perturbation strength during the search, more information is described in Hong et al. (1997).

After finding another solution x'' by using the perturbation procedure, the acceptance criterion is used to decide if the move will be taken or not. It controls the balance between the intensification and diversification during the search. A Markovian acceptance criterion (or better acceptance criterion) for minimisation problems is a very strong intensification. It is simply achieved by accepting better solutions. It can be written as

$$Better\_acceptance\_criterion = \begin{cases} x'^* & if\ f(x'^*) < f(x^*) \\ x^* & otherwise \end{cases}$$

On the another hand, the random acceptance criterion, which favours diversification, can be used for applying the perturbation to the visited local optima, irrespective of its cost. It can be described as

$$random\_acceptance\_criterion(x^*, x'^*, history) = x'^* \tag{1.5}$$

There are many intermediate choices between the better acceptance and the random acceptance criterion. For instance in Martin et al. (1991, 1992) the large step Markov chains algorithm has been applied with a simulated annealing type acceptance criterion. It is denoted as LSMC and it is given in Algorithm 11.

Procedure `LSMC` (x^*, x'^*, history)

1 **if** $f(x'^*) < f(x^*)$ **then**
  $x'^* \leftarrow x^*$
else
  accept $x'^* \leftarrow x^*$ with probability, $\exp\{\frac{f(x^*)-f(x'^*)}{T}\}$

Algorithm 11: LSMC acceptance criterion

where T is called a temperature parameter as in SA. The LSMC behaves as a better acceptance criterion when the temperature is very low, and as random acceptance criterion when the temperature is very high.

There is a limited case for using the memory with the acceptance criteria. It has been used to restart the algorithm when the intensification becomes inefficient to switch to diversification. This idea could be applied when no improved solution have been found. It uses the new initial solution for a given number of iterations to restart the algorithm. The restart

acceptance criterion is given in Algorithm 12.

Procedure `Restart` (x^*, x'^*, history)

1 **if** $f(x'^*) < f(x^*)$ **then**
  x'^*

else if $f(x'^*) \geq f(x^*)$ *and* $i - i_{last} > i_r$ **then**
  x

else
  x^*

Algorithm 12: Restart acceptance criterion

where i_{last} is the last iteration where a better solution has been found, and i is the iteration counter. Also, i_r indicates the number of iterations without any improvement.

The ILS algorithm is very sensitive to the choice of embedded heuristics. There are many different algorithms which may fit an embedded heuristic. In general, the better the choice of the local search, the better is the corresponding ILS. For example, when the CPU time is fixed, it is better to choose a less efficient and fast local search than a slower and more powerful one. The best choice depends on how much time is needed to find a better solution. However, if the speed does not make any difference, then the better heuristic is worth applying. Besides, the local search can not easily undo the perturbation. Consequently, good ILS depends on the combination of all four components. The best choice of perturbation depends on the local search, while the best choice of acceptance criterion depends on perturbation and local search. Briefly, the search space has to have these two points:

- The perturbation should not easily become undone by the local search. Moreover, the perturbation should compensate for the local search, if the local search has short comings.

- Having a good combination of perturbation and acceptance criterion. The relation between these two makes the balance between intensification and diversification.

More applications and some interesting developments of the ILS algorithm can be found in Hong et al. (1997); Martin et al. (1991); Stutzle (2006); Tang and Wang (2006).

1.3 Future in metaheuristics area

Hybridization is a recent trend in metaheuristics, which can be defined as the integration between the single-solution methods with the population-based methods. In general, we can distinguish three different types (forms) of hybrid metaheuristics: component exchange among metaheuristics, cooperative search and integrating metaheuristics (Blum and Roli, 2003).

Component exchange among metaheuristics is one of the most popular and uses hybridization by combining the single-methods in population-based methods. The reason of the power of this combination becomes apparent by explaining the strength of two types: population-based methods and trajectory methods (single-solution methods).

The main idea of population-based methods is based on recombining the solutions to obtain new ones. The explicit recombining solutions are implemented by one or more recombination operators in evolutionary computation and scatter search. However, in Ant Colony Optimisation and Estimation of Distribution Algorithms, the recombining solutions are implicit according to the usage, where the distribution over the search space will generate new recombining solutions. This recombination procedure in population-based methods allows "big" guided steps in the search space. Usually these guided steps are larger than the steps performed by trajectory methods. However, some trajectory methods like iterated local search and variable neighbourhood search have big steps as well, because their steps are usually not guided. They perform from random mechanisms, which are called "kick moves" or "perturbation". Moreover, the strength of trajectory methods drives the search to explore the promising areas in the search space. In conclusion, we can note that the population-based methods are more powerful in finding the promising areas in the search space, whereas trajectory methods are superior in searching specific zones (promising areas) of the domain. That leads to very successful applications by using this form of hybrid metaheuristics.

The second form of hybrid metaheuristics is *cooperative search*. Its basic idea is exchanging information between different algorithms, where the algorithms could be approximate or complete or a mix of both types. This exchange might consist of exchanging in states,

models, entire sub problems, solutions or other search space characteristics. Typically, this type consists of the parallel execution of search algorithms with different communication levels. The used algorithms could be different or the same with different models or parameters. Nowadays, cooperative search receives much attention due to increasing interest in parallelization of metaheuristics. For more information of parallel metaheuristics see the survey in Crainic and Toulouse (2003).

The last form is the *integration of approximate and systematic (or complete) methods*. This type of hybrid metaheuristics is really powerful and gives very effective algorithm when it is applied to real world problems. There are three main approaches for integration of metaheuristics (especially trajectory "single solution" methods) and systematic techniques. Theses are as follows

- When the metaheuristics and systematic method are applied in sequence. If the metaheuristics algorithm is working to find the solutions, then these solutions will be the heuristic information that will be improved by the systematic search and vice versa. This approach can be seen as a cooperative search, or a kind of loose integration.

- When the metaheuristics are applied a complete methods to efficiently explore the neighbourhood structure rather than using random sampling or simply enumerating all the neighbours. This type of search combines two advantages, the fast exploration by using metaheuristics, and the efficient exploration of the neighbourhood by systematic method. This approach is really efficient when the large neighbourhood structures are used or when it is applied to real world problems. This type of problems has additional constraints, they are called *side constraints*, where it might be difficult to explore the neighbourhood by using metaheuristics.

- When the concepts or strategies for classes of algorithms are used together. Generally, this means this type of hybridization is achieved by integrating strategies from metaheuristics into tree search methods. For example, the idea of tabu list or aspiration criteria, which are defined in Tabu search, is applied on other algorithms not only on tabu search.

To sum up, the hybrid metaheuristics gives a developed type of metaheuristics as compared to their parent, where they can be applied to more problems and give better results.

Chapter 2

Variable neighbourhood search metaheuristics

All local search based metaheuristics discussed in Chapter 1 are dealing with a single neighbourhood structure at each iteration, which may or may not be updated from one iteration to another. However, this is not the case with varaible neighbourhood search, where more than one neighbourhood structure is included at each iteration. That means the solution process could be significantly improved if more than one neighbourhood of the currently observed solution is explored and thus a few new candidate solutions are generated at each iteration. This is the basic idea of the variable neighbourhood search metaheuristic.

The variable neighbourhood search is thoroughly explained in this chapter, where this variable neighbourhood search metaheuristic is the main idea in this thesis as the research reported.

2.1 Variable neighbourhood search

Variable neighbourhood search (VNS) is a metaheuristic or framework for building heuristics. It was introduced by Mladenović and Hansen (1997). VNS is based upon systematic changes of neighbourhoods in order to find better solutions in distant parts of a solution space.

Most local search metaheuristics use just few neighbourhoods (one or two, number of

neighbourhood ≤ 2) at each iteration, which could be changed from one iteration to another. Changing the neighbourhood structure during the search makes the search process more effective. Therefore, if there is more than one neighbourhood at each observed solution, that will help to improve the solution process to explore the search space and thus find new candidate solutions, fulfilling the basic idea of VNS. There are three obvious facts that could explain why change of neighbourhoods works well:

Fact 1. A global minimum is a local minimum with respect to all possible neighbourhood structures.

Fact 2. A local minimum with respect to one neighbourhood structure is not necessarily a local minimum with respect to another neighbourhood structure.

Fact 3. For many problems local minima with respect to one or several neighbourhoods are relatively close to each other (Hansen et al., 2008).

The last fact is an empirical one, which implies that a local optimum gives information about the global optimum. For instance, it may appear that some variables have the same values in both local and global optima. Moreover, those simple facts are used within VNS in several different ways (see for example recent surveys of VNS in Hansen et al. (2008, 2010)). Furthermore, these three facts can be combined in three different ways: the deterministic one, stochastic one and both deterministic and stochastic, to make a balance between intensification and diversification.

VNS has been used in different applications, for each case it has a selection of neighbourhood structures. Neighbourhood changes scheme or the way of selecting the solutions within a neighbourhood, etc., depends on the problem. In the next subsection, basic schemes of VNS will be explained in more detail. Also, many types of VNS will be described in the following subsections.

2.1.1 VNS basic schemes

VNS is designed for solving both continuous and discrete optimisation problems, that may be formulated as

$$\min\{f(x)|\ x \in X, X \subseteq \mathcal{S}\}. \tag{2.1}$$

$\mathcal{S}, X, x$ and f respectively denote the *solution space*, *feasible set*, a *feasible solution* and a real-valued *objective function*. If $\mathcal{S}$ is a finite but large set, a *combinatorial optimisation* problem is defined. If $\mathcal{S} = \mathbb{R}^n$, we refer to *continuous optimisation*. An *exact algorithm* for problem (2.1), if one exists, finds an optimal solution x^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $X = \emptyset$.

Let $\mathcal{N}_k,\ k = 1, \ldots, k_{max}$, denotes a finite set of pre-selected neighbourhood structures and let $\mathcal{N}_k(x)$ be the set of solutions in the k^{th} neighbourhood of x. Moreover, the neighbourhoods for the same solution are nested i.e. $\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq \ldots \subseteq \mathcal{N}_{k_{max}}(x)$, that means, as opposed to other metaheuristics, VNS is dealing with more than one neighbourhood for each candidate solution x. This phenomenon is explained in Figure 2.1.

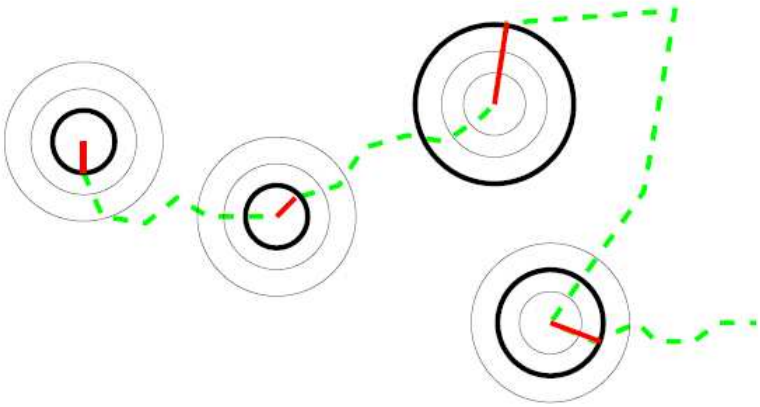


Figure 2.1: The change of neighbourhoods during the VNS search

The neighbourhood structures $\mathcal{N}_k$ may be induced from one or more metrics, where the metric function is $\rho : \mathcal{S}^2 \rightarrow R$, thus the formula for finding $\mathcal{N}_k(x)$ can be described

$$\mathcal{N}_k(x) = \{y \in X \mid r_{k-1} < \rho_k(x, y) \leq r_k\}, \tag{2.2}$$

Or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_k(x, y) \leq r_k\}, \tag{2.3}$$

Where the metric $\rho_k(x, y)$ is monotonically increasing with r_k, where r_k is a given raduis of neighbourhood $\mathcal{N}_k$. The $\rho_k(x, y)$ between any two solutions x and y is given as

$$\rho_k(x, y) = (\sum_{i=1}^{n} \mid x_i - y_i \mid^p)^{\frac{1}{p}} \quad (1 \leq p < \infty) \tag{2.4}$$

or

$$\rho_k(x, y) = \max_{0 \leq i \leq n} \mid x_i - y_i \mid, \quad p = \infty \tag{2.5}$$

We define $x' \in X$ as a local minimum w.r.t. $\mathcal{N}_k$, if there is no solution $x \in \mathcal{N}_k(x') \subseteq X$ such that $f(x) \leq f(x')$. This is a brief idea about VNS scheme and the next subsections will focus on VNS variants.

2.1.2 Variable neighbourhood descent

Variable neighbourhood descent (VND) is performed in a deterministic way to make changes of neighbourhoods. It completely explores the neighouborhood (Hansen and Mladenović, 1999). Due to that, VND requires a large amount of computational effort, where the diversification process is rather slow, whereas intensification is enforced. VND steps are explained in Algorithm 13.

Function VND(x, $\mathcal{N}_k$, $\mathcal{S}$)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)

2 Find an initial point $x \in \mathcal{S}$ (or apply the rules to find it)

3 **repeat**

4 Set $k \leftarrow 1$

5 **repeat**

6 Find the best improvement $x' \in \mathcal{N}_k(x)$ after completely exploring the neigbourhood $\mathcal{N}_k(x)$

7 **if** $f(x') \leq f(x)$ **then**
Set $x \leftarrow x'$, and $k \leftarrow 1$
else
$k \leftarrow k + 1$// Next neighbourhood

until $k \leftarrow k_{max}$

until *no improvement is obtained*

8 **return** x

Algorithm 13: Variable neighbourhood descent

As a first step in Algorithm 13, an initial solution x has been selected from the current neighbourhood. Then for each iteration k, all possible candidate solutions have been generated to find the best neighbour of x. This means the current neighbourhood is completely discovered before moving to another one (step 6). This is the case if there is no other stopping condition. This sequential order of neighbourhood structures can develop a nested strategy.

Using the intensification rather than the diversification gives more chance to reach the global minimum. On the other hand, completely exploring the neighbourhood requires more computational time, which makes the search expensive. Sometimes, VND is used as a local search in other metahuristic frameworks according to its robustness. More applications are discussed (Gao et al., 2008; Hertz and Mittaz, 2001; Ognjanović et al., 2005).

2.1.3 Reduced variable neighbourhood search

Reduced variable neighbourhood search (RVNS) chooses the new candidate solutions randomly from the current neighourhood. This means it uses stochastic search (Hansen et al., 2008). Moreover, RVNS does not apply any local search to improve these candidate solutions. This strategy makes RVNS very useful in very large instances, when using a local search may be costly. More applications are given in Hansen and Mladenović (1999); Mladenović et al. (2003); Remde et al. (2007); Sevkli and Sevilgenr (2008). RVNS is illustrated in Algorithm 14 (Hansen et al., 2008).

Function RVNS(x, $k_{\max}$, $t_{\max}$)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)
2 Choose the stopping condition
3 **while** *termination conditions* **do**
4 **repeat**
5 Set $k \leftarrow 1$
6 **repeat**
7 select at random x', where $x' \in \mathcal{N}_k(x)$ //Shaking
8 **if** $f(x') \leq f(x)$ **then**
Set $x \leftarrow x'$, and $k \leftarrow 1$
else
$k \leftarrow k + 1$// Next neighbourhood
until $k \leftarrow k_{max}$
until $t > t_{\max}$
9 $t \leftarrow CpuTime()$
10 return x

Algorithm 14: Reduce Variable Neighbourhood Search

The algorithm initializes the search by selecting the maximum time (CPU time) and maximum number of neighbourhood structures. Sometimes, another termination condition

may be added like the number of iteration without any improvement in the objective function. It selects then new points at random inside the neighbourhood $\mathcal{N}_k(x)$, which compares with the incumbent one. The update happens, when the improvement is found (see steps 7 and 8), and this process is iterated until no improvement is reached.

2.1.4 Basic variable neighbourhood search

Basic variable neighbourhood search (BVNS) is applied to problems by combining the deterministic and stochastic way in changing of the neighbourhood (Mladenović and Hansen, 1997). This leads to a balance between the intensification and diversification.

However, VND completely explores the neighbourhood, which means that a large amount of computational effort will be required, whereas RVNS just chooses the candidate solution at random. This means the RVNS technique discards the quality of solutions. In BVNS the next candidate solution from the current neighbourhood can be found by selecting a random element (first solution) from the same neighbourhood. Then a local search approach applies to improve it. Thus the best one is chosen to be considered as the next candidate solution for the same neighbourhood. In Figure 2.2, we can note that the BVNS is not exploring all the neighbourhood, but it provides a reasonable-quality solution. BVNS steps are explained in Algorithm 15 (Mladenović and Hansen, 1997).

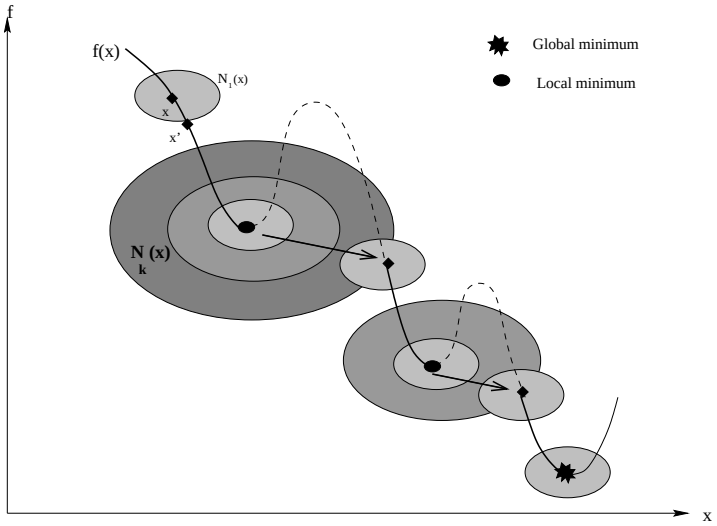


Figure 2.2: The basic variable neighbourhood search scheme

Function BVNS(x, $\mathcal{N}_k$, $\mathcal{S}$)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)
2 Find an initial point $x \in \mathcal{S}$
3 Choose the stopping condition
4 **while** *termination conditions* **do**
5 Set $k \leftarrow 1$
6 **repeat**
7 Generate at random x', where $x' \in \mathcal{N}_k(x)$ //Shaking
8 Apply local search with x' as an initial solution to obtain the local optimum x''
9 **if** $f(x'') \leq f(x)$ **then**
Set $x \leftarrow x''$, and $k \leftarrow 1$
else
$k \leftarrow k + 1$// Next neighbourhood
until $k \leftarrow k_{max}$
10 **return** x

Algorithm 15: Basic Variable Neighbourhood Search

The new point x' is generated at random (i.e. in stochastic rule in step 7 in Algorithm 15) to avoid cycling, which might occur if the deterministic way is applied. After that, a local search method is applied on x' as an initial solution to find x'' (step 8 in Algorithm 15). Sometimes BVNS may be replaced by the local search by using variable neighbourhood descent, where this combination leads to the most successful applications (see Hansen and Mladenović (2001a)).

2.1.5 General variable neighbourhood search

General variable neighbourhood search (GVNS) is derived from the basic variable neighbourhood search, when BVNS is used as a local search to find the improvement. GVNS has

led to the most successful applications (see Andreatta and Ribeiro (2002); Brimberg et al. (2000); Caporossi and Hansen (2000, 2004); Hansen and Mladenović (2001a)). GVNS steps are explained in Algorithm 16 (Hansen and Mladenović, 2001a).

Function GVNS(x, $\mathcal{N}_k$, $\mathcal{S}$)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)
2 Find an initial point $x \in \mathcal{S}$
3 Choose the stopping condition
4 **while** *termination conditions not satisfied* **do**
5 Set $k \leftarrow 1$
6 **repeat**
7 Generate at random x', where $x' \in \mathcal{N}_k(x)$ //Shaking
8 Apply variable neighbourhood descent (VND) with x' as an initial solution to obtain the local optimum x'' //Shaking
9 **if** $f(x'') \leq f(x)$ **then**
Set $x \leftarrow x''$, and $k \leftarrow 1$
else
$k \leftarrow k + 1$// Next neighbourhood
until $k \leftarrow k_{max}$
10 **return** x

Algorithm 16: General Variable Neighbourhood Search

where VND in step (8) in Algorithm 16 is a min VND, that means it has less number of neighbourhood structures as in general one.

2.1.6 Skewed variable neighbourhood search

Skewed variable neighbourhood search (SVNS) explores the valleys far from the incumbent solution (Hansen and Mladenović, 2003). Indeed, in a large region problem, when the best solution has been found, to improve that solution, the search has to go further to obtain an

improved one. If small neighbourhoods have been used, reaching the global optimum requires a significant amount of computational time, which will make the search time consuming. To overcome this problem, SVNS has a flexible acceptance criteria to deal with this dilemma. It uses large neighbourhoods of the incumbent solution in order to escape from local optimum and to have a better solution.

In addition, some metaheuristics like simulated annealing and tabu search use the idea of diversification. They allow the search to accept worse solutions than the incumbent one to escape from stalling in valleys, where SVNS has the same idea. Moreover, the solutions are randomly choosen in distant neighbourhoods, which may make a substantial difference between them and the incumbent one and allow VNS to degenerate into a Multistart heuristic. Consequently, SVNS makes some compensation for distance from the incumbent solution. SVNS is explained in Algorithm 17 (Hansen and Mladenović, 2003).

Function SVNS(x, $k_{\max}$, $t_{\max}$, α)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)

2 Find an initial point $x \in \mathcal{S}$

3 **while** $t \leq t_{\max}$ **do**

4 Set $k \leftarrow 1$ and $x_{best} \leftarrow x$

5 **repeat**

6 Generate at random x', where $x' \in \mathcal{N}_k(x)$

7 Apply local search with x' as an initial solution to obtain the local optimum x'' //Shaking

8 **if** $f(x'') < f_{best}$ **then**
Set $f_{best} \leftarrow f(x)$ and $x_{best} \leftarrow x''$

9 **else if** $f(x'') - \alpha\delta(x, x'') < f(x)$ **then**
Set $x \leftarrow x''$ and $k \leftarrow 1$

else
$k \leftarrow k + 1$// Next neighbourhood

until $k \leftarrow k_{max}$

10 **return** x

Algorithm 17: Skewed variable neighbourhood search

In Algorithm 17 step 9 allows to move to worse solutions to avoid stalling in large valleys. The $\delta : \mathcal{S}^2 \leftarrow R$ is used to measure the distance between the local optimum found x'' and the incumbent solution x, where a move is made if $f(x'') - \alpha\delta(x, x'') < f(x)$. The $\alpha \in R^+$ is a parameter, and it is used to control the diversification. This function $\delta(x, x'')$ may or may not be defined as the distance function $\rho : \mathcal{S}^2 \leftarrow R$, which it is explained in (2.4) and (2.5), where it is used to define the $\mathcal{N}_k$.

Moreover, the α must be chosen in order to guarantee that the search goes far away from x when $f(x'')$ is larger than $f(x)$, but not too much larger (otherwise one will always leave x). In some case, the α can be found experimentally in each case, or it can be defined as a large value when δ is small for avoiding frequent moves from x to closer solution, where the

more sophisticated choices for finding α could be made through the learning process. For SNVS applications (see Brimberg et al. (2009); Souza and Martins (2008)).

2.1.7 Variable neighbourhood decomposition search

The variable neighbourhood decomposition search (VNDS) was introduced by Hansen et al. (2001). VNDS extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem.

The basic VNS is very useful for solving many combinatorial and global optimisation problems. However, if the problem has very large instances, the basic VNS almost practically fails to find a good quality solution in reasonable computational time, because it has limited tools to deal with big size problems. When the heuristic methods are applied to very large instance problems, their strengths and weaknesses become clearly apparent. Due to that the improvement scheme is desirable, where VNSD is improved to sort out this issue.

The main difference between VNS and VNDS is that VNS applies the local search method in the whole solution space $\mathcal{S}$. VNDS is divided at each iteration into a subproblem in some subspace, where VNS is used as a local search here, thus the two-level VNS-scheme arises. VNDS steps are explained in Algorithm 18 (Hansen et al., 2001).

At the beginning, the set of all solution attributes is defined as $\mathcal{A}$ and t_d, where t_d is an additional parameter and it is used as the running time for solving decomposed small size problem by VNS. At each iteration, VNDS chooses a subset $y \subseteq \mathcal{A}$ at random, where y is a set of k solution attributes present in x', but not in x ($y = x' \setminus x$). Then, a new local optimum y' has been found in the space y, where it is denoted as x'' in the space $\mathcal{S}$ ($x'' = (x' \setminus y) \setminus y'$). Due to the above, the VNDS is becoming popular with a number of successful applications (see Costa et al. (2002); Hansen et al. (2007b); Lazić et al. (2010); Lejeune (2006); Urosević et al. (2004)).

Function VNDS(x, $k_{\max}$, $t_{\max}$,t_d)

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_k$ is given by (2.2) or (2.3)

2 Find an initial point $x \in \mathcal{S}$

3 Let $\mathcal{A}$ be a set of all solution attributes

4 **while** $t \leq t_{\max}$ **do**

5 Set $k \leftarrow 1$

6 **repeat**

7 Generate at random x', where $x' \in \mathcal{N}_k(x)$

8 Let $y \subseteq \mathcal{A}$ be a set of k solution attributes present in x' but not in x, $(y = x' \setminus x)$ //Shaking

9 **repeat**

10 Find the local optimum in the subspace y by inspection or by some heuristics and name the incumbent by y'. Let x'' be in the whole space $\mathcal{S}$, where $x'' = (x' \setminus y) \setminus y'$

until $t \leq t_d$

11 **if** $f(x'') < f(x)$ **then**
Set $x \leftarrow x''$
else
$k \leftarrow k + 1$

until $k \leftarrow k_{max}$

12 **return** x

Algorithm 18: Variable neighbourhood decomposition search

2.1.8 Continuous variable neighbourhood search

Continuous variable neighbourhood search (CVNS) was introduced by Mladenović et al. (2003). It was developed to solve constrained and unconstrained continuous optimisation problems. The continuous box constrained nonlinear optimisation problem (COP) can be written as

$$(COP) \quad \begin{cases} \min \quad f(x) \\ \text{s.t.} \quad a_j \leq x_j \leq b_j \quad \forall j \in \{1, 2, \ldots, n\} \end{cases}$$

where $x = (x_1, \ldots, x_n)$, $f : R^n \rightarrow R$, $a, b \in R^n$ are the lower and upper bounds on the variables.

The COP, as defined above, naturally arises in many applications, e.g. in advanced engineering design, data analysis, financial planning, risk management, scientific modeling, etc. Most cases of practical interest are characterised by multiple local optima and, therefore, a search effort of global scope is needed to find the globally optimal solution (COP).

For solving COP, VNS has already been used in two different ways: with neighbourhoods induced by using an ℓ_p norm (Drazić et al., 2006; Liberti and Drazic, 2005; Mladenović et al., 2008) and without using an ℓ_p norm (Toksari and Guner, 2007). The metric function ρ_k can be defined as in (2.4) or (2.5). Thus, the neighbourhood $\mathcal{N}_k(x)$, where it denotes the set of solutions in the k–th neighbourhood of x, can be written by using the metric ρ_k as in (2.2) or (2.3), where metric the $\rho_k(x, y)$ and r_k are monotonically increasing with k, and r_k is a given radius of the neigbourhood $\mathcal{N}_k$.

In Drazić et al. (2006) a software package GLOB was developed for solving box constrained CGOP by using CVNS. Its steps are given in Algorithm 19.

The `Glob-VNS` procedure from Algorithm 19 contains the following parameters in addition to k_{max} (a maximum number of neighbourhoods used in the search) and t_{max} (total maximum time allowed):

1. *Values of radii* r_k, $k = 1, \ldots, k_{\max}$. Those values may be defined by the user or calculated automatically in the minimising process;

Function `Glob-VNS` (x, k_{max}, t_{max})

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$, as in (2.2) or (2.3)
2 Find an initial point $x \in X$
3 Select the array of geometry distributions types
4 **while** $t < t_{max}$ **do**
5 $k \leftarrow 1$
6 **repeat**
7 **for** *each (geometry, distribution) pair* **do**
8 Generate $x' \in \mathcal{N}_k(x)$ at random
9 Apply a local search on x' to obtain a local minimum x''//Shaking
10 **if** $f(x'') < f(x)$ **then**
11 $x \leftarrow x''$, `go to line 5`
12 $k \leftarrow k + 1$
 until $k = k_{\max}$
13 $t \leftarrow$ `CpuTime()`
14 return x

Algorithm 19: VNS for COP

2. *Geometry* of the neighbourhood structures $\mathcal{N}_k$, defined by the choice of the metric and their order. The usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms;

3. *Distribution* used for obtaining the random point x' from $\mathcal{N}_k$. Uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to much better performance on some problems. Besides uniform (u), we also implement hypergeometric distribution (h), the special distribution (denoted by 2) uses a specially designed distribution on ℓ_1 unit sphere, as follows:

   - The coordinate d_1 is taken uniformly on $[-1, 1]$, d_k is taken uniformly from $[-A_k, A_k]$ where $A_k = 1- \mid d_1 \mid - \ldots - \mid d_{k-1} \mid, k = 2, \ldots, n-1$ and the last d_n takes A_n with a random sign.
   - The coordinates of d are permuted randomly.

   Note that different choices of geometric neighbourhood shapes and random point distributions lead to different VNS-based heuristics. We denote them as (α, γ), where α and γ represents geometry (the metric) used and distribution, respectively.

Figure 2.3 explains all types of distributions. Moreover, for the local search phase GLOB includes several nonlinear methods like steepset descent, Rosenbrock, Nelder-Mead, and Fletcher-Reeves. The type of local search is chosen by the user to decide which one fits better with the problem. Furthermore, the type and order of geometries to generate the neighbourhood structures is a user decision, where not all of them should be included. Also, a random starting point in each neighbourhood is generated by the local search according to the chosen metric.

To sum up, `Glob-VNS` algorithm should contain the following:

- defined $t_{\max}$ which is the maximum running time for the search.
- defined $k_{\max}$ which is the maximum number of neighbourhood structures during the search.
- defined radii r_k or the procedure to generate them.

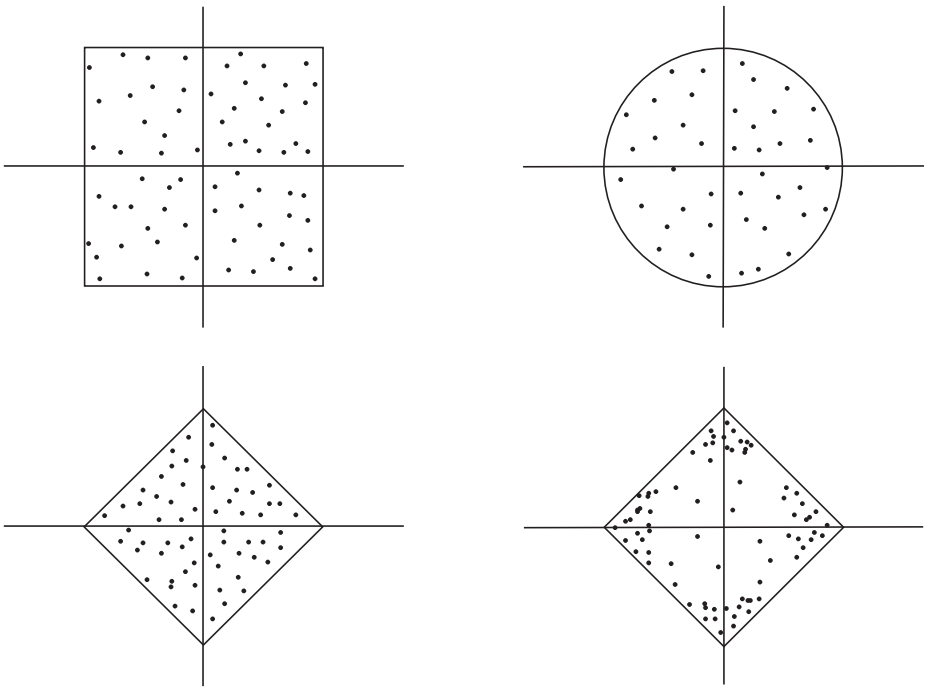


Figure 2.3: Distribution types

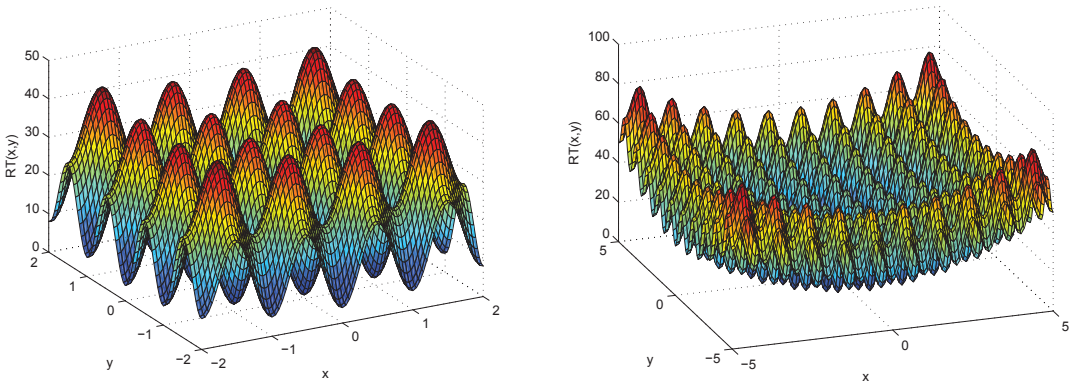


Figure 2.4: Rastrigin function

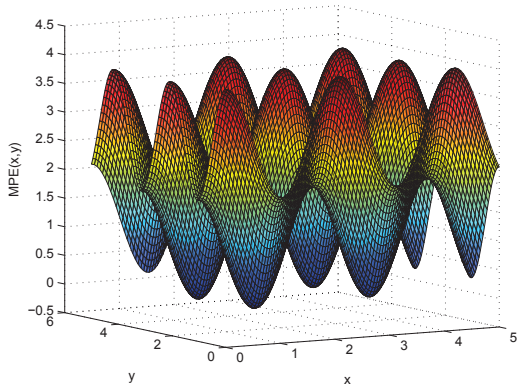


Figure 2.5: Molecular potential energy function

- defined choice of geometries which are used to build the neighbourhood structure.
- chosen types of distributions that will be used during the search.
- decided local heuristic method that will be used in the search.
- finally, decided order of neighbourhoods and distributions in the shaking steps.

Some figures for applying `GLOB-VNS` on continuous optimisation problems are given such as Rastrigin function in Figure 2.4 and Molecular potential energy function in Figure 2.5.

Continuous optimisation applications

In this part, some application on continuous optimisation problems, which have been solved by CVNS, will be explained.

Bilinear programming problem (BLP) is structured as a global optimisation problem with bilinear constraints. This problem has three sets of variables x, y and z, with cardinalities n_1, n_2 and n_3 respectively. This problem has a bilinear property, which means, when the set of y is fixed, BLP becomes a linear program in x and z, whereas when the set of z is fixed, the BLP is linear program in x and y. The steps of solving BLP can be written as

- Fixe one variable y (or z).
- LP_1: Solve this problem as a linear program in (x, z) (or (x, y)).
- LP_2: For z (or y), which it has been found in previous step, solve a linear program in (x, y) (or (x, z)).
- If stability is not reached return to LP_1.

This algorithm is suggested as a well-known Alternate heuristics, where it may be solved by a Multistart framework. Applying CVNS on BLP outperforms the Multistart Alternate heuristic (Audet et al., 2004). In CVNS algorithm, the neighbourhood $\mathcal{N}_k(x, y, z)$ for solution (x, y, z) is defined, then the alternate heuristic is used as the local search method.

Continuous Min-Max problem has been solved by CVNS. The algorithm starts by defining different neighbourhood structures. They are derived from the Euclidean distance, then the random point has been selected from the current neighbourhood. In the *local search* step, the gradient local search has been applied. This step will be repeated until the number of active functions in the current point equals n. The results have been compared with multi-level Tabu search, where CVNS outperforms better on quality and computing time for all test instances (Hansen and Mladenović, 2001b; Mladenović et al., 2003). For more applications for CVNS (see Mladenović et al. (2008); Toksari and Guner (2007)).

2.1.9 Reformulation descent variable neighbourhood search

The traditional techniques to solve an optimisation problem are attempted by considering its formulation and searching through the feasible region X. In fact, the same optimisation problem can often be formulated in different ways, where this allows the search to jump from one formulation to another. It has a local search method, which works totally within the formulation, and gives a final solution. Any solution which has been found by one of the formulations, should then be translated to an equivalent one in any other formulations.

This strategy helps to escape from local optima by switching from one formulation to another, where the local minima for the first one, may not be a local minima for another. This idea can help if the local search for each formulation will behave differently.

The VNS algorithm for *reformulation descent* (RD) is given in Algorithm 20. Let P be a given optimisation problem (combinatorial or continuous) and φ_1, φ_2 are two different formulations for the problem. It explains how the two formulations are incorporated, where the current active formulation is denoted as φ_{active}.

Function `VNS-RD` $(P, x, \varphi_1, \varphi_2, k_{max}, t_{max})$

Select the set of neighbourhood structures $\mathcal{N}_{k\varphi_1}$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_{k\varphi_1} : \mathcal{S}_{\varphi_1} \rightarrow P(\mathcal{S}_{\varphi_1})$

Select the set of neighbourhood structures $\mathcal{N}_{k\varphi_2}$, $k = 1, \ldots, k_{\max}$, where $\mathcal{N}_{k\varphi_2} : \mathcal{S}_{\varphi_2} \rightarrow P(\mathcal{S}_{\varphi_2})$

Find an initial point $x \in X$

Set $\varphi_{active} = \varphi_1$.

while $t < t_{max}$ **do**

  $k \leftarrow 1$

  repeat

    Generate $x' \in \mathcal{N}_{k\varphi_{active}}(x)$ at random

    Applying a local search on x' to obtain a local minimum x''

    if $f(x'', \varphi_{active}) < f(x, \varphi_{active})$ **then**

      $x \leftarrow x''$, and $k \leftarrow 1$

      Set $\varphi_{active} = \varphi_2$

    else

      $k = k + 1$

  until $k \leq k_{\max}$

  $t \leftarrow$ `CpuTime()`

return x

Algorithm 20: VNS for RD

where $k_{\max}$ could be equaled for φ_1 and φ_2 or not.

This idea was recently investigated in Mladenović et al. (2005). It was applied on circle packing problems (CPP) to investigate systematical changes between two formulations (one for Cartesian coordinates and one for polar coordinates system). It is shown that the stationary point in polar coordinates is not necessarily a stationary point in the Cartesian coordinates system. In this case a RD method is applied, which alternates between two

formulations until a final solution is found, where this one is a stationary point with respect to both formulations. The results obtained were comparable with the best known values, but they were 150 times faster than other single formulation approaches. An extension for RD has been suggested by using more than two formulations, this case is called *formulation space search* (FSS) (Mladenović et al., 2007). For more application (see Hansen et al. (2010); Hertz et al. (2008); Mladenović et al. (2007)).

2.1.10 Primal-dual VNS

The optimal solution can not be guaranteed by most of heuristic methods. Moreover, the difference between the solution obtained and the optimal one is completely unknown. Thus there is no information that could help to decide if the obtained solution is close to the optimal one or not. In the heuristic method, if the lower bound of the objective function value is known, the optimal solution may be guaranteed. To solve this issue, mathematical programming is applied on relaxing the integrality constraints on the primal variables. However, the commercial solvers may fail to find the exact solution when the relaxed problem has a large instance. Therefore, the new idea is to solve the dual relaxed problem with the primal one.

In Hansen et al. (2007a), the Primal-dual VNS has been successfully applied on large scale simple plant location problems (SPLP). It has been used to find the exact solution or the guaranteed bounds. The algorithm is given in Algorithm (21)

Function `PD-VNS` $(P, x, k'_{max}, k_{max}, t_{max})$

1 Solve the primal by using BVNS$(P, x, k'_{max}, k_{max}, t_{max})$

2 Find infeasible dual solution such that $f_P(x) = f_D(y)$

3 Use VNS to decrease the infeasibility of dual solution y

4 Find the exact dual solution

5 Apply branch-and-bound method on x and y

6 **return** x and y

Algorithm 21: Primal-dual VNS

This algorithm has three phases. The first one is based on VNS, and it finds the nearest optimal solution for the primal problem. Moreover, VNS decomposition is very powerful for solving large scale simple plant location problems with up to 15000 facilities and 15000 users. In the second phase, the approach is designed to find an exact solution to the relaxed dual problem. Then the standard branch-and-bound is applied on the original problem within tight upper and lower bounds, where they are obtained from the heuristic primal solution and exact dual one. More details on PD-VNS for SPLP can be found in Hansen et al. (2007a).

2.1.11 Parallel variable neighbourhood search

The parallel variable neighbourhood search strategy could be classified into three categories:

- **Low-level parallelism**. This strategy aims to speed up the computations by executing in parallel one or several tasks on one iteration. The implementation is divided among the master processor and the slave processors. The master processor dispatchs work to the other slave processors, then it has the results again. At this point the sequential algorithm continues. The difference from one parallel approach to another is how much work the slave processors will have.

- **Domain decomposition**. It is generally applied by dividing the vector of variables and the solution space into subspaces. For finding the solution, the VNS procedure is repeated for all subspaces to explore the whole region, which increases the exploration in the search space. The master processor has the partial slave's solutions and builds the complete solution. At this point, the new partitions are decided by the master processor which then restarts.

- **Multiple search**. It is obtained from multiple concurrent explorations of the solution space. Moreover, the concurrent searches may or may not use the same heuristic. They may or may not start from the same initial solutions. Besides, they may have a communication during the search or at the end to identify the best overall solution. This leads to two strategies known as the *independent search* methods, and the *cooperative multi-search* methods.

For more details and applications for parallel variable neighbourhood search (see Crainic et al. (2004); Garclá-López et al. (2002); Yazdani et al. (2010)).

2.1.12 Variable neighbourhood search with dynamic selection

In this subsection, the advance scheme of variable neighbourhood search will be highlighted with dynamic change of parameters and / or neighbourhood structure during the search. To characterise any local search metaheuristics, the next four components should be defined: problem formulation, neighbourhood structure, initial solution and sets of parameters. These components are fixed during the search. However, the behaviour of the search process could be changed which may make these components or some of them not efficient enough to reach the global optima. There is a possibility of switching between different formulations for the same problem during the search and it has been explained in subsection 2.1.9.

There are two possibilities of using the neighbourhood structure to improve the search process. First, by restricting the existing neighbourhood structure to a subspace of interest. Second, by switching between different types of neighbourhood structures. Moreover, updating the neighbourhood structure can be used when we are sure that some of the search space is not of interest anymore and it should be discarded. This idea is very similar to the tabu search (it has been discussed in chapter 1). If S is the solution space and let S' denotes the set of solutions not of interest thus the new solution space will be $S \setminus S'$. Furthermore, if the neighbourhood can be written as $\mathcal{N} : S \rightarrow P(S)$, thus the new reduction one can be formulated as $\mathcal{N} : S \setminus S' \rightarrow P(S \setminus S')$.

The other type of dynamic change of neighbourhood structure is called a variable neighbourhood descent (it is discussed in subsection 2.1.2) as proposed in Hu and Raidl (2006). The basic idea behind VND is switching systematically between different neighbourhood structures $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_n$. The search process starts with the first neighbourhood structure $\mathcal{N}_1$ until the local optima has been found. It will then switch to neighbouhood structure $\mathcal{N}_2$, and if the new improved solution has been found the search will start again from $\mathcal{N}_1$. Otherwise it will continue with $\mathcal{N}_3$ and so on. If the last neighbourhood structure $\mathcal{N}_n$ has been used without any improvement, the VND will terminate with a solution, which is repre-

sented as a local optima to all neighbourhood structure. For some successful application (see Hansen et al. (2006); Hu and Raidl (2006); Hu et al. (2009); Puchinger and Raidl (2008)).

Chapter 3

Censored quantile regression

This chapter focuses on continuous variable neighbourhood search (VNS) for solving censored quantile regression (CQR). CQR models are very useful for the analysis of censored data when standard linear models are felt to be inappropriate. However, fitting censored quantile regression is hard numerically due to the fact that the function that has to be minimised (Powell estimator) is not convex, nor concave in regressors. The performance of standard methods is not satisfactory, in particular if a high degree of censoring is present. The usual approach in the literature is to simplify (linearise) the estimator function and show theoretically that such approximation tends to provide good real optimal values.

In this chapter a new approach to solve CQR will be suggested, i.e., the nonlinear, non-convex, and non differentiable optimisation problem is solved directly. Our method is based on variable neighbourhood search approach, a recent successful technique for solving global optimisation problems. Simulation results presented indicate that our new method can improve the quality of the censored quantile regression estimator considerably.

This chapter is organised as follows. Section 3.1 describes the censored quantile regression problem and Powell estimator. In section 3.2, the literature review of censored quantile regression is presented. The variable neigbuorhood search approach for solving censored quantile regression by the Powell estimator, and its algorithm are presented in section 3.3. Section 3.4 includes details on how the data instances are generated and then reports extensive computational analysis.

3.1 Description of the problem

The research of quantile regression began four decades ago. The quantile in general can be written as, if there is a random variable y with probability function

$$F(y) = P(Y \leq y) \tag{3.1}$$

the quantile regression on θth quantile is the inverse of the cumulative distribution function, $F^{-1}(\theta)$, and it can be formulated as,

$$F^{-1}(\theta) = inf\{y : F(y) \geq \theta\} \tag{3.2}$$

where $0 < \theta < 1$. The median quantile will be given when $\theta = 1/2$. Furthermore, the quantile divides the population into segments. It is called quintiles, if the population is divided into five segments, each segment has equal proportions of the population. Furthermore, it is called quartiles, if it divides the population into four equal segments. Also, it could be called deciles, if the population is divided into ten equal segments.

The quantile regression model (QR) was introduced by Koenker and Bassett (1978). This estimator is the most famous approach, where the quantiles of responses are linearly related to the input vector. It can be formulated as,

$$y_i = x_i'\beta_\theta + \varepsilon_{\theta i} \qquad , i = 1, \ldots, n, \tag{3.3}$$

where y_i is the θth quantile function due to the response of the input vector x_i. For estimating the $(\beta_\theta, \varepsilon_\theta)$, the θth quantile regression are defined to minimise the objective function

$$f(\beta, \varepsilon) = \sum_{i=1}^{n} \rho_\theta(y_i - x_i'\beta_\theta) \tag{3.4}$$

where $\varepsilon_{\theta i}$ is independent and identically distributed (i.i.d.) with distribution function, and the ρ_θ is the check function and it can be written as

$$\rho_\theta(\lambda) = [\theta - I(\lambda < 0)]\lambda \tag{3.5}$$

where $I(.)$ is the indicator function, it is given as

$$I(\lambda < 0) = \begin{cases} 1 & if\ \lambda < 0 \\ 0 & if\ \lambda \geq 0 \end{cases}$$

Furthermore, the censored quantile regression with fixed censored point can be written as,

$$y_i = \max\{y_0,\ x_i'\beta_\theta + \varepsilon_{\theta i}\} \qquad , i = 1, \ldots, n, \tag{3.6}$$

where y_i are given latent or dependent values, and y_0 is a given censoring point. x_i' is $g-$dimensional vector of the independent covariates, which are observed (given) for each i. θ represents confidence level, and $\varepsilon_{\theta i}$ is an unobservable error term which is assumed to be normally distributed (also known as quantile). $\beta_\theta = (\beta_{0\theta}, \ldots, \beta_{g-1,\theta})^T$ is $g-$ dimensional parameter of interest that we would like to find. The censored quantile regression model is sometimes referred to by economists as the censored "Tobit" model (Tobin, 1958). The regression is used to quantify the relationship between a response variable y_i and some covaraites x_i, $i = 1, \ldots, n$.

Powell (Powell, 1984, 1986) suggested an intuitive estimator for censored quantile regression model. This estimator solves

$$f(\beta, \theta, y_0) = \min_\beta \frac{1}{n} \sum_{i=1}^{n} \rho_\theta(y_i - \max\{y_0,\ x_i'\beta_\theta\}), \tag{3.7}$$

where ρ_θ is given as (3.5) and $I(.)$ is the usual indicator function. Since θ and y_0 are given, function (3.7) depends only on β. Therefore, we denote Powell estimator (3.7) as $f(\beta)$. Right censoring estimator can be easily found from (3.7) by replacing the max with min, where it is mostly used with the duration model applications. In most econometric applications the censoring point is fixed $y_0 = 0$ as in the original "Tobit" model but this case is not a general one. However, the most important point that the censoring point y_0 is known for all observations, where in the Powell estimator it is fixed. The Powell estimator has several disadvantages.

- First, the censoring point y_0 must be known.
- Second, obtaining the global minimum of (3.7) can be difficult because the objective function $f(\beta)$ is non convex, nor concave and even non differentiable in β.

An example of $f(\beta)$ with $g = 2$, $n = 100$, $\theta = 0.95$, $y_0 = 0$ and normally distributed random variable ε is illustrated in Figure 3.1 (more details of this instance will be given in section 3.4). Thus, the problem belongs to the global (nonlinear) optimisation area, and

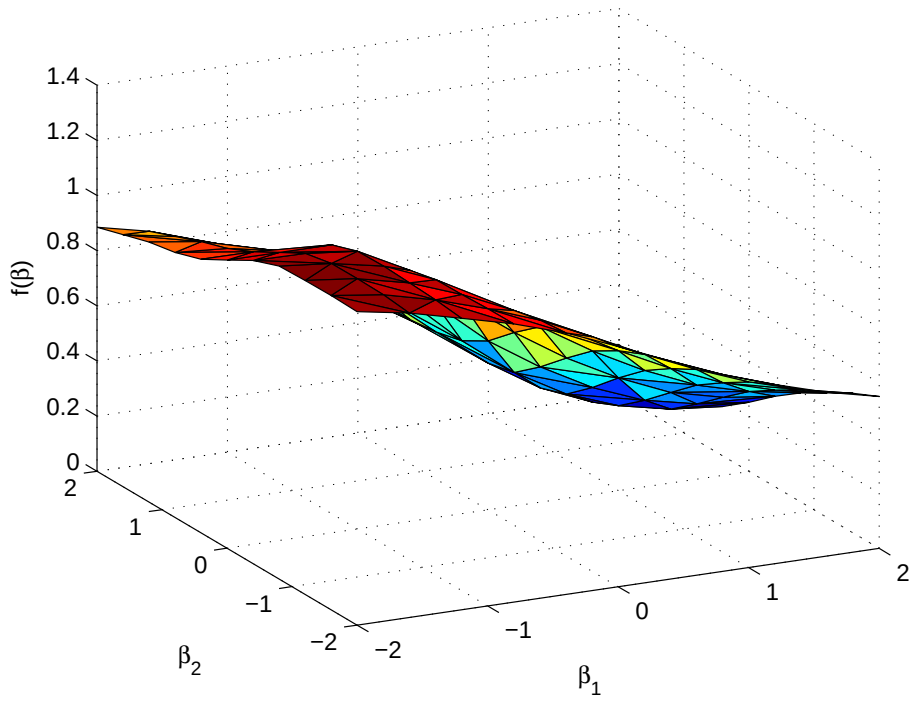


Figure 3.1: Powell function $f(\beta_1, \beta_2)$ with $n = 100$, $\theta = 0.95$, $y_0 = 0$ and Gaussian r.v. $\varepsilon \neq 0$

.

may have many local optima. Consequently, standard optimisation tools, that require the objective function to be differentiable and/or convex, may fail to discover the true CQR estimator.

However, several convex optimisation algorithms have been adapted for finding CQR, where the Powell estimator (3.7) has been used see Fitzenberger (1997) for a survey of such algorithms). More details are given in the next section 3.2.

Those algorithms have difficulties in solving CQR problems. They exhibit a high degree of complexity in their implementation. Most of them achieve convergence to local optima,

whereas finding the global optima for these problems require a heavy computational load.

Therefore, the basic question we would like to answer in this chapter is, what approach is more promising: use the original Powell estimator (3.7) and solve CQR problem approximately, or using a simplified approximative model and solve it exactly? In contrast to the majority of authors who used to simplify models, here we suggest, for the first time, the use of approximate global optimisation method for solving (3.7). In order to do that, we developed nonlinear programminng (NP) code based on variable neighbourhood search (VNS) metaheuristic (or framework for building heuristics). As far as we know, this is the first time that some metaheuristics approach is used for solving the CQR problem. Based on computational results section, it appears that our approach outperforms other methods from the literature.

3.2 Literature review

There are many applications of censored quantile regression. Econometrics and statistics have been interested in the CQR model in recent years, especially due to unknown conditional heteroscedasticity and their robustness to distributional misspecification of error term. Various applications of CQR have been published (Amemiya, 1982; Buchinsky and Hahn, 1998; Chaudhuri et al., 1997; Chen and Khan, 2000; Chernozhukov and Hong, 2002; Portnoy, 1991; Rao and Zhao, 1995; Yu. et al., 2003).

As mentioned above, the objective function of CQR is non convex, nor concave and non differentiable. The optimisation problem may therefore have many local optima which means that the local optimisation methods could be terminated in local optima instead of global optima. Many algorithms described in the literature failed to provide satisfactory results (Fitzenberger, 1997).

Womersley (1986) linearised the problem by using a reduced-gradient algorithm. In that way a local minimiser is found by using linear programming.

Dueck and Scheuer (1990) used a new approach called “threshold accepting” (TA). This algorithm is applied on the traveling salesman problem and the construction of error-

correcting codes problem. The computational results show that the algorithm is very close to optimum in 442-cities traveling salesman problem. This algorithm is also used to provide convergence results (Althofer and Koschnick, 1991).

Pinkse (1993) tries to solve CQR with the Powell estimator by using the simplex algorithm with Nelder-Mead method. He finds that the simplex algorithm is preferably not used for estimating CQR.

Buchinsky (1994) has used iterative linear programming algorithm (ILPA) to study the change of wages in the United States. The algorithm is applied on March Current Population Survey since 1964. Besides, the changes in the return to schooling and the experience at different wages are also examined. The linear model is divided into two groups, a one group and a sixteen group model. The computational results show that the experience at different wages and the returns to schools are similar in patterns of change, but they are different across quantiles of the wage distribution. ILPA algorithm is available in many statistics software likes STATA, and it can be easily written by TSP or R software languages.

The interior point algorithm for solving nonlinear quantile regression problems (NLRQ) is discussed by Koenker and Park (1996). This algorithm has been applied on different quantile problems, where the linear censored quantile problem of Powell estimator is also included. NLRQ algorithm is available in R software.

Fitzenberger (1997) adapts the simplex algorithm of Barrodale and Roberts (1974) with ℓ_1 norm. This algorithm is called a BRCENS, where it studies the standard quantile regressions and the CQR case. The objective function is the piecewise linear, where it depends on an exact fit to p observations. The computation results are obtained by deleting one of the p points from the "basis". This strategy ensures convergence to a local optima.

The ILPA and NLRQ algorithms cannot guarantee the convergence to global optima, even they cannot guarantee the convergence to local optima, whereas the BRCENS can guarantee the convergence to local optima. Furthermore, in contrast to BRCENS and the simplex algorithm in Pinkse (1993), TA algorithm almost guarantees convergence to the global minimum with an infinity number of iterations. However, TA algorithm improves the estimation of CQR better than any other algorithms, but it needs more CPU time. For that

it has been used widely with CQR.

Since the late 1990s there have been different modification of CQR model . For example, Buchinsky and Hahn (1998) introduced an alterative to the CQR model, this model is a globally convex one. It can be solved by linear programming. Their stepwise estimator has been used to estimate the coefficients. During the first step, a non-parametric approach is applied on the estimation of the probability of censoring point at each observation. In the second step, the uncensored observations are reweighted by using the estimated censoring point. They then applied the ILPA algorithm in two cases, with unknown censoring point, and with a Powell estimator with a fixed censoring point. The results show that the algorithm outperforms for the bias induced by censoring.

Chernozhukov and Hong (2002) suggest the three-step estimator for solving CQR. This estimator has variant fixed censoring point probabilities, where it is a stepwise estimation approach as in Buchinsky and Hahn (1998). The parametric model has been used to estimate the censoring points at the first step. They use the estimated censoring points to determine the observations with the small censoring probability. The computational results show that the three-step estimator is useful for small sample or models with many regressors. The estimators suggest in Buchinsky and Hahn (1998); Chernozhukov and Hong (2002) are asymptotically equivalent to the original Powell estimator, they do not allow explicitly for censored observations to be interpolated by the estimated CQR, where the interpolation property suggests finding an exact solution by using a computationally expensive algorithm.

In Honoré et al. (2002), the distribution function is estimated by the censoring points. This function is assumed to be independent of the response variables and covariates. Then, the CQR is used when the censoring point is unknown.

The estimator suggested in Portnoy (2003) mimics the Kaplan-Meier estimator. It reweights the censored observation if the censored region contains the value of it's conditional quantile function. This estimator is a right censored one, where it has been started from the lower tail of the data. This CQR estimator is different from the Powell estimator. Additional applications are described in the literature (Chernozhukov and Hong, 2003; Blundella and Powell, 2007; Qian and Peng, 2010; Portnoy and Lin, 2010; Pang et al., 2010;

Hosseinkouchack, 2011).

3.3 Variable neighbourhood search for censored quantile regression

As mentioned earlier, CQR problem belongs to continuous global optimisation. In this section, the general rules of VNS for solving global optimisation problems (GOP) are discussed. This is followed by an explanation of their use in solving CQR.

3.3.1 Variable neighbourhood search metahuristics

This subsection gives a brief revision of general variable neighbourhood search. Variable neighbourhood search (VNS) (Mladenović and Hansen, 1997) is a metaheuristic based upon systematic changes of neighbourhoods in order to enable finding a better solution in distant parts of a solution space. VNS is designed for solving both continuous and discrete optimisation problems, that may be formulated as

$$\min\{f(\beta)|\ \beta \in \mathcal{B}, \mathcal{B} \subseteq \mathcal{S}\}. \tag{3.8}$$

$\mathcal{S}, \mathcal{B}, \beta$ and f respectively denote the *solution space*, *feasible set*, a *feasible solution* and a real-valued *objective function*. If $\mathcal{S}$ is a finite but large set, a *combinatorial optimisation* problem is defined. If $\mathcal{S} = R^n$, we refer to *continuous optimisation*. An *exact algorithm* for problem (3.8), if one exists, finds an optimal solution β^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $\mathcal{B} = \emptyset$.

Let $\mathcal{N}_k,\ k = 1, \ldots, k_{max}$, denotes a finite set of pre-selected neighbourhood structures and let $\mathcal{N}_k(\beta)$ be the set of solutions in the k^{th} neighbourhood of β. The neighbourhood structures $\mathcal{N}_k$ may be induced from one or more metrics introduced into a solution space $\mathcal{S}$, either discrete or continuous. We define $\beta'' \in X$ as a local minimum w.r.t. $\mathcal{N}_k$, if there is no solution $\beta \in \mathcal{N}_k(\beta'') \subseteq \mathcal{B}$ such that $f(\beta) \leq f(\beta'')$.

Those simple facts are used within VNS in several different ways (see for example recent surveys of VNS in (Hansen et al., 2008, 2010)). The deterministic change of neighbourhoods

leads us to a so-called Variable neighbourhood descent (VND) heuristic. The basic VNS (BVNS) combines deterministic and random search (Mladenović and Hansen, 1997). Its pseudo-code is given in Algorithm 22.

Function `BVNS` $(\beta, k_{max}, t_{max})$;
1 **repeat**
2 $k \leftarrow 1$;
3 **repeat**
4 $\beta' \leftarrow$ `Shake`(β, k);
5 $\beta'' \leftarrow$ `LocalSearch`(β') ;
6 If $(f(\beta'') < f(\beta))$ $\beta \leftarrow \beta''$; `goto 2`;
8 $k \leftarrow k + 1$
until $k = k_{max}$;
9 $t \leftarrow$ `CpuTime()`
until $t > t_{max}$;

Algorithm 22: Steps of the Basic VNS for CQR

Let β be the incumbent (the best solution found so far). Within BVNS a point β' from the neighbourhood k of β ($\beta' \in \mathcal{N}_k(\beta)$) is taken at random where $k = 1, \ldots, k_{\max}$. Such a point is an initial one for a local search routine that provides local minimum β''. If $f(\beta'')$ is better (smaller in the case of minimisation) then the new incumbent is β'' ($\beta \leftarrow \beta''$) also k is set to 1 ($k \leftarrow 1$) and the process is repeated. Otherwise we generate a random point from the larger neighbourhood ($k \leftarrow k + 1$). The only parameter for the BVNS is the number of neighbourhoods used ($k_{\max}$). Once that neighbourhood is reached without finding improvement, k is again set to 1. The process is repeated until some stopping criterion, such as maximum CPU time $t_{\max}$ used, is satisfied (see Figure 3.2).

We may view the VNS as a "shaking" process, where a movement to a neighborhood further from the current solution corresponds to a harder shake. Unlike random restart, the VNS allows a controlled increase in the level of the shake. In this chapter we design the GVNS heuristic for solving the CQR problem, by minimising the nonlinear Powell function $f(\beta)$.

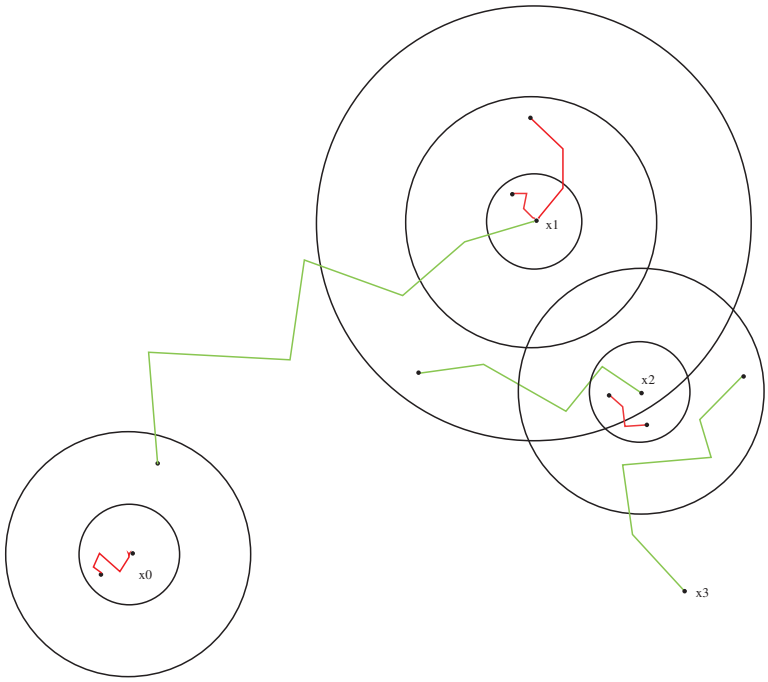


Figure 3.2: Illustration of the Basic Variable Neighbourhood Search (BVNS)

3.3.2 VNS for CQR

In this subsection we explain how we use VNS to solve the CQR problem. $f(\beta)$ defined in (3.7), is a nonlinear objective function with continuous variables $\beta_0, \ldots, \beta_{g-1}$. Thus, the CQR problem may be solved as an unconstrained nonlinear program. If $\theta = \frac{1}{2}$, then $f(\beta)$ gives the median (Chernozhukov and Hong, 2002). Observe also that any unconstrained nonlinear program may be considered as box constrained, if left and right values of variables that define a box are set to the same large negative and positive values a_i and b_i. Therefore, given input data $X = (x_{ij}), i = 1, \ldots, n;\ j = 1, \ldots, g-1$, $Y = (y_1, \ldots, y_n)$, $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)$ and the value of $\beta = (\beta_0, \ldots, \beta_{g-1})$, the pseudo-code for evaluating the Powell estimator is given in Algorithm 23.

Function `Powell`$(\beta, X, Y, y_0, \varepsilon)$

```
1 Powell← 0
2 for i = 1, ..., n do
3     s ← 0
4     for j = 1, ..., g − 1 do
5         s ← s + x_ij β_j
6     r ← y_i − max{y_0, s − ε_i}
7     Powell ← Powell + θr
8     if r < 0 then
9         Powell ← Powell - r
```

Algorithm 23: Pseudo-code for finding Powell estimator value

Neighborhoods - Shaking. For solving GOP, VNS has already been used in two different ways: with neighbourhoods induced by using an ℓ_p norm (Drazić et al., 2006; Mladenović et al., 2008; Liberti and Drazic, 2005) and without using an ℓ_p norm (Toksari and Guner, 2007). Here we apply VNS that uses the ℓ_p norm, i.e., we define distances between any two solutions β and γ as

$$\delta(\beta, \gamma) = (\sum_{i=0}^{g-1} |\beta_i - \gamma_i|^p)^{\frac{1}{p}}, \tag{3.9}$$

or

$$\delta(\beta,\gamma) = \max_{0\le i\le g-1} |\beta_i - \gamma_i|, \quad p=\infty. \tag{3.10}$$

The neighbourhood $\mathcal{N}_k(\beta)$ denotes the set of solutions in the k–th neighbourhood of β, and using the metric δ, it is defined as

$$\mathcal{N}_k(\beta) = \{\gamma \in \mathcal{B} \mid r_{k-1} \le \delta(\beta,\gamma) \le r_k\}, \tag{3.11}$$

where r_k is a given radius of neigbourhood $\mathcal{N}_k$ $(k = 1,\ldots,k_{\max})$.

Our `CQR-VNS` procedure for solving the CRQ problem contains the following parameters in addition to k_{max} (a maximum number of neighbourhoods used in the search) and t_{max} (the maximum time allowed in the search):

(i) *Values of radii* $r_k,\ k = 1,\ldots,k_{\max}$. These values may be defined by the user or calculated automatically during the minimisation process. The geometry of the neighbourhood structure is induced by the ℓ_1 (3.9) and ℓ_∞ (3.10) norms. We use balls as in (3.11). Radii $r_1 \le r_2 \le \ldots \le r_{k_{\max}}$ are automatically computed as follow: let $\beta = (\beta_0,\ldots,\beta_{g-1})^T \in R^g$ be the current incumbent solution and let

$$a_j \le \beta_j \le b_j,\ \ j = 0,\ldots,g-1 \tag{3.12}$$

defines a box or hyper-cube

$$H = \prod_{j=0}^{g-1} [a_j, b_j]$$

around the incumbent solution β. In order to find k_{max} neighbourhoods automatically and thus make our `CQR-VNS` more user-friendly, we divide $\beta_j - a_j$ and $b_j - \beta_j$ into $k_{\max}$ intervals:

$$\underline{\delta}_j = \frac{\beta_j - a_j}{k_{\max}};\ \ \overline{\delta}_j = \frac{b_j - \beta_j}{k_{\max}}.$$

Then the $k_{\max}$ hyper-cubes (boxes) $H_1, H_2, \ldots, H_{\max}$ around the incumbent (the best solution found so far) β are given

$$a_j + (k-1)\underline{\delta}_j \le \beta_j \le b_j - (k-1)\overline{\delta}_j, k = 1,\ldots,k_{\max} \tag{3.13}$$

or

$$\underline{d}_{jk} \le \beta_j \le \overline{d}_{jk}, k = 1,\ldots,k_{\max} \tag{3.14}$$

Figure (3.3) illustrates our construction of continuous neighbourhoods as hyper-cubes for the case of $k_{\max} = 3$ and $g = 2$ (or $\beta = (\beta_0, \beta_1)^T$).

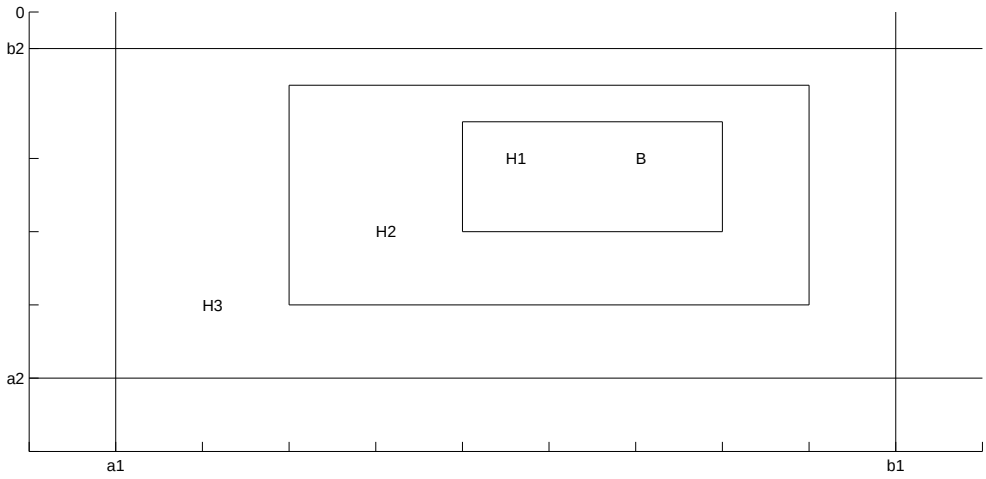


Figure 3.3: Automatic construction of neighbourhoods with $g = 2$ and $k_{max} = 3$.

(ii) *(Geometry, distribution) pairs*. Geometry of neighbourhood structures $\mathcal{N}_k$ is defined by the choice of the metric functions used in the search through the solution space. The usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms. Their order in the search is also important within VNS. Different *distributions* may be used for obtaining the random point y from the same neighbourhood $\mathcal{N}_k$ in the *Shaking* step. Uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to much better performance on some problems. Besides uniform (u), we also implement the hypergeometric distribution (h)(Drazić et al., 2006; Mladenović et al., 2008). The special distribution (h) is designed as follows:

- The coordinate β_1 is taken uniformly on $[-1, 1]$, β_k is taken uniformly from $[-A_k, A_k]$ where $A_k = 1 - \mid \beta_1 \mid - \ldots - \mid \beta_{k-1} \mid, k = 2, \ldots, g-1$ and the last β_n takes A_n with a random sign.

- The coordinates of β are permuted randomly.

Note that different choices of geometric neighbourhood shapes and random point distributions lead to different VNS based heuristics. We denote them as (α, γ), where α and γ represents geometry (metric) and distribution used, respectively. Therefore, in total we have 6 dif-

ferent variants of VNS defined by (geometry, distribution) pairs: (ℓ_1, u),(ℓ_2, u),(ℓ_∞, u),(ℓ_1, h), (ℓ_2, h), (ℓ_∞, h). Note that "u" denotes uniform distribution, while "h" denotes hypergeometric (special) distribution. For simplicity, we will denote those variants in pseudo-code as (1,1),(2,1),(3,1),(1,2),(2,2) and (3,2): (1,1)= (ℓ_1, u), (2,1)=(ℓ_2, u), etc. For example, pair (3,2) indicates that ℓ_∞ norm (3) and the special distribution (2) are used in the shaking step.

However, after extensive computational analysis, we select on four (geometry, distribution) pairs in our `CQR-VNS` in the following order: `distribution_type_order` = (1,2) (1,1) (3,1) and (3,2). After that a radius from interval $[0, r_k]$ is taken at random in order to get a point from $\mathcal{N}_k(x)$. Therefore, a random point within the Shaking step of `CQR-VNS` is generated in two steps: (i) find random direction; (ii) find random radius along that direction.

Local Search. As a local search for solving CQR we apply the direct search Nelder-Mead nonlinear programming method since it does not use derivatives. The left and right boundaries a_j and b_j for variables are defined as appropriate. The `GLOB` has six local search methods: steepest descent, Fletcher-Powell, Fletcher-Reeves, Nelder-Mead, Hook-Jeeves and Rosenbrock, where we chose Nelder-Mead nonlinear programming method by an empirical way. At the beginning, we fixed the other parameters on `GLOB` with fixed maximum running time and we run the code for each method. Then we found that the Nelder-Mead method gave better results than the other five methods. For that the Nelder-Mead method has been used here as a local search method for solving the censored quantile regression problem.

Pseudo-code. The algorithm Glob-VNS for solving CQR is given in Algorithm 24, where $k_{\max}$ and $t_{\max}$ are usual VNS parameters, given by user.

Function `CQR-VNS` $(\beta^*, k_{max}, t_{max}, X, Y, y_0, \varepsilon)$

1 Select $(geometry, distribution)$ pairs as: (1,2), (1,1), (3,1), (3,2)
2 Choose an initial point $\beta^* \in \mathcal{B}$ at random
3 $f^* \leftarrow$ `Powell`$(\beta^*, X, Y, y_0, \varepsilon)$
4 $\beta \leftarrow \beta^*$, $t \leftarrow 0$
5 **while** $t < t_{max}$ **do**
6 $k \leftarrow 1$
7 **repeat**
8 **for** *each (geometry, distribution) pair* **do**
9 $\beta' \leftarrow$ `Shake`(β, k) // Get $\beta' \in \mathcal{N}_k(\beta)$ at random
10 $\beta'' \leftarrow$ `Nelder-Mead`(β', f) // Get local minimum β'' by Nelder-Mead
11 **if** $(f < f^*)$ **then**
12 $\beta \leftarrow \beta''$; $f^* \leftarrow f$; `go to line 6`
13 $k \leftarrow k + 1$
  until $k = k_{max}$
14 $t \leftarrow$ `CpuTime`()

Algorithm 24: VNS for CQR

After choosing $(geometry, distribution)$ pairs and random initial solution $\beta^* \in R^{g-1}$ in steps 1 and 2 respectively, we apply Algorithm 23 to find the Powell estimator $f(\beta^*)$. We denote with β the incumbent solution. As explained in Algorithm 22, outer loop of VNS is running until a predefined stopping condition is met. The inner loop is repeated k_{max} times, if there is no improvement in regressors β. In each neighbourhood a random point from the $\mathcal{N}_k(\beta)$ is taken (line 9) and the well known Nelder-Mead unconstrained nonlinear programming code run (line 10). The local minimum value for Powel's estimator is denote with f. If a better solution is obtained, we save it (line 12) and repeat all process with the first neighbourhood (i.e., return to step 6).

3.4 Computational results

We perform extensive computational analysis to investigate how our new `CQR-VNS` method compares with other approaches. We first give general rules for the computational simulation performed, and then present comparative results on various test instances.

Methods compared. We compare our `VNS-CQR` with the following approaches from the literature.

1. The first group of methods are the same as those described by Bilias et al. (2000):
   - the direct heteroscedastic bootstrap method;
   - modified bootstrap and
   - resampling methods.
2. The second group of methods are from Buchinsky and Hahn (1998):
   - CV method which denotes theCQR-LP estimator with log likelihood cross-validated bandwidth;
   - CVa method, which denotes the CV estimator with bandwidth adjusted to conform with assumption K;
   - PR method, which denotes the CQR-LP estimator with probit estimates for censoring probability;
   - HO and HOa are the same as CV and CVa except that the kernel function involves a higher order kernel, and
   - CR denotes Powell's estimator.
3. Lastly, we apply our method to an extramarital affairs. Data set taken from Fair (1976).

Computer support. Our code was written in C++ and complied with Microsoft Visual Studio 8.0. The program was run on Intel(R) Core(TM) 2 at 1.73 GHz with 2 GB of RAM. Unfortunately, there is no information about the computers which were used to get results by

other methods. Therefore, the efficiency (i.e., the running CPU time of methods) could not be compared in this study. The comparison will therefore be restricted to their effectiveness or precision.

VNS parameters. Along the space dimension, initial and boundary conditions which are different in each test instance, in the CQR-VNS we used the following parameters.

- CPU time was limited to $t_{max} = 5$ seconds;
- The number of neighbourhoods structures used is set to 10, $k_{max} = 10$;
- We choose the Nelder-Mead local search method. It stops when one among the following three criteria are met:
  - a diameter of a simplex is less than 0.1e-5 (ls_eps = 0.1e-5),
  - the difference between two consecutive objective function values is less than 0.1e-5 (ls_fun_eps = 0.1e-5) and
  - the number of iterations reached 500.

3.4.1 GLOB-VNS for finding standard and percentile

In this part, we compared our CQR-VNS (whose pseudo-code is given in Algorithm 24) with the three algorithms used by Bilias et al. (2000). There the following model is considered:

$$y_i = \max\{\beta_0 + x_{1i}\beta_1 + x_{2i}\beta_2 + \varepsilon_i, y_0\}. \tag{3.15}$$

The details regarding simulation are listed below:

- x_1 is generated as a Bernoulli distribution centered at zero, with the success probability equal to $\frac{1}{2}$;
- x_2 is a standard normal variable $N(0, 1)$;
- The censoring point is $y_0 = 0$;
- Three different types of error ε are considered:

  - a standard normal distribution;
  - a heteroscedastic normal $(1 + x_2) \times N(0, 1)$, and
  - a normal mixture $0.75 \times N(0, 1) + 0.25 \times N(0, 4)$, as suggested in (Bilias et al., 2000).

- It is assumed that the best estimator values are known and all equal to 1, $(\beta_0, \beta_1, \beta_2) = (1, 1, 1)$. Then y_i is calculated by using formula (3.15).
- For each of the following confidence level $\theta \in \{0.95, 0.90, 0.85\}$ the standard (S) and percentile (P) methods (Efron and Tibshirani, 1993) are used to construct confidence intervals. In particular, we compare the 95%, 90% and 85% confidence levels for each type of error.
- A size $n = 100$ of random sample is generated, i.e., $\{(x_{1i}, x_{2i}, y_i), i = 1, 2, \ldots, 100\}$. Those data space points are obtained with the three different types of error ε. Two of them are plotted in Figure 3.4.
- The simulation is repeated 1000 times and the average results reported.

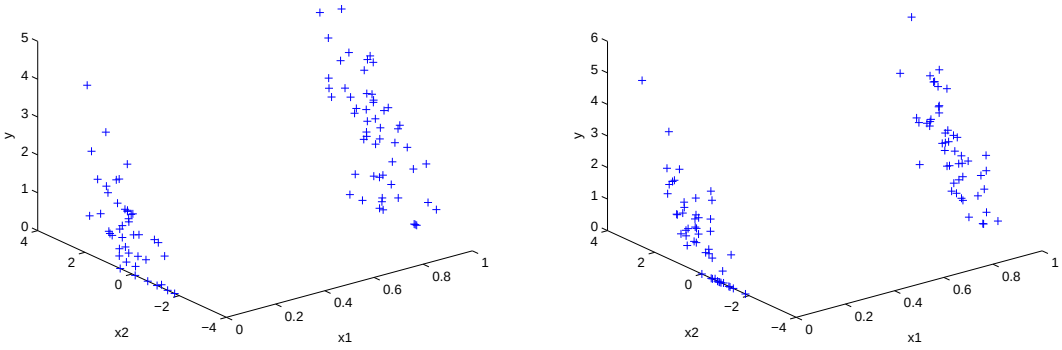


Figure 3.4: Points (x_{1i}, x_{2i}, y_i), $i = 1, \ldots, 100$, in data space with the standard normal (left) and normal mixture (right) errors with fixed $\beta_1 = 1$ and $\beta_2 = 1$.

In Figure 3.1 an instance of this type is plot in the regressor space (β_1, β_2), where the value of β_0 is fixed to 1. Powell's estimator values are obtained by applying Algorithm 23

and taking (β_1, β_2) in each point of the square grid [0,2]×[0,2] and increment 0.2 for each variable: $\beta_{jk} = 0.2 \cdot k, \forall j = 1, 2; \forall k = 0, \ldots, 10$. The version of the same instance, but with $\varepsilon_i = 0$ in (3.15), is presented in Figure 3.5.

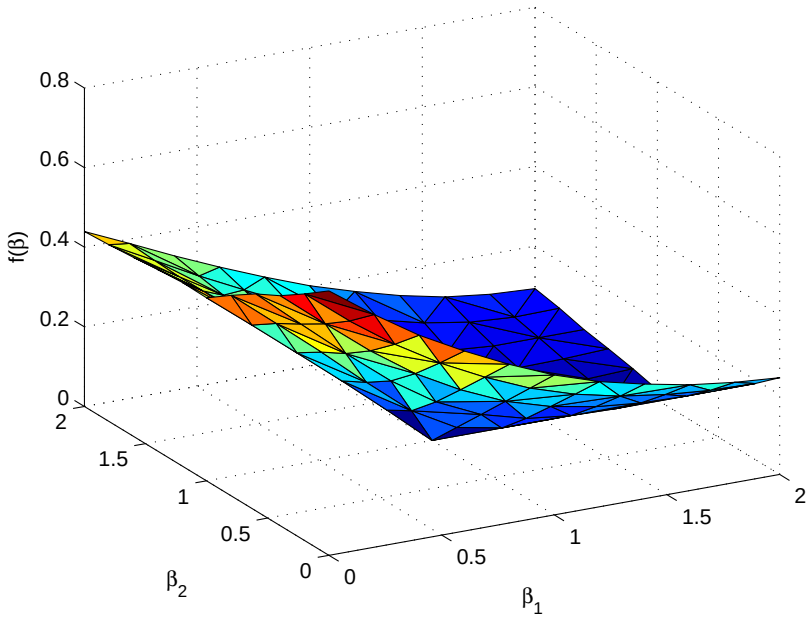


Figure 3.5: Censored Quantile Regression function $f(\beta)$ and $\varepsilon = 0$

.

Table 3.1 contains results for the estimation of the regression coefficients β_2 only. We compare the empirical coverage probabilities to the other three algorithms used in Bilias et al. (2000). Therefore, we investigate the finite sample performance of four methods: our VNS for CQR (CQR-VNS), the direct heteroscedastics bootstrap method (Bootstrap for short), the resampling method (Resampling for short) and Bilias, Chen and Ying's bootstrap method (M-Bootstrap for short) (Bilias et al., 2000). The quality of solutions obtained by CQR-VNS may be seen in Figure 3.6 as well

Table 3.1: Empirical coverage probabilities for confidence intervals

		Bootstrap	*Resample*	$M-Bootstrap$	VNS
Confidence level		ECP	ECP	ECP	ECP
Standard Normal					
0.95	S	0.956	0.929	0.912	0.948
	P	0.974	0.952	0.943	0.951
0.90	S	0.909	0.878	0.863	0.897
	P	0.941	0.900	0.886	0.901
0.85	S	0.868	0.830	0.812	0.847
	P	0.906	0.846	0.833	0.851
Normal Mixture					
0.95	S	0.957	0.936	0.926	0.950
	P	0.975	0.938	0.935	0.951
0.90	S	0.923	0.892	0.875	0.899
	P	0.941	0.878	0.872	0.901
0.85	S	0.879	0.843	0.824	0.852
	P	0.901	0.829	0.822	0.851
Heteroscedastic Normal					
0.95	S	0.963	0.950	0.946	0.937
	P	0.966	0.948	0.943	0.951
0.90	S	0.922	0.906	0.896	0.895
	P	0.925	0.898	0.887	0.901
0.85	S	0.887	0.859	0.851	0.846
	P	0.868	0.838	0.832	0.851

Note: The model includes three regressors, a constant and two other, the real vector of coefficient is $(1, 1, 1)$*, and the censoring point here is* $y_0 = 0$*. P denotes percentile. S denotes the standard. VNS denotes variable neighbourhood search. ECP is the empirical coverage probabilities.*

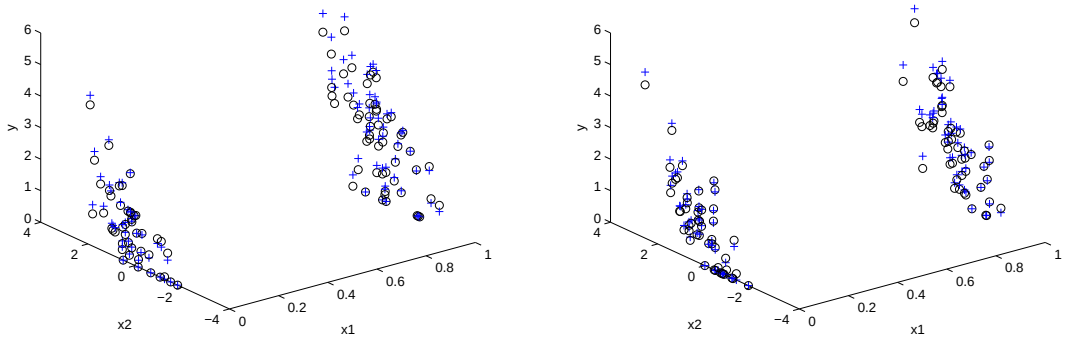


Figure 3.6: Points (x_{1i}, x_{2i}, y_i), $i = 1, \ldots, 100$, in data space with the standard normal and normal mixture errors, and their estimated values (denoted as "o"), obtained by CQR-VNS (denoted as "+")

The bootstrap regression model in Table 3.1 can be formulated as in Hahn (1995),

$$f(\beta) = \min \sum_{i=1}^{n} \rho_\theta (y_i - \beta_{\theta i} x_i') \tag{3.16}$$

where $y_i = \beta_{\theta i} x_i' + \varepsilon_{\theta i}$. Moreover, the regressor vector x is deterministic and $\varepsilon_{\theta i}$ are i.i.d of random variables.

The M-bootstrap and the resampling method in Bilias et al. (2000) are an extension to the PWY method (Parzen et al., 1994), where they are based on the next equation,

$$f(\hat{\beta}) = \sum_{i=1}^{n} x_i [I(y_i - \beta_{\theta i} x_i' \leq 0) - \frac{1}{2}] I(\hat{\beta_{\theta i}} x_i' > 0) + U^* = 0 \tag{3.17}$$

where $U^* = \sum_{i=1}^{n} x_i [B_i - \frac{1}{2}] I(\hat{\beta_{\theta i}} x_i' > 0)$. $B_i, i = 1, \ldots, n$ is a sequence of i.i.d. Bernoulli random variables with success probability 1/2.

Moreover, the steps of calculating the standard S in Table 3.1 are given by:

- We run the code for 1000 times for each type of error, and we then save the regression coefficient β_2.
- Then the mean and the standard deviation (sd) of 1000 β_2 has been calculated.

- The confidence interval for the quantile θ is given by $(mean(\beta_2) - \theta.sd(\beta_2), mean(\beta_2) + \theta.sd(\beta_2))$, where θ in Table 3.1 has three probabilities $\theta = 0.95, 0.90$ or 0.85, where $\theta(0.95) = 1.96, \theta(0.90) = 1.64$ and $\theta(0.85) = 1.44$.
- The standard S is the number of β_2 within the confidence interval divided by 1000.

Furthermore, the steps of calculating the percentile P in Table 3.1 are given by

- We run the code for 1000 times for each type of error, and we then save the regression coefficient $\beta 2$.
- The percentile interval is written as $(\frac{(1-\theta)}{2} * 100, (1 - \frac{(1-\theta)}{2}) * 100)$, where θ in Table 3.1 has three probabilities $\theta = 0.95, 0.90$ or 0.85.
- The percentile P is the number of β_2 within the percentile interval divided by 1000.

As we can see in Table 3.1, for standard normal distribution error and normal mixture error, VNS method gives better results when compared to other methods. For heteroscedastic normal error term, our `CQR-VNS` reports better results than others for finding the percentile "P", but it is not the best one in finding standard "S" case. The distribution best solutions obtained by our `CQR-VNS` in 100 runs are presented at Figure 3.7. Therefore, we can conclude that VNS based heuristic with the Powell's estimator is a new promising method for solving the CQR problem. Our results also show that the choice of approximate solution method applied on exact model could be a better choice than the use of exact methods on an approximate model.

3.4.2 `GLOB-VNS` for finding Finding root mean square, mean bias, mean absolute deviation and median bias

In this subsection, there are two possible regression functions. The first one can be written as

$$y = \max\{\beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + \varepsilon_i, y_0\}, \tag{3.18}$$

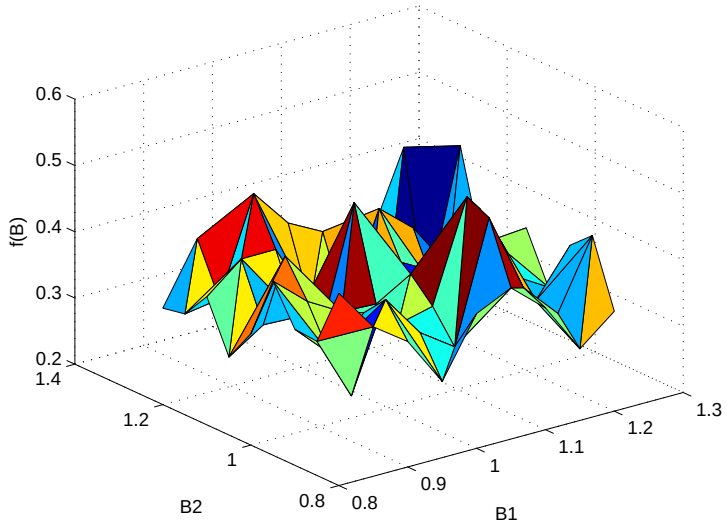


Figure 3.7: Distribution of local minima in (β_1, β_2) space, obtained by 100 restart of `CQR-VNS`

and the second one can be written as

$$y = \max\{\beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + x_{i3}\beta_3 + x_{i4}\beta_4 + x_{i5}\beta_5 + \varepsilon_i, y_0\}, \tag{3.19}$$

where $(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$ equals $(1, 1, 0.5, -1, -0.5, 0.25)$. x_i are generated as a standard normal distribution, truncated as $\{\| \ x_i \ \|_\infty < 2\}$. The error term has the multiplicative herteroscedasticity structure, where it can be formulated as

$$\varepsilon_i = u_i v(x_i), \tag{3.20}$$

where $v(x_i)$ can be written as

$$v(x_i) = a_0 + \sum_{j=1}^{m} (a_{j1} x_{ji} + a_{j2} x_{ji}^2), \tag{3.21}$$

$a_0 = 1$, $a_{j1} = 0.5$ and $a_{j2} = 0.5$. The censoring point is $y_0 = -0.75$.

Two alternative distributions are considered for u_i: a normal distribution $N(0, 25)$, and a χ^2 distribution with four degrees of freedom, re-centered to have zero median. In this part we have done the following:

- We generate the data as above according to Buchinsky and Hahn (1998).
- We repeat the simulation 10000 times for each of the three cases of sample size: $\{(y_i, x_i), i = 1, 2, \ldots, 100\}$,$\{(y_i, x_i), i = 1, 2, \ldots, 400\}$, and $\{(y_i, x_i), i = 1, 2, \ldots, 600\}$.
- We apply the VNS for two cases of regression function. the first one as (3.18), and the second one as (3.19).
- We find the root mean square errors (RMSEs), mean bias, mean absolute deviation (MAE), and median bias for all (β_1, β_2).
- Our results are compared with the results from the CV method (which denotes the CQR-LP estimator with log likelihood cross-validated bandwidth), the CVa method (which denotes the CV estimator with bandwidth adjusted to conform with assumption K), the PR method (which denotes the CQR-LP estimator with probit estimates for the censoring probability), the HO and HOa methods (which are the same as CV and CVa, except that the kernel function is order kernel), the CR method, which is a Powell's estimator, and the VNS method (Buchinsky and Hahn, 1998).

Table 3.2 shows the computational results, when the function is presented as in (3.18)

Table 3.2: Monte carlo simulation with three regressors for 0.50 quantile and 0.75 censoring point (10,000) repetition

	Intercept - β_0							Slope - β_2						
	CV	CVa	PR	HO	HOa	CR	VNS	CV	CVa	PR	HO	HOa	CR	VNS
							$N(0,25)$							
$n = 100$														
$RMSE$	2.88	3.34	1.59	4.39	3.02	4.11	0.42	2.16	2.10	2.18	2.41	1.98	2.85	0.73
$Mean\ bias$	0.14	0.07	0.60	0.18	0.44	−0.08	−0.23	0.31	0.28	0.70	0.32	0.40	0.33	−0.65
MAE	0.82	0.80	0.94	0.83	0.92	0.74	0.23	0.86	0.83	1.21	0.85	0.90	0.92	0.65
$Median\ bias$	0.93	0.32	0.70	0.45	0.60	0.35	−0.00	0.06	0.01	0.49	0.05	0.13	−0.31	−0.69
$n = 400$														
$RMSE$	0.58	0.57	0.56	0.60	0.66	0.68	0.28	0.61	0.59	0.90	0.67	0.71	0.66	0.65
$Mean\ bias$	0.20	0.17	0.17	0.19	0.30	0.19	−0.14	−0.06	−0.10	0.28	−0.05	0.04	−0.45	−0.61
MAE	0.39	0.38	0.36	0.41	0.45	0.41	0.14	0.39	0.39	0.53	0.43	0.44	0.55	0.61
$Median\ bias$	0.20	0.16	0.16	0.19	0.29	0.19	−0.00	−0.12	−0.17	0.21	−0.12	−0.05	−0.52	−0.57
$n = 600$														
$RMSE$	0.48	0.48	0.46	0.48	0.52	0.49	0.24	0.50	0.48	0.71	0.54	0.56	0.57	0.62
$Mean\ bias$	0.18	0.16	0.14	0.12	0.20	0.20	−0.12	−0.06	−0.11	0.25	−0.17	−0.10	−0.47	−0.58
MAE	0.33	0.32	0.31	0.33	0.35	0.33	0.12	0.33	0.33	0.44	0.40	0.40	0.50	0.58
$Median\ bias$	0.18	0.16	0.15	0.13	0.20	0.19	−0.00	−0.10	−0.15	0.23	−0.22	−0.16	−0.49	−0.54
							$\chi^2(4)$							
$n = 100$														
$RMSE$	0.63	0.62	0.62	0.65	0.70	0.67	0.25	0.72	0.70	0.92	0.76	0.79	0.90	0.59
$Meanbias$	0.26	0.23	0.25	0.30	0.37	0.20	−0.12	−0.05	−0.08	0.19	−0.01	0.04	−0.14	−0.52
MAE	0.41	0.40	0.39	0.43	0.45	0.39	0.12	0.46	0.45	0.54	0.48	0.49	0.56	0.52
$Median\ bias$	0.24	0.21	0.22	0.27	0.33	0.19	−0.00	−0.11	−0.13	0.12	−0.08	−0.03	−0.37	−0.49
$n = 400$														
$RMSE$	0.32	0.31	0.28	0.32	0.35	0.31	0.13	0.35	0.35	0.39	0.37	0.37	0.51	0.50
$Mean\ bias$	0.17	0.16	0.09	0.17	0.21	0.12	−0.06	−0.11	−0.14	0.00	−0.09	−0.07	−0.45	−0.47
MAE	0.22	0.21	0.18	0.22	0.23	0.20	0.06	0.24	0.24	0.25	0.25	0.25	0.48	0.47
$Median\ bias$	0.17	0.16	0.08	0.17	0.20	0.11	−0.00	−0.13	−0.15	−0.02	−0.11	−0.08	−0.47	−0.45
$n = 600$														
$RMSE$	0.26	0.265	0.23	0.27	0.29	0.25	0.10	0.30	0.30	0.32	0.30	0.30	0.50	0.48
$Mean\ bias$	0.14	0.14	0.06	0.15	0.18	0.09	−0.04	−0.10	−0.13	−0.01	−0.09	−0.07	−0.46	−0.46
MAE	0.18	0.18	0.15	0.18	0.19	0.17	0.04	0.21	0.21	0.22	0.21	0.21	0.48	0.46
$Median\ bias$	0.14	0.13	0.06	0.14	0.17	0.10	−0.00	−0.12	−0.14	−0.03	−0.10	−0.08	−0.48	−0.44

Note:The model includes three regressors: a constant and two random i.i.d. N(0,1)regressors. The vector of coefficients is (1,1,.5). The censoring point is set at 0.75.

The steps of calculating $RMSE$, $Mean\ bias$, MAE and $Median\ bias$ in Table 3.2 are given by

- We run the code for 10,000 times for each type of error and we then save the constant and slop regression coefficient β_0 and β_1.
- We find the mean and the variance for β_0 and β_1 independently.
- The $RMSE$ has been found by the equation

$$RMSE = \sqrt{(mean - \beta)^2 + variance} \tag{3.22}$$

  where β is equal 1.
- The $Mean\ bias$ is given $Mean\ bias = Mean - \beta$, where $\beta = 1$.
- The $Median\ bias$ is given $Median\ bias = Median - \beta$, where $\beta = 1$.
- The MAE is calculated by $MAE = \frac{1}{n}\sum_{i=1}^{n} \mid mean - \beta_{0i} \mid$ for β_0 and $MAE = \frac{1}{n}\sum_{i=1}^{n} \mid mean - \beta_{1i} \mid$ for β_1, where $n = 10,000$.

We use in Table 3.2 the regression function as in (3.18). Applying VNS method to original Powell estimator for solving CQR outperforms better than the six other methods for the regression coefficient β_1 for all sample sizes and both kinds of error. On the other hand, the VNS method did not work very well for the regression coefficient β_2. The explanation is that the VNS works to give the minimum of the objective function in total, not the minimum of each component of this objective function.

In Table 3.3 the regression function has five regressors, as in (3.19). The data are generated in the same way as in Table 3.2.

Table 3.3: Monte carlo simulation with six regressors for 0.50 quantile and 0.75 censoring point (10,000) repetition

	Intercept					Slope				
	CV	CVa	PR	CR	VNS	CV	CVa	PR	CR	VNS
					$N(0,25)$					
$n = 100$										
$RMSE$	3.29	3.25	4.36	3.93	0.52	2.24	2.24	2.74	3.05	0.61
$Meanbias$	1.66	1.38	3.59	0.70	−0.30	0.26	0.21	0.72	0.43	−0.41
MAE	2.07	1.89	3.56	1.55	0.30	1.12	1.09	1.60	1.11	0.41
$M - bias$	1.77	1.54	3.53	1.22	−0.00	0.05	0.00	0.51	−0.12	−0.13
$n = 400$										
$RMSE$	1.34	1.28	1.46	1.31	0.42	0.78	0.77	1.15	0.89	0.52
$Meanbias$	0.78	0.62	1.15	0.74	−0.23	−0.19	−0.24	0.26	−0.38	−0.32
MAE	0.89	0.80	1.17	0.81	0.23	0.52	0.51	0.69	0.66	0.32
$M - bias$	0.79	0.64	1.15	0.78	−0.00	−0.25	−0.30	0.19	−0.54	−0.01
$n = 600$										
$RMSE$	1.11	1.01	1.07	0.91	0.38	0.63	0.63	0.88	0.71	0.48
$Meanbias$	0.67	0.54	0.81	0.64	−0.20	−0.14	−0.16	0.16	−0.50	−0.29
MAE	0.74	0.66	0.84	0.65	0.20	0.45	0.45	0.54	0.63	0.29
$M - bias$	0.67	0.56	0.81	0.62	−0.00	−0.20	−0.22	0.11	−0.60	−0.00
					$\chi^2(4)$					
$n = 100$										
$RMSE$	1.53	1.46	2.01	1.33	0.38	1.02	1.03	1.34	1.23	0.51
$Meanbias$	0.96	0.87	1.63	0.69	−0.20	−0.22	−0.24	0.10	−0.12	−0.32
MAE	0.99	0.93	1.54	0.75	0.20	0.66	0.64	0.80	0.73	0.32
$M - bias$	0.93	0.84	1.53	0.68	−0.00	−0.30	−0.34	−0.02	−0.37	−0.00
$n = 400$										
$RMSE$	0.88	0.80	0.69	0.62	0.19	0.51	0.52	0.57	0.60	0.36
$Meanbias$	0.72	0.62	0.51	0.46	−0.07	−0.24	−0.29	−0.04	−0.47	−0.20
MAE	0.70	0.61	0.52	0.46	0.07	0.37	0.38	0.38	0.54	0.20
$M - bias$	0.70	0.60	0.50	0.45	−0.00	−0.27	−0.31	−0.07	−0.52	−0.00
$n = 600$										
$RMSE$	0.80	0.72	0.54	0.54	0.13	0.42	0.43	0.45	0.57	0.31
$Meanbias$	0.69	0.60	0.40	0.42	−0.04	−0.17	−0.19	−0.04	−0.50	−0.17
MAE	0.69	0.60	0.41	0.41	0.04	0.29	0.31	0.30	0.53	0.17
$M - bias$	0.69	0.60	0.40	−0.00	−0.65	−0.18	−0.20	−0.06	−0.53	−0.00

Note:The model includes six regressors: a constant and five random i.i.d. N(0,1)regressors. The vector of coefficients is (1,1,.5,-1,-.5,.25). The censoring point is set at 0.75.

Table 3.3 shows that the VNS method with Powell estimator in general for solving CQR gives very good results in the case of five regressors. We note that VNS is the best for all sample sizes in both kind of error for slope coefficient. In the case of the x_2 coefficient VNS works well, better than other methods most of the time. We may conclude that, when we increase the dimension of the regression function, the VNS method with the exact Powell estimator outperforms other methods from the literature.

3.5 Conclusion and future research

In this chapter we suggest a new method for solving censored quantile regression (CQR). It is based on variable neighbourhood search (VNS) global optimisation technique. As the objective function it uses Powell estimator, which is known to be non convex, nor concave. Other approaches in the literature try to find a linear approximation of Powell's function and then solve the problem exactly. We rather apply an approximation method on an exact CQR model. Our method adapts VNS rules in order to solve this global optimisation problem. The basic idea of the VNS metaheuristic is the use of different metric functions in defining the neighbourhood structure of the current solution.

It appears that our new approach is competitive with state-of-the-art methods from the literature. Moreover, our results indicate that better solutions are usually obtained by using a nonlinear model and effective approximate solution method that use an approximate (linear) model with exact solution procedure.

Future research may include the use of other, more sophisticated global optimisation techniques for solving CQR. Moreover, it may include the extension of our approach to semi censored quantile regression (Powell, 1986) as well. In addition, new neigbourhood structures may be tried out within variable neigbourhood approach.

Chapter 4

Circle packing problem

In this chapter, variable neighbourhood search (VNS), a nonlinear global optimiser for solving circle packing problem (CPP) is studied. The problem is an optimisation arrangement of n arbitrary sized circles inside a container (e.g. a circle, a square or a rectangle) such that no two circles overlap. Several years ago, CPP has been formulated as a continuous, nonlinear, nonconvex global optimisation problem. This problem has proven to be NP-hard (Hochbaum and Maass, 1985). Recently it has been solved by the reformulation descent approach where two different formulations of the problem switch between a polar and a Cartesian system (Mladenović et al., 2005).

This chapter presents a VNS algorithm based on reformulation descent by using two different Cartesian formulations, maximising the radius of small circles or minimising the radius of the container. Moreover, we consider two types of containers to pack n equal circles: a circle and a square. The problem has been solved by each formulation independently as well as by using both within one reformulation descent method. Our work investigates the effect of this type of reformulation.

This chapter is organised as follows. In section 4.1, the description of the circle packing problem is presented. As mentioned earlier, the CPP has various type of containers, in this thesis CPP is studied within two types of containers, a circle and a square. Furthermore, a brief idea of the CPP within the circle and the square container is given in section 4.2. Section 4.3 is devoted to focus on the variable neighbourhood search for solving the circle

packing problem with reformulation descent. The reformulation descent has used two Cartesian formulations for each container type. The computational results of the algorithm are presented in section 4.4. Section 4.5 gives a brief conclusion with future research.

4.1 Problem description

The circle packing problem (CPP) consists of positioning a given number n of circular disks of equal radius without overlap, such that its radius is a maximum. It is a NP-hard problem, i.e. it is difficult to find the optimal solution and therefore it is unlikely to find a polynomial time algorithm to solve the problem. Moreover, it is harder to be solved, when the search space becomes large. Thus, the use of heuristic methods provides good solutions with more speed, efficiency, and reliability. In a general setting, the CPP can be defined as follows.
If there is a given container which depends on a size parameter r, where $C(r) \subset R^d$, and given n are geometrical objects. The position of these n in d-dimensional space depends on t position parameters $\alpha_{i1}, \ldots, \alpha_{it}$, i.e., $D_i(\alpha_{i1}, \ldots, \alpha_{it}) \subset R^d, i = 1, \ldots, n$. Then the solution is determined by choosing the parameters where all the objects are packed inside a container such that no two circles overlap, and the size of container is minimised. The problem can be written

$$\min \ r$$

s.t.

$$D_i(\alpha_{i1}, \ldots, \alpha_{it}) \subseteq C(r) \qquad i = 1, \ldots, n$$

$$D_i^0(\alpha_{i1}, \ldots, \alpha_{it}) \cap D_j^0(\alpha_{i1}, \ldots, \alpha_{it}) = \phi \quad i \neq j$$

If d is equal to 2, many optional features for the container could exist like a circle, a square, a rectangle or a strip. Moreover, the radius of circles within the container could be equal or unequal.

The CPP has arisen in many fields of natural sciences such as engineering design and has many applications such as in coverage, storage, packaging and cutting industry (Correia

et al., 2001; Cui, 2005; Dowsland, 1991; Fraser and George, 1994; George et al., 1995; Hifi and M'Hallah, 2004; Hifi et al., 2004). Hifi et al. (2004) consider cutting out as many circles as possible from the rectangle plate, where these problems could be considered as constrained or unconstrained circle cutting problems. Cui (2005) gives optimal patterns for solving the problem of cutting circle blanks from silicon steel to build electric motors. These types could be expensive due to the use of silicone. The quality of the packing is measured by one of these three options (Castillo et al., 2008):

- the size of the container.
- the weighted average pairwise distance between centers of circles.
- a linear combination of both criteria.

The geometric approach to the circle packing problems has a long history (see Szabó et al. (2007) for an overview). However, solving problems with a large n is almost impossible by using geometrical arguments. Therefore, most researchers switch to mathematical programming approaches.

The same problem can sometimes be formulated in different ways, and one can switch between formulations using the same local optimisation method. A solution obtained by one formulation is an initial solution for another one. The method that alternates between different formulations of the problem is called a formulation space search (FSS). Mladenović et al. (2005) is the simplest FSS, where formulations are changed in a deterministic way until there is no improvement in the objective function value. CPP can be manipulated through different formulations according to container types. CPP is solved by using a nonlinear reformulation of the problem, where the idea of reformulation descent (RD) has been applied. Switching between the Cartesian and the polar coordinates (and vice versa) was possible since MINOS is used as an NLP solver (see details in Mladenović et al. 2005). Their strategy avoids stopping at a stationary point by switching between two different formulations. An extension of the RD idea is discussed by Mladenović et al. (2007).

As previously mentioned, the CPP is considered for packing equal or unequal circles within different types of containers. Each type could be manipulated by using different

formulations. In this chapter, the CPP has been considered within two types of containers in the plane (i.e. $d = 2$), a circle and a square. Each type of container will accommodate equal circles only with their mathematical formulations.

4.1.1 Circle packing problem inside a circle container (CPP-1)

There are two different formulations of CPP-1. The first formulation attempts to find a maximum radius for n equal circles in a unit circle container without any overlap. We assume that the container is centered at the origin. This problem can be written in $2-dimensional$ Cartesian as follows:

$$\max \ r \tag{4.1}$$

s.t.

$$\sqrt{x_i^2 + y_i^2} \leq 1 - r, \qquad i = 1, \ldots, n,$$

$$d_{ij} \geq 2r, \qquad i, j = 1, \ldots, n \ ; i < j$$

where the center of circle i is denoted by (x_i, y_i). d_{ij} is the distance between the centers of the two circles, i and j, where $(i < j)$. Moreover, it can be calculated by using Euclidean distance as follow:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \qquad \forall i, j = 1, \ldots, n, \tag{4.2}$$

The second formulation attempts to minimise the radius of the circle container R to accommodate n unit circles without overlap. We assume that the problem is in $2-dimensional$ Cartesian as well, where it is given as:

$$\min \ R \tag{4.3}$$

s.t.

$$\sqrt{X_i^2 + Y_i^2} \leq R - 1, \qquad i = 1, \ldots, n,$$

$$d_{ij} \geq 2, \qquad i, j = 1, \ldots, n \ ; i < j$$

To prove the equivalence between the two formulations (4.1) and (4.3), we divide the constraints in formulation (4.1) by $1/r^2$. For the first constraint we have

$$x_i^2 + y_i^2 \leq (1-r)^2, \qquad i = 1, \ldots, n,$$

by dividing this constraint with $1/r^2$, we will have

$$\frac{x_i^2}{r^2} + \frac{y_i^2}{r^2} \leq (\frac{1-r}{r})^2$$

then we will get

$$(\frac{x_i}{r})^2 + (\frac{y_i}{r})^2 \leq (\frac{1}{r} - 1)^2$$

this gives

$$\sqrt{(\frac{x_i}{r})^2 + (\frac{y_i}{r})^2} \leq \frac{1}{r} - 1$$

where $\frac{x_i}{r} = X_i$, $\frac{y_i}{r} = Y_i$ and $\frac{1}{r} = R$, and this gives the first constraint in the formulation (4.3).

For the second constraint, we will do the same

$$d_{ij}^2 \geq 4r^2, \qquad i, j = 1, \ldots, n \; ; i < j$$

this gives

$$\frac{(x_i - x_j)^2 + (y_i - y_j)^2}{r^2} \geq \frac{4r^2}{r^2}$$

where

$$(\frac{x_i}{r} - \frac{x_j}{r})^2 + (\frac{y_i}{r} - \frac{y_j}{r})^2 \geq 4$$

where this gives the second constraint in formulation (4.3).

4.1.2 Circle packing problem inside a square container (CPP-2)

In this subsection, CPP-2 formulations inside a square container will be explained. Once again two formulations are presented. The first one is maximising the radius r for n equal circles inside a unit square container without any overlap. It can be formulated as

$$\max \; r \tag{4.4}$$

s.t.

$$r \leq x_i \leq 1 - r, \qquad i = 1, \ldots, n,$$

$$r \leq y_i \leq 1 - r, \qquad i = 1, \ldots, n,$$

$$d_{ij} \geq 2r, \qquad i, j = 1, \ldots, n \; ; i < j$$

where d_{ij} is given by (4.2). The second one is minimising the length of the side container L of the square to accommodate a number of n unit circles without overlap. The nonlinear formulation can be written as

$$\min \; L \tag{4.5}$$

s.t.

$$1 \leq X_i \leq L - 1, \qquad i = 1, \ldots, n,$$

$$1 \leq Y_i \leq L - 1, \qquad i = 1, \ldots, n,$$

$$d_{ij} \geq 2, \qquad i, j = 1, \ldots, n \; ; i < j$$

We can note that the optimal solution in formulation (4.4) can be transferred to the other formulation (4.5) by using $X_i = \frac{x_i}{r}$, $Y_i = \frac{y_i}{r}$ and $L = \frac{1}{r}$. This can be proved the same as in circle packing problem within a circle container.

4.2 Literature review

Historically, the CPP was introduced in European mathematics in the nineteen century, when the Italian mathematician G. Malfatti posed this question "*Consider a right prism with a right triangular base. How do we cut out three cylinders (perhaps of different sizes) from the prism, such that the total volume of cylinders is maximal?*" (Szabó et al., 2007). This problem was studied and solved earlier by the Japanese mathematician Chokuen Ajima between (1732-1798) (Szabó et al., 2007).

The packing problems have been studied for $2-$*dimensional* and $3-$*dimensional* space. Only the $2-$*dimensional* space will be considered here. Details of packing problems in $3-$*dimensional* space have been published by many authors (Kravitz 1967; Clare and

Kepert Jun. 9, 1986; Hsiang 1993). The circle packing problems in $2-dimensional$ space may have various types of containers such as a circle, a square, a rectangle, a triangle or a strip. The next subsections will give a brief review of CPP within a circle and a square container respectively. For other types of container the reader may be refereed to (Dowsland and Dowsland, 1992; George et al., 1995; Huang et al., 2005; Birgin et al., 2005; Birgin and Gentil, 2010).

4.2.1 Circle packing problem within the circle container CPP-1

CPP-1 has two options of packing n non-overlapping circles, equal or unequal ones. This thesis considers only packing n equal circles within a circle container without overlap. Details of packing unequal circles inside a circle container without overlap have been discussed by many researchers (Wenqi and Ruchu, 1999; Wang et al., 2002; Huang et al., 2003; Hifi and M'Hallah, 2008; Huang et al., 2006; Grosso et al., 2010).

Kravitz (1967) is perhaps one of the first researchers to study uniform sized circles inside a unit circle container. He was able to pack up to 19 without proofs of optimality. He thought that the global optima for $n = 10$ within the larger container is found with $R = 3.8284271$. Indeed, the global optimal solution for $n = 10$ has been found later by mathematical programming with $R = 3.8130256$, see Figure 4.1.

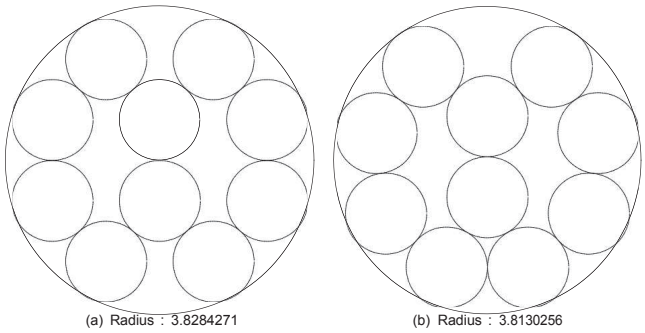


Figure 4.1: Packing 10 unit circles into a circle

The optimal solutions for $n = 7, \ldots, 10$ were independently proven by Graham (1968) and Pirl (1969). Pirl (1969) also improved the results of packing problem for n up to 19. However, some of his results ($n = 14, 15, 16$) were improved later (Goldberg, 1971). Goldberg (1971) showed the packing results for n up to 20, but his results of $n = 17$ were improved by Reis (1975).

Reis (1975) improved the results of packing equal circles within a circle container for $n = 17, 21, \ldots, 25$ as compared with previous literature. This involved a new technique by using an iris diaphragm as a variable circumscribing circle to eliminate any assumption connected with symmetry (Reis, 1975). The optimal solutions for n up to 11 have also been published in Melissen (1994).

Lubachevsky and Graham (1997) solve CPP-1 by using the curved hexagonal packings. For each $k \geq 1$, the corresponding hexagonal number is $h(k) = 3k(k + 1) + 1$, where $n = h(k)$ is the number of equal circles within the container. There are $m(k) = \max\{(k - 1)!/2, 1\}$ different curved hexagonal packings, they give the same density. Billiards simulation algorithm is used to solve the problem, which works by simulating the movements of n circles during their movements along a fixed direction, each circle collides with other circles and the boundary of the circle container. They assume that there is no gravity and friction. For $k \leq 5$, the optimal results are found, whereas for $k > 5$ good quality solutions have been found by curved hexagonal packings (Lubachevsky and Graham, 1997).

Graham et al. (1998) use two packing algorithms to find the minimum radius for circle container to accommodate n unit circles. They note that there research is poor for $n > 20$. They use two variant strategies. One is based on the repulsion forces algorithm and it is similar to that described by Nurmela and Ostergard (1997). The other is based on Billiards simulation algorithm, which has been used by Lubachevsky and Graham (1997). Both algorithms attempt to maximise the minimum pairwise distance among n points spread in the unit circle centered at the origin. Moreover, both algorithms need similar CPU time. Whereas billiard simulation algorithm is better in finding the optimal solutions, the algorithms find the best packing results for $21 \leq n \leq 65$.

As mentioned before, the same problem can sometimes be formulated in different ways,

and one can switch between formulations within the same method. A solution obtained by one formulation is an initial solution for another. The method that alternates between two different formulations of the problem is called a *reformulation descent* (RD) (Mladenović et al., 2005). The CPP problem can be formulated by different formulations due to container types. In Mladenović et al. (2005), the CPP is solved by using a nonlinear reformulation of the problem, where the idea of RD has been applied. Switching between Cartesian and polar coordinates (and vice versa) was possible since MINOS is used as an NLP solver (see details in Mladenović et al. (2005)). Their strategy avoids stopping at a stationary point by switching between two formulations. Their results are compared with a Netwon-type solution approach, where two approaches find solutions of good quality. However, RD needs less CPU time. Extension of the RD idea by using more than two formulations is called *formulation space search* (FFS) and it was put forward by (Mladenović et al., 2007).

Zhang and Deng (2005) use a hybrid algorithm for solving CPP-1 into a larger container circle. The hybrid algorithm combines simulated annealing with tabu search. The power of this algorithm comes from getting out of local minima. It uses simulated annealing to escape from local optima and tabu search to prevent cycling and enhance diversification. The hybrid algorithm outperforms simulated annealing and tabu search algorithms (Zhang and Deng, 2005).

Birgin and Sobral (2008) solve CPP-1 in $2-dimensional$ and $3-dimensional$ space within various types of containers (a circle, a triangle, a square, a rectangle and a strip). Each type of container accommodates equal or different-sized circles. They use twice-differentiable models for all pervious cases. Their strategy reduces the computational cost of computing the overlapping (Birgin and Sobral, 2008).

Lu and Huang (2008) incorporate the PERM scheme into the strategy of maximum cave degree to solve equal or unequal circles within a larger container circle. This approach evaluates the benefit of a partial configuration by using the maximum cave degree. Besides, the PERM strategy enhances the efficiency of search. Zhang's algorithm is more powerful than Lu and Huang's algorithm for several large instances with equal size circles (Lu and Huang, 2008).

4.2.2 Circle packing problem within the square container CPP-2

Several approaches have been developed to solve the circle packing problem within a square container during the last five decades. In a theoretical way, the optimal solutions are proven for $n = 2, \ldots, 9, 14, 16, 25$ and 36 circles (Nurmela and Ostergard, 1997; Markót, 2004), $n \leq 20$ (de Groot et al., 1990, 1992). The optimal solutions for $21 \leq n \leq 27$ can be proven using computer-aided software (Nurmela and Ostergard, 1999). Recently, the function values for $n = 10, \ldots, 35, 37, 38$ circles were correct within the tolerance equal to $1e-5$ by Locatelli and Raber (2002).

Maranas et al. (1995) use a min max optimisation approach with the MINOS and the GAMS modeling language. This approach improves the result of $n = 15$, besides new configurations are found for n= 28 and 29 and it matches the best results for up to $n = 30$.

Nurmela and Ostergard (1997) solve the problem by using a different nonlinear approach based on the energy function $\sum_{i \neq j} (\lambda / d_{ij}^2)^m$. The problem is solved by minimising the energy function, where it is transformed into an unconstrainted problem and is solved by using a multistart hybrid line-search algorithm. The algorithm starts with a gradient direction, and close to a solution it then uses a Newton direction (Nurmela and Ostergard, 1997). This algorithm is used for solving the problem when $n \leq 50$. Some results improve and others find alternative ones. Furthermore, Graham and Lubachevsky (1996) extend and improve the work carried out by Nurmela and Ostergard (1997) by using a Billiard simulation.

Boll et al. (2000) solve the problem by using a two-phase approach, the approximation one moves each point along a pre-chosen direction while decreasing the step size. The second phase uses a Billiard simulation, where the starting point is the result of the first phase. The results of $n = 32$ and 48 circles are slightly improved, whereas the big difference is seen for $n = 37$ and 50.

In Casado et al. (2001) , the unit square is subdivided into $k \times k$ subsquares, where $k = \mid \sqrt{n} \mid$. They find the initial solution by randomly placing n points at the center of the n subsquares. Each point has been perturbed randomly, where the algorithm may accept the nonimproved perturbed point. The algorithm finds the results up to $n = 100$, where it can find most of the optimal solutions from the literature and improve some results. This

idea is extended by other researchers (Markót and Csendes, 2005; Szabó et al., 2005), where they eliminate large groups of subproblems to improve the solution. The algorithm solves the problem for $n = 28, 29$ and 30 within tight tolerance values (Markót and Csendes, 2005; Szabó et al., 2005). Further extensions of this idea has been documented by Markót and Csendes (2006); Markót (2007).

Locatelli and Raber (2002) consider finding the maximum radius for n non-overlapping circles within a unit square. Their approach starts from a general rectangle branch-and-bound algorithm, where the problem is modeled as a quadratic optimisation problem. Within the tolerance 10^{-5}, the algorithm has proven optimal for best known solutions in the literature for $n = 10, \ldots, 35$ and $n = 38, 39$ except $n = 32$. In the case of $n = 32$, a new solution has been found with a proof of its optimality within the given tolerance. However, for $n = 37$ a new solution has been detected without a proof of optimality within the given tolerance. The running time for $n \leq 13$ is less than 0.5 seconds, for $n \leq 21$ is more than 2 minutes, and for $n \leq 26$ is more than 30 minutes, whereas for $n \leq 28$ exceeds the 27 hours (Locatelli and Raber, 2002).

Addis et al. (2008) reformulate the problem to have more efficient local search procedures. They change the Euclidean distance by its square, where it can be written as

$$\bar{d}^2 \leq (x_i^{'} - x_j^{'})^2 + (y_i^{'} - y_j^{'})^2 \qquad 1 \leq i < j \leq n, \tag{4.6}$$

They conjecture that the problem possesses a funnel landscape. This feature is commonly known in molecular conformation problems. Their algorithm improves 32 best known solutions in the range $n \leq 130$ (Addis et al., 2008).

van Dam et al. (2007) solve the problem by using maximin Latin hypercube designs for general n. They find the maximum Latin hypercube designs for $n \leq 70$ and the approximation maximum Latin hypercube designs for $71 \leq n \leq 1000$ (van Dam et al., 2007).

4.3 Reformulation descent within variable neighbourhood search for solving circle packing problem

In this section, the rules for solving the CPP by using `RD-VNS` will be explained.

4.3.1 RD-VNS for CPP

In this subsection, we have applied VNS in four different ways: using formulation in (4.1) or (4.4) only, then using formulations in (4.3) or (4.5). Our `RD-VNS` switches between the first and the second formulation, i.e. between (4.1) and (4.3), or between (4.4) and (4.5), and vice versa. The CPP objective function is a nonlinear continuous one. Thus, the CPP problem may be solved as an unconstrained nonlinear problem. Observe also that any unconstrained nonlinear program may be considered as a box constrained, if left and right values of variables that define the box are set to the same large negative and positive values a_i and b_i. The `GLOB-VNS` package (Drazić et al., 2006) has been used. In order to run VNS global optimiser for solving CPP, we need to adapt CPP variables to that general solver. Therefore, given input data in `GLOB-VNS` is the vector x, and it contains $2n + 1$ variables denoted by $x = (x_1, y_1, \ldots, x_n, y_n, r)$, for models in (4.1) and (4.4). In the models given by (4.3) and (4.5), the last variables are R and L respectively.

The steps of the VNS heuristic (`Glob-VNS`) for solving CPP for one formulation only are given in Algorithm 25:

Function `Glob-VNS` (x, k_{max}, t_{max})

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$ induced from ℓ_1 and ℓ_∞ norms.
2 Choose (*geometry*, *distribution*) pairs order.
3 Find an initial point $x \in X$.
4 **while** $t < t_{max}$ **do**
5 $k \leftarrow 1$
6 **repeat**
7 **for** *(geometry, distribution) pairs order* **do**
8 Generate $x' \in \mathcal{N}_k(x)$ at random //Shaking
9 Apply Hooke-Jeeves NLP method starting with x' to get x'' //Local search
10 **if** $f(x'') < f(x)$ **then**
11 $x \leftarrow x''$, `go to line 4`
  else
12 Set $k \leftarrow k + 1$
  until $k = k_{\max}$
13 $t \leftarrow$ `CpuTime()`
14 return x

Algorithm 25: VNS for one formulation only

where k_{max} is a maximum number of neighbourhoods used in the search and t_{max} is the maximum time allowed in the search.

Neighbourhoods - Shaking. For solving GOP, VNS has already been used in two different ways: with neighbourhoods induced by using an ℓ_p norm (Drazić et al., 2006; Mladenović et al., 2008; Liberti and Drazic, 2005) and without using an ℓ_p norm (Toksari and Guner, 2007). Here we apply VNS that uses the ℓ_p norm, i.e., we define distances between any two

solutions x and y as

$$\rho_k(x,y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}, \tag{4.7}$$

or

$$\rho_k(x,y) = \max_{1\le i\le n} |x_i - y_i|, \quad p = \infty. \tag{4.8}$$

The neighbourhood $\mathcal{N}_k(x)$ denotes the set of solutions in the k–th neighbourhood of x, and using the metric ρ_k, it is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x,y) \le r_k\}, \tag{4.9}$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid r_{k-1} \le \rho(x,y) \le r_k\}, \tag{4.10}$$

where r_k is a given radius of neigbourhood $\mathcal{N}_k$ $(k = 1, \ldots, k_{\max})$.

(i) *Values of radii* $r_k,\ k = 1, \ldots, k_{\max}$. Those values may be defined by the user or calculated automatically during the minimisation process. The geometry of the neighbourhood structure is induced by the ℓ_1 (4.7) and ℓ_∞ (4.8). We use balls as in (4.10). Radii $r_1 \le r_2 \le \ldots \le r_{k_{\max}}$ are automatically computed as follow: let $x = (x_1, y_1, \ldots, x_n, y_n, r) \in R^{2n+1}$ be the current incumbent solution and let

$$a_i \le x_i \le b_i,\ \ i = 1, \ldots, 2n+1 \tag{4.11}$$

defines the box or hyper-cube

$$H = \prod_{i=1}^{2n+1} [a_i, b_i]$$

around the incumbent solution x. In order to find k_{max} neighbourhoods automatically and thus make our `GLOB-VNS` more user-friendly, we divide $x_i - a_i$ and $b_i - x_i$ into a $k_{\max}$ intervals:

$$\underline{\rho}_i = \frac{x_i - a_i}{k_{\max}};\ \ \overline{\rho}_i = \frac{b_i - x_i}{k_{\max}}.$$

Then the $k_{\max}$ hyper-cubes (boxes) $H_1, H_2, \ldots, H_{\max}$ around the incumbent (the best solution found so far) x are given

$$a_j + (k-1)\underline{\rho}_i \le x_i \le b_i - (k-1)\overline{\rho}_i, k = 1, \ldots, k_{\max} \tag{4.12}$$

or

$$\underline{d}_{ik} \leq x_i \leq \overline{d}_{ik}, k = 1, \ldots, k_{\max} \tag{4.13}$$

Figure (4.2) illustrates different type of distributions.

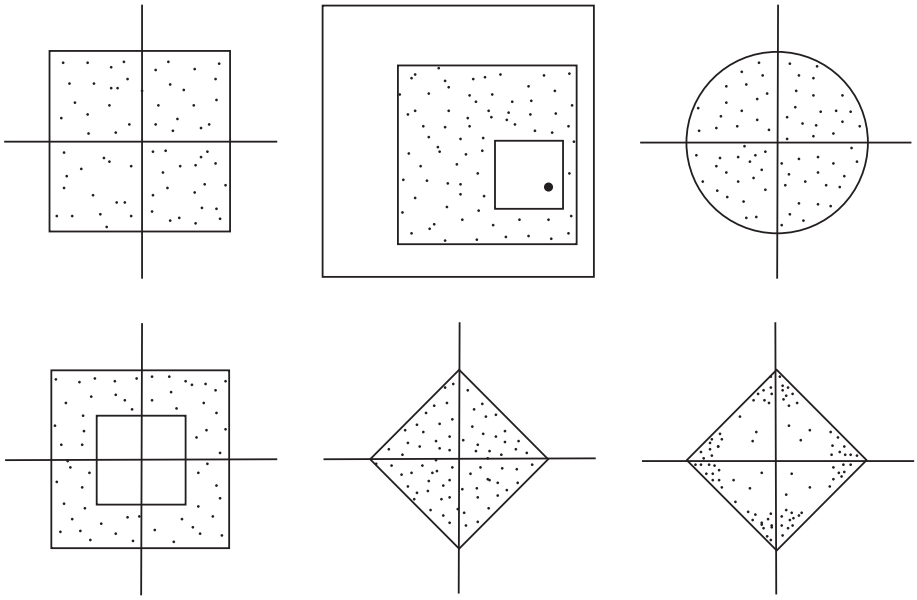


Figure 4.2: Different distribution types.

(ii) *(Geometry, distribution) pairs*. Geometry of neighbourhood structures $\mathcal{N}_k$ is defined by the choice of the metric functions used in the search through the solution space. The usual choices are the ℓ_1, ℓ_2, and ℓ_∞ norms. Their order in the search is also important within VNS. Different *distributions* may be used for obtaining the random point y from the same neighbourhood $\mathcal{N}_k$ in the *Shaking* step. Uniform distribution in $\mathcal{N}_k$ is the obvious choice, but other distributions may lead to a much better performance on some problems. Beside uniform (u), we also implement the hypergeometric distribution (h)(Drazić et al., 2006; Mladenović et al., 2008). The hypergeometric distribution (h) is designed as follows:

- The coordinate x_1 is taken uniformly on $[-1, 1]$, x_k is taken uniformly from $[-A_k, A_k]$ where $A_k = 1 - \mid x_1 \mid - \ldots - \mid x_{k-1} \mid, k = 2, \ldots, n-1$ and the last x_n takes A_n with a random sign.

- The coordinates of x are permuted randomly.

Note that different choices of geometric neighbourhood shapes and random point distributions lead to different VNS-based heuristics. We denote them as (α, γ), where α and γ represents geometry (metric) and distribution used, respectively. Therefore, in total we have 6 different variants of VNS defined by (geometry, distribution) pairs: (ℓ_1, u),(ℓ_2, u),(ℓ_∞, u),(ℓ_1, h), (ℓ_2, h), (ℓ_∞, h). Note that "u" denotes uniform distribution, while "h" denotes hypergeometric (special) distribution. For simplicity, we will denote those variants in pseudo-code as (1,1),(2,1),(3,1),(1,2),(2,2) and (3,2): (1,1)= (ℓ_1, u), (2,1)=(ℓ_2, u), etc. For example, pair (3,2) indicates that ℓ_∞ norm (3) and the special distribution (2) are used in the shaking step.

However, after extensive computational analysis, we have used different (*geometry*, *distribution*) pairs orders due to the used formulation. We have applied three (*geometry*, *distribution*) pairs in our `GLOB-VNS` for solving CPP with single formulation. For solving CPP by formulation (4.1), the distribution order is given by `dist_type_order` = (3,2) (1,1) and (1,2). For applying the formulation (4.3), the distribution order is given by `dist_type_order` = (3,1) (1,2) and (1,1), and for the formulation (4.4) the distribution order is `dist_type_order` = (3,2) (3,1) and (1,1). However, for the formulation (4.5) only two (*geometry*, *distribution*) pairs have been used `dist_type_order` = (3,2) and (1,2).

Furthermore, variant (*geometry*, *distribution*) pairs orders has been applied for solving CPP by using `RD-VNS`. We use four (*geometry*, *distribution*) pairs in our `RD-VNS` for solving CPP-1 and CPP-2. Switching between CPP-1 formulations, from (4.1) to (4.3) was after the mid of the CPU time, the distribution order is given by `dist_type_order` = (3,1) (1,1) (1,2) and (3,1), where in the inverse case the distribution order is `dist_type_order` = (3,1) (1,1) (3,2) and (1,2). For CPP-2, switching from (4.4) to (4.5) after half of the CPU time, the order distribution is `dist_type_order` = (3,2) (1,2) (3,2) and (1,1), for visa versa case it is `dist_type_order` = (1,2) (3,2) (3,1) and (1,2).

The (*geometry*, *distribution*) pairs have been found empirically. Table 4.1 gives an example of finding the best (*geometry*, *distribution*) pairs, the formulation (4.1) has been used with fixed CPU to 2 seconds, where all (*geometry*, *distribution*) pairs have been tried up to four pairs. The best result and the best (*geometry*, *distribution*) pairs order are given in

red in Table 4.1.

Table 4.1: Empirical example for choosing the $(geometry, distribution)$ pairs for formulation (4.1)

$(geometry, distribution) pair$	$(3,2)$	$(1,2)$	$(1,1)$	$(3,1)$
results	0.203976	0.257700	0.260078	0.261200
$(geometry, distribution)$ *pairs*	$(3,1)(1,1)$	$(3,1)(1,2)$	$(3,1)(3,2)$	
results	0.262230	0.262230	0.261140	
$(geometry, distribution)$ *pairs*	$(3,2)(1,1)$	$(3,2)(1,2)$	$(3,2)(3,1)$	
results	0.258920	0.259110	0.260764	
$(geometry, distribution)$ *pairs*	$(1,2)(1,1)$	$(1,2)(3,2)$	$(1,2)(3,1)$	
results	0.256880	0.257780	0.259338	
$(geometry, distribution)$ *pairs*	$(1,1)(3,2)$	$(1,1)(3,2)$	$(1,1)(3,1)$	
results	0.260070	0.260068	0.259379	
$(geometry, distribution)$ *pairs*	$(3,2)(1,2)(1,1)$	$(3,2)(1,2)(3,2)$	$(3,2)(1,2)(3,1)$	$(3,2)(1,1)(1,2)$
results	0.259169	0.259179	0.259178	0.258914
$(geometry, distribution)$ *pairs*	$(3,2)(1,1)(3,2)$	$(3,2)(1,1)(3,1)$	$(3,2)(3,1)(3,2)$	$(3,2)(3,1)(1,2)$
results	0.252600	0.258914	0.260700	0.260769
$(geometry, distribution)$ *pairs*	$(3,2)(3,1)(1,1)$	$(3,1)(1,2)(3,1)$	$(3,1)(1,2)(3,2)$	$(3,1)(1,2)(1,1)$
results	0.260771	0.261424	0.261480	0.262249
$(geometry, distribution)$ *pairs*	$(3,1)(1,1)(3,1)$	$(3,1)(1,1)(3,2)$	$(3,1)(1,1)(1,2)$	$(3,1)(3,2)(1,1)$
results	0.261670	0.261194	0.261987	0.261751
$(geometry, distribution)$ *pairs*	$(3,1)(3,2)(1,2)$	$(3,1)(3,2)(3,1)$	$(1,2)(1,1)(1,2)$	$(1,2)(1,1)(3,2)$
results	0.262210	0.261833	0.257070	0.260904
$(geometry, distribution)$ *pairs*	$(1,2)(1,1)(3,1)$	$(1,2)(3,2)(1,1)$	$(1,2)(3,2)(1,2)$	$(1,2)(3,2)(3,1)$
results	0.261902	0.258220	0.256383	0.260300
$(geometry, distribution) pairs$	$(1,2)(3,1)(1,1)$	$(1,2)(3,1)(1,2)$	$(1,2)(3,1)(3,2)$	$(1,1)(1,2)(1,1)$
results	0.258220	0.256380	0.257756	0.260067
$(geometry, distribution)$ *pairs*	$(1,1)(1,2)(3,2)$	$(1,1)(1,2)(3,1)$	$(1,1)(3,2)(1,1)$	$(1,1)(3,2)(1,1)$
results	0.26007	0.26139	0.262241	0.26006
$(geometry, distribution)$ *pair*	$(1,1)(3,2)(1,1)$	$(1,1)(3,1)(1,1)$	$(1,1)(3,1)(1,2)$	$(1,1)(3,1)(3,2)$
results	0.260070	0.259300	0.259909	0.260200
$(geometry, distribution)$ *pairs*	$(3,1)(1,1)(3,1)(1,1)$	$(3,1)(1,1)(1,2)(3,2)$	$(1,1)(1,2)(1,1)(1,2)$	$(1,1)(1,2)(1,1)(3,2)$
results	0.261390	0.255807	0.262239	0.260070
$(geometry, distribution)$ *pairs*	$(1,1)(1,2)(1,1)(3,1)$	$(1,1)(1,2)(3,1)(1,2)$	$(1,1)(1,2)(3,1)(1,1)$	$(1,1)(1,2)(3,1)(3,2)$
results	0.260070	0.261267	0.261170	0.261360
$(geometry, distribution)$ *pairs*	$(1,1)(1,2)(3,2)(1,2)$	$(1,1)(1,2)(3,2)(3,1)$	$(1,1)(1,2)(3,2)(1,1)$	$(1,1)(3,2)(1,2)(1,1)$
results	0.262244	0.260070	0.260070	0.261780
$(geometry, distribution)$ *pairs*	$(1,1)(3,2)(1,2)(3,2)$	$(1,1)(3,2)(1,2)(3,1)$	$(1,1)(3,2)(1,1)(1,2)$	$(1,1)(3,2)(1,1)(3,1)$
results	0.260096	0.260069	0.260074	0.260096
$(geometry, distribution)$ *pair*	$(1,1)(3,2)(1,1)(3,2)$	$(1,1)(3,2)(3,1)(1,1)$	$(1,1)(3,2)(3,1)(3,2)$	$(1,1)(3,2)(3,1)(1,2)$
results	0.260069	0.260980	0.260067	0.262244
$(geometry, distribution)$ *pairs*	$(1,1)(3,1)(1,1)(1,2)$	$(1,1)(3,1)(1,1)(3,2)$	$(1,1)(3,1)(1,1)(1,3)$	$(1,1)(3,1)(3,2)(1,1)$
results	0.260150	0.250000	0.250000	0.259500
$(geometry, distribution)$ *pairs*	$(1,1)(3,1)(3,2)(1,2)$	$(1,1)(3,1)(3,2)(3,1)$	$(1,1)(3,1)(1,2)(1,1)$	$(1,1)(3,1)(1,2)(3,2)$
results	0.259500	0.260036	0.259900	0.259900
$(geometry, distribution)$ *pairs*	$(1,1)(3,1)(1,2)(3,1)$	$(1,2)(1,1)(1,2)(1,1)$	$(1,2)(1,1)(1,2)(3,2)$	$(1,2)(1,1)(1,2)(3,1)$
results	0.259900	0.258260	0.257120	0.259080
$(geometry, distribution)$ *pairs*	$(1,2)(1,1)(3,2)(3,1)$	$(1,2)(1,1)(3,2)(1,1)$	$(1,2)(1,1)(3,2)(1,2)$	$(1,2)(1,1)(3,1)(1,1)$
results	0.257618	0.257570	0.258791	0.261322
$(geometry, distribution)$ *pairs*	$(1,2)(1,1)(3,1)(1,2)$	$(1,2)(1,1)(3,1)(3,2)$	$(1,2)(3,2)(1,2)(1,1)$	$(1,2)(3,2)(1,2)(3,2)$
results	0.261330	0.261307	0.256400	0.256324
$(geometry, distribution)$ *pairs*	$(1,2)(3,2)(1,2)(3,1)$	$(1,2)(3,2)(1,1)(3,1)$	$(1,2)(3,2)(1,1)(3,2)$	$(1,2)(3,2)(1,1)(1,2)$
results	0.256324	0.262122	0.258221	0.258822

$(geometry, distribution)$ *pairs*	$(1,2)(3,2)(3,1)(1,1)$	$(1,2)(3,2)(3,1)(1,2)$	$(1,2)(3,2)(3,1)(3,2)$	$(1,2)(3,1)(1,2)(1,1)$
results	0.260357	0.259501	0.260163	0.260002
$(geometry, distribution)$ *pair*	$(1,2)(3,1)(1,2)(3,1)$	$(1,2)(3,1)(1,2)(3,2)$	$(1,2)(3,1)(3,2)(1,1)$	$(1,2)(3,1)(3,2)(1,2)$
results	0.259732	0.259732	0.260510	0.259906
$(geometry, distribution)$ *pairs*	$(1,2)(3,1)(3,2)(3,1)$	$(1,2)(3,1)(1,2)(3,1)$	$(1,2)(3,1)(1,2)(3,2)$	$(1,2)(3,1)(1,2)(1,1)$
results	0.25447	0.259732	0.259732	0.259937
$(geometry, distribution)$ *pair*	$(3,2)(1,1)(1,2)(1,1)$	$(3,2)(1,1)(1,2)(3,1)$	$(3,2)(1,1)(1,2)(3,2)$	$(3,2)(1,1)(3,2)(1,1)$
results	0.256397	0.255807	0.255807	0.258919
$(geometry, distribution)$ *pairs*	$(3,2)(1,1)(3,2)(1,2)$	$(3,2)(1,1)(3,2)(3,1)$	$(3,2)(1,1)(3,1)(1,1)$	$(3,2)(1,1)(3,1)(1,2)$
results	0.259496	0.258917	0.258925	0.259490
$(geometry, distribution)$ *pair*	$(3,2)(1,1)(3,1)(3,2)$	$(3,2)(1,2)(3,2)(3,1)$	$(3,2)(1,2)(3,2)(1,1)$	$(3,2)(1,2)(3,2)(1,2)$
results	0.258917	0.259108	0.259130	0.259437
$(geometry, distribution)$ *pairs*	$(3,2)(1,2)(1,1)(3,1)$	$(3,2)(1,2)(1,1)(3,2)$	$(3,2)(1,2)(1,1)(1,2)$	$(3,2)(1,2)(3,1)(1,2)$
results	0.258231	0.259163	0.259111	0.259437
$(geometry, distribution)$ *pair*	$(3,2)(1,2)(3,1)(1,1)$	$(3,2)(1,2)(3,1)(3,2)$	$(3,2)(3,1)(3,2)(1,1)$	$(3,2)(3,1)(3,2)(1,2)$
results	0.258231	0.259108	0.260767	0.260076
$(geometry, distribution)$ *pairs*	$(3,2)(3,1)(3,2)(3,1)$	$(3,2)(3,1)(1,2)(1,1)$	$(3,2)(3,1)(1,2)(3,1)$	$(3,2)(3,1)(1,2)(3,2)$
results	0.256440	0.260770	0.261013	0.260773
$(geometry, distribution)$ *pairs*	$(3,2)(3,1)(1,1)(1,2)$	$(3,2)(3,1)(1,1)(3,1)$	$(3,2)(3,1)(1,1)(3,2)$	$(3,1)(1,1)(1,2)(1,1)$
results	0.260772	0.260766	0.260762	0.261569
$(geometry, distribution)$ *pair*	$(3,1)(1,1)(1,2)(3,1)$	$(3,1)(1,1)(1,2)(3,2)$	$(3,1)(1,1)(3,2)(3,1)$	$(3,1)(1,1)(3,2)(1,1)$
results	0.261560	0.261532	0.261549	0.261398
$(geometry, distribution)$ *pairs*	$(3,1)(1,1)(3,2)(1,2)$	$(3,1)(1,1)(1,2)(1,1)$	$(3,1)(1,1)(1,2)(3,1)$	$(3,1)(1,1)(1,2)(3,2)$
results	0.262064	0.261569	0.261568	0.261570
$(geometry, distribution)$ *pairs*	$(3,1)(1,2)(3,2)(1,1)$	$(3,1)(1,2)(3,2)(1,2)$	$(3,1)(1,2)(3,2)(3,1)$	$(3,1)(1,2)(3,1)(3,2)$
results	0.262220	0.262230	0.262233	0.262238
$(geometry, distribution)$ *pairs*	$(3,1)(1,2)(3,1)(1,2)$	$(3,1)(1,2)(3,1)(1,1)$	$(3,1)(1,2)(1,1)(1,2)$	$(3,1)(1,2)(1,1)(3,2)$
results	0.262234	0.262220	0.262234	0.262244
$(geometry, distribution)$ *pairs*	$(3,1)(1,2)(1,1)(3,1)$	$(3,1)(3,2)(1,1)(3,1)$	$(3,1)(3,2)(1,1)(3,2)$	$(3,1)(3,2)(1,1)(1,2)$
results	0.261417	0.261188	0.262234	0.262212
$(geometry, distribution)$ *pairs*	$(3,1)(3,2)(1,2)(3,2)$	$(3,1)(3,2)(1,2)(3,1)$	$(3,1)(3,2)(1,2)(1,1)$	$(3,1)(3,2)(3,1)(1,1)$
results	0.261525	0.261972	0.264729	0.261855
$(geometry, distribution)$ *pairs*	$(3,1)(3,2)(3,1)(1,2)$	$(3,1)(3,2)(3,1)(3,2)$		
results	0.262237	0.261865		

After that a radius from interval $[0, r_k]$ is taken at random in order to get a point from $\mathcal{N}_k(x)$. Therefore, a random point within the Shaking step of `GLOB-VNS` or `RD-VNS` is generated in two steps: (i) find random direction; (ii) find random radius along that direction.

Local Search. As a local search for solving CPP we apply the direct search Hooke-Jeeves nonlinear programming method since it does not use derivatives. The left and right boundaries a_j and b_j for variables are defined as $a_j = -1$ and $b_j = +1$. The `GLOB` has six local search methods: steepest descent, Fletcher-Powell, Fletcher-Reeves, Nelder-Mead, Hook-Jeeves and Rosenbrock, where we chose Hook-Jeeves nonlinear programming method by an empirical way. At the beginning, we fixed the other parameters on `GLOB` with fixed maximum running

time and we run the code for each method. Then we found that the Hook-Jeeves method gave better results than the other five methods. For that the Hook-Jeeves method has been used here as a local search method for solving the CPP.

Pseudo-code. Our `RD-VNS` procedure for solving the CPP problem contains two different formulations for solving the same problem. The `RD-VNS` algorithm for solving CPP is given in Algorithm 26, where $k_{\max}$ and $t_{\max}$ are the usual VNS parameters, given by the user.

Function `RD-VNS` $(x, \varphi_1, \varphi_2, k_{max}, t_{max})$

1 Select the set of neighbourhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$ induced from ℓ_1 and ℓ_∞ norms as in (4.7) and (4.8).
2 Choose (*geometry*, *distribution*) pairs order.
3 Set $\varphi_{active} = \varphi_1$
4 Find an initial point $x \in X$ with respect to φ_{active}
5 **while** $t < t_{max}$ **do**
6 $k \leftarrow 1$
7 **repeat**
8 **for** *(geometry, distribution) pairs order* **do**
9 Generate $x' \in \mathcal{N}_{k\varphi_{active}}(x)$ at random //Shaking
10 Apply Hooke-Jeeves NLP method starting with x' to get x'' //Local search
11 **if** $f(x'', \varphi_{active}) < f(x, \varphi_{active})$ **then**
12 $x \leftarrow x''$, and go to line 6.
else
13 Set $k \leftarrow k + 1$
until $k \leq k_{\max}$
14 Transform the point x to corresponding point in another formulation, where $x = 1/f(x, \varphi_{active})$
15 **if** $\varphi_{active} = \varphi_1$ **then**
$\varphi_{active} = \varphi_2$
16 **else**
$\varphi_{active} = \varphi_1$
17 $t \leftarrow$ `CpuTime()`
18 **return** x

Algorithm 26: RD-VNS algorithm

The pseudo-code for the RD-VNS in Algorithm 26 contains a few more lines than the pseudo-code for GLOB-VNS. They correspond to different formulations of CPP-1 or CPP-2. In line 3 the current formulation is denoted by the notation φ_{active}. Lines 14, 15 and 16 explain the switching between two formulations. Each variable $x_j, j = 1, \ldots, 2n+1$ is divided by pervious x_{2n+1}, which is in fact an objective function value of the current problem (r, R or L).

After choosing $(geometry, distribution)$ pairs and the active formulation φ_{active} in steps 1 and 2 respectively in Algorithm 26, we then choose a random initial solution x. We denote with x the incumbent solution. As explained in Algorithm 26, the outer loop of VNS is running until a predefined stopping condition is met. The inner loop is repeated k_{max} times, if there is no improvement in x. In each neighbourhood a random point from the $\mathcal{N}_k(x)$ is taken (line 9) and the well known Hooke-Jeeves unconstrained nonlinear programming code is run (line 10). The active formulation is denoted with φ_{active}. If the better solution is obtained, we save it (line 12) and repeat the entire process with the first neighbourhood (i.e., return to step 7).

4.4 Computational results

Our code is written in C++ and compiled with Microsoft Visual Studio 8.0. The program is run on Intel(R) Core(TM) 2 at 1.73 GHz with 2 GB of RAM. There is no information regarding the computers used to calculate the results for other methods.

This section is divided into two subsections, one is devoted for CPP-1 results, and the second includes the CPP-2 results.

4.4.1 CPP inside a circle container (CPP-1)

In this subsection all the CPP-1 results for $n = 10, \ldots, 200$ in four variant cases will be provided. The first case uses GLOB-VNS with the formulation (4.1) only, the second one uses GLOB-VNS with the formulation (4.3). The third case uses RD-VNS between the formulation (4.1) and the formulation (4.3), where in this case the algorithm starts with the formulation

(4.1), and it then switches to the formulation (4.3) after half of the CPU time allowed. Finally, this one is the opposite case of the third one, where the algorithm `RD-VNS` is between the formulation (4.1) and the formulation (4.3), but it starts with the formulation (4.3), and it then switches to the formulation (4.1) at the middle of the CPU time.

The CPP-1 results for applying each formulation, (4.1) or (4.3), independently and RD between them are given in Table 4.2, Table 4.3, Table 4.4 and Table 5.1, Table 5.2 in Appendix B respectively.

Table 4.2: Circle packing problem inside a circle container from $n = 10$ till $n = 200$

n	$Packonomia\ r$	$Packonomia\ R$	$VNS\ R$	$1/R$	$VNS\ r$	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R)$	CPU
10	0.262258924190165	3.81302563139812	3.81314511564	0.262250706352	0.262252685613	0.262237917065	3.841744800261	0.260298393566	5
11	0.254854701717148	3.92380440016308	3.98405329532	0.251000658343	0.254846440437	0.254832672306	3.924018529065	0.254840794607	5
12	0.248163470571686	4.02960193011618	4.11386484620	0.243080421304	0.248119614992	0.235600029831	4.246134559107	0.235508316112	10
13	0.236067977499789	4.23606797749978	4.23628332473	0.236055977220	0.236055876760	0.229716221641	4.236314101585	0.236054262271	10
14	0.231030727971008	4.32842855486083	4.34108348613	0.230357237587	0.230777936242	0.229249443398	4.378285093315	0.228399927983	10
15	0.221172539086390	4.52135696470616	4.80487625856	0.208121904954	0.220866792107	0.217302850498	4.561675997699	0.219217673615	10
16	0.216664742924422	4.61542559487319	4.89708085643	0.204203285450	0.211835467945	0.202357598459	4.795839725060	0.208514057460	10
17	0.208679665570499	4.79203374831057	4.88374718690	0.204760803893	0.205600293411	0.203583327960	4.919412554847	0.203276303593	10
18	0.205604646759568	4.86370330515627	4.99931887699	0.200027248632	0.205583373616	0.205585043415	5.121484214919	0.195255898102	10
19	0.205604646759568	4.86370330515627	5.40202550741	0.185115749385	0.205597969390	0.186239487270	4.869966786294	0.205340209468	10
20	0.195224011018748	5.12232073699152	5.55650196368	0.179969341600	0.193098879025	0.187340811950	5.259618626097	0.190127853574	10
21	0.190392146849053	5.25231747501024	5.68516021031	0.175896538181	0.190339459444	0.171149846172	5.289199593052	0.189064523357	15
22	0.183833026581681	5.43971895907221	5.72993577768	0.174522025865	0.183819088010	0.180651119858	5.542134974305	0.180435879789	15
23	0.180336009254436	5.54520422257485	5.83210171273	0.171464773637	0.178920412056	0.177214917927	5.714224071765	0.175001887823	15
24	0.176939130595961	5.65166109176543	5.83131006442	0.171488051390	0.172120523576	0.174432678254	5.694883407191	0.175596220063	15
25	0.173827661421222	5.75282433085711	6.24969644263	0.160007771446	0.173780662546	0.169432658411	5.951040457779	0.168037842642	15
26	0.171580252187166	5.82817653694294	6.34831576441	0.157522095168	0.171426730865	0.170039286552	5.859786161790	0.170654691552	20
27	0.169307931134573	5.90639784739415	6.41219448772	0.155952849202	0.169268984376	0.165696451404	5.932482276118	0.168563504020	20
28	0.166252750038606	6.01493809737151	6.60952951474	0.151296699375	0.165468862609	0.149572008226	6.203082173035	0.161210181020	20
29	0.162903649276644	6.13859790397814	6.52544140063	0.153246338233	0.162823425215	0.162404985561	6.215855098140	0.160878911141	20
30	0.161349109064689	6.19774107087922	7.01070678725	0.142638970698	0.160744429878	0.160312645702	6.239991150446	0.160256637532	20
31	0.158944541560340	6.29150262212918	7.06966816747	0.141449354667	0.157351242321	0.153610812669	6.646518081830	0.150454717446	25
32	0.155533985422770	6.42946297095011	7.23481695020	0.138220497752	0.154838345786	0.151579081027	6.551286154611	0.152641783064	25
33	0.154161517947058	6.48670312356043	7.01595075863	0.142532357253	0.151332222468	0.146796287163	6.792623470179	0.147218523799	25
34	0.151264028246755	6.61095709000100	7.22099553894	0.138485059935	0.149867372988	0.149874928733	6.659755283280	0.150155667508	25
35	0.149316776635116	6.69717109179024	7.58135684016	0.131902510472	0.149127566056	0.140770532362	6.836224299979	0.146279577164	25
36	0.148219429761119	6.74675379342417	7.50373266699	0.133267007818	0.148084669132	0.145488673038	6.791007128978	0.147253563574	25
37	0.147955904479076	6.75877048314363	7.67411109425	0.130308251694	0.147947064797	0.141077937921	7.018179050502	0.142487102823	25
38	0.143639218073289	6.96188696522814	8.39598637274	0.119104528712	0.142787890555	0.140992182434	7.302906495254	0.136931781976	25
39	0.141685521745403	7.05788416262400	7.72316547400	0.129480587120	0.141584629353	0.138533028344	7.102005144106	0.140805304940	25
40	0.140373604202714	7.12384643594312	8.16103174790	0.122533526506	0.138189404123	0.138013507202	7.207766588209	0.138739231877	25
41	0.137740812925344	7.26001232867704	7.89154122796	0.126717959282	0.136882256404	0.133025665694	7.344077029923	0.136164149140	30
42	0.136113748715697	7.34679640694276	8.18127504094	0.122230336347	0.135682101054	0.131897206125	7.450155308249	0.134225389757	30
43	0.134771891080212	7.41994485634121	7.94334335396	0.125891574296	0.134187183820	0.132508941303	7.927938859689	0.126136189708	30
44	0.133368245886005	7.49803668299524	8.17540463697	0.122318104657	0.132642531549	0.130884233526	7.706403373127	0.129762218714	30
45	0.132049594251630	7.57291232636752	8.81345273743	0.113462910597	0.131714472373	0.124456568404	7.710802705360	0.129688183995	30

Table 4.3: Circle packing problem inside a circle container from $n = 10$ till $n = 200$, continued table

n	*Packonomia r*	*Packonomia R*	$VNS\ R$	$1/R$	$VNS\ r$	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R)$	CPU
46	0.130715880038233	7.650179914693666	9.711150141801	0.102974414503	0.130393549434	0.129688148374	7.800061674111	0.128204114503	30
47	0.129463747326957	7.724170052598011	9.362703401112	0.106806758386	0.129346113931	0.126883662379	7.889059178121	0.126757827191	30
48	0.128348756542845	7.791271430558666	8.882680750641	0.112578626664	0.128196449396	0.126968807145	7.929513394321	0.126111143304	30
49	0.126792996262206	7.886870958802866	8.879820285361	0.112614891728	0.126375783520	0.124958516110	8.139511130757	0.122857501383	30
50	0.125825489530404	7.947515274783511	10.011468499161	0.099885446384	0.124564685920	0.124944304264	8.173678827402	0.122343931186	30
51	0.124571676602365	8.027506952419188	9.103142242133	0.109852177787	0.124194648834	0.121763067522	8.175598251601	0.122315207918	30
52	0.123690164592469	8.084717190689881	10.870518734041	0.091991930143	0.119833090913	0.122091006837	8.481913080114	0.117897930638	30
53	0.122255623687619	8.179582826841061	9.494682723741	0.105322108078	0.121281393917	0.121786441570	8.399227753409	0.119058564592	30
54	0.121892021856964	8.203982383469331	9.231971516411	0.108319225013	0.119821913351	0.117584577620	8.537828722286	0.117125797732	30
55	0.121786324527999	8.211102550927971	9.447042700881	0.105853231711	0.119192833291	0.118670928943	8.489578328959	0.117791480478	30
56	0.119281497082362	8.383529922578951	9.474312282331	0.105548558059	0.118082335563	0.116224778275	8.715745224561	0.114734882014	30
57	0.118382637651500	8.447184653410421	9.396388110261	0.106423871414	0.117725986457	0.116282385911	8.904228551644	0.112306191850	30
58	0.117308193128286	8.524553770139601	10.194893779631	0.098088319664	0.116899752212	0.115758691304	8.759456683478	0.114162331767	30
59	0.116380564996047	8.592499959370071	9.891061448381	0.101101383832	0.116169315086	0.115802860862	8.888930875515	0.112499468609	30
60	0.115657480132814	8.646219845457911	10.167607599191	0.098351553229	0.115396844555	0.112431117849	9.218931091457	0.108472445458	30
61	0.115456141678356	8.661297575540381	10.375308153431	0.096382679455	0.115412964402	0.111575593920	9.049962249811	0.110497698487	30
62	0.113253291982580	8.829765408972051	10.469362856721	0.095516796360	0.111349337987	0.112014555944	9.102283082216	0.109862546678	30
63	0.112456192917835	8.892351537550551	10.125857857531	0.098757064742	0.111843927185	0.111189823808	9.477373442410	0.105514466226	30
64	0.111582595825726	8.961971108485731	10.361721933501	0.096509055775	0.109809371749	0.110507103922	9.519741190601	0.105044872542	30
65	0.110896743722961	9.017397323208731	10.799771598541	0.092594550808	0.110804117032	0.109153997203	9.498548927397	0.105279238718	30
66	0.109935057298270	9.096279426924271	10.830108723941	0.092335176450	0.109327994306	0.108628117136	9.567425223748	0.104521329053	30
67	0.109063482023183	9.168971881783791	10.456755112421	0.095631961278	0.107734119498	0.107166009716	9.842119361176	0.101604132535	30
68	0.108345017704475	9.229773746750661	10.586260229911	0.094462064816	0.107248817079	0.107357918988	9.615872728393	0.103994720838	30
69	0.107877643364849	9.269761266641091	10.617233344727	0.094186495417	0.106195981743	0.105373178058	10.035732359450	0.099643948661	30
70	0.107001616605762	9.345653194048481	10.260285632251	0.097463173623	0.104518804208	0.105683324614	10.125739276656	0.098758221269	30
71	0.106204499837112	9.415796896870821	10.626392645391	0.094105312440	0.104751069307	0.104298846890	9.924274539356	0.100763032707	30
72	0.105553253159066	9.473890856713021	10.689045376901	0.093553723905	0.102786725139	0.103982162080	10.330917746174	0.096796821402	30
73	0.104817999688188	9.540346152137891	10.614724857851	0.094208753726	0.103160610821	0.102956240026	10.459777900994	0.095604324438	30
74	0.104283629835208	9.589232764339171	11.053722400581	0.090467261931	0.100259517968	0.103550939828	10.324170954991	0.096860077614	30
75	0.103390915666444	9.672029631947111	10.966016394901	0.091190817521	0.101945676278	0.100492804853	10.446533813798	0.095725531341	30
76	0.102779181946967	9.729596802161641	11.253836357881	0.088858587259	0.101879942176	0.101582972657	10.479120212497	0.095427858420	35
77	0.102052146983690	9.798911924506781	12.068179603681	0.082862538746	0.100067917676	0.097326021515	10.611446093375	0.094237862700	35
78	0.101443439719369	9.857709899884821	11.625806446211	0.086015538331	0.100288636962	0.100002201488	10.795755707180	0.092628994868	35
79	0.100958466545456	9.905063467661011	10.892478586341	0.091808469214	0.099198892714	0.099187456732	10.626567160005	0.094103766997	35
80	0.100319499416176	9.968151813153371	12.052760067531	0.082968547818	0.099016527045	0.097325570576	10.795998026410	0.092626915784	35
81	0.099891475491636	10.010864241200701	11.410396632251	0.087639372428	0.096561370400	0.097935937565	10.409409387558	0.096066929714	35
82	0.099494327805157	10.050824223450501	11.607011392901	0.086154821956	0.097141880486	0.097847896195	10.785440539076	0.092717585005	35
83	0.098844919276867	10.116857875102001	11.352181313841	0.088088797417	0.096799735776	0.096449283086	10.954750809594	0.091284595823	40
84	0.098526721390455	10.149530867236101	11.806908288961	0.084696177486	0.095063022076	0.095636929439	10.876584186213	0.091940629786	40
85	0.098395063692606	10.163111465876701	11.542832893781	0.086633845366	0.094092833097	0.095464160495	11.247099510240	0.088911812249	40
86	0.097099624005301	10.298701053110101	12.269126328241	0.081505396003	0.095145191083	0.095778323119	11.054432489329	0.090461450732	40
87	0.096495211836181	10.363208505077801	12.467952911481	0.080205628550	0.094978106431	0.094261565294	11.004219690868	0.090874230803	50
88	0.095855792771801	10.432337692732301	11.367872889411	0.087967204571	0.092427153927	0.094655969454	11.198920791429	0.089294318499 50	
89	0.095233634543870	10.500491814573601	11.328816612251	0.088270472921	0.092624483778	0.093330805678	11.281048940899	0.088644239134	50
90	0.094822059586948	10.546069177953701	11.704736876761	0.085435495947	0.091684199756	0.092973568144	11.396890351255	0.087743232512	50

Table 4.4: Circle packing problem inside a circle container from $n = 10$ till $n = 200$, continued table

n	$Packonomia\ r$	$Packonomia\ R$	$VNS\ R$	$1/R$	$VNS\ r$	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R)$	CPU
91	0.094636278506047	10.56677223350560	13.13680095335	0.0761220333328	0.092509279340	0.093365244805	11.1401676608412	0.089765256247	60
92	0.093592245754695	10.68464584791560	11.91947201106	0.0838963335585	0.090771236804	0.092751757265	11.1696305100046	0.089528476264	60
93	0.093167534622482	10.73335260025960	12.47710619593	0.0801467891911	0.091432171149	0.092318778139	11.5452975073328	0.086615351347	60
94	0.092781315283867	10.77803216025200	13.08013976987	0.0764517824422	0.090591131822	0.087474063849	11.6608792455455	0.085756826647	60
95	0.092249177760721	10.84020502159740	12.20956018137	0.0819030321144	0.091040097111	0.089818410600	11.5796057571148	0.086358725934	60
96	0.091884716482625	10.88320275972220	12.41680715576	0.0805360015222	0.090046565489	0.090554053904	11.6587650431142	0.085772377803	60
97	0.091419459906364	10.93859011007320	12.98542540928	0.0770094138990	0.090248337704	0.090461408511	11.7281092940530	0.085265235421	70
98	0.091079798229366	10.97938312820690	12.24106615142	0.0816922306950	0.087922281472	0.089795030156	11.7841968420230	0.084859410735	70
99	0.090636019812819	11.03314115144490	12.09575093573	0.0826736599750	0.087797192368	0.089571462972	11.9824296604630	0.083455528498	70
100	0.090235200288474	11.08214972431030	12.53929396485	0.0797493066840	0.087059469088	0.087558057432	11.9563095327710	0.083637848055	70
101	0.089710770521190	11.14693357542650	12.68969541281	0.0788040979290	0.086400389354	0.081578961698	12.3095645433960	0.081237642199	70
102	0.089310725493080	11.19686347276960	12.96591198929	0.0771253114190	0.086544966026	0.083964196155	12.1783894378070	0.082112664003	70
103	0.088769396870680	11.26514356582630	12.60422246141	0.0793384917680	0.086408579975	0.087864299140	12.1127520685780	0.082557621450	80
104	0.088357498517840	11.31765856633040	13.21841197418	0.0756520527540	0.083594896080	0.087420390981	12.0032770489810	0.083310582262	80
105	0.088007564607860	11.36265961290660	12.82010826628	0.0780024613860	0.084921166913	0.086792755265	12.4354702894750	0.080415133221	80
106	0.087551611059320	11.42183436604560	12.60535956355	0.0793313348150	0.085252120937	0.085729262451	12.5590883744550	0.079623613608	80
107	0.087168364837560	11.47205183742470	13.17002672387	0.0759299901940	0.084945109265	0.086094826430	12.3562491802290	0.080930708455	90
108	0.086775303711080	11.52401613403520	13.64243986735	0.0733006712670	0.085002656812	0.084234042858	12.3533276356070	0.080949848454	90
109	0.086489335895330	11.56211907107420	13.36958916153	0.0747966140110	0.084946174453	0.079188820879	12.5056656608120	0.079963756198	90
110	0.086081769647460	11.61686155030700	12.85029980871	0.0778191960410	0.082851066808	0.085963092740	12.5209171222610	0.079866354057	90
111	0.085742621069750	11.66281118449310	13.23203007097	0.0755741934260	0.081767143464	0.081757897936	12.8873872861600	0.077595247027	100
112	0.085431571702320	11.70527452643000	12.76920254570	0.0783134261060	0.083016234231	0.084299597829	12.5891497753810	0.079433481835	100
113	0.085124290793260	11.74752812248040	12.84516563831	0.0778503001170	0.082688453613	0.082432284913	12.7704778722360	0.078305605319	100
114	0.084780441044600	11.79517336403040	13.10587432232	0.0763016625530	0.081714318586	0.079491236019	12.8378977956170	0.077894373045	100
115	0.084463211728030	11.83947400934650	13.76727101132	0.0726360365230	0.080816310432	0.081808477903	12.7484088939150	0.078441161428	110
116	0.084056891551100	11.89670450033290	13.24268453551	0.0755113898510	0.081762742670	0.073532648804	12.8635492030701	0.077739042636	110
117	0.083727272204896	11.94347553627780	13.56138660241	0.0737387723920	0.081847965866	0.081398077860	12.6114132055370	0.079293254745	110
118	0.083433794252410	11.98555104631500	13.55945943174	0.0737492526920	0.081520924151	0.081480612994	13.2205660055610	0.075639726739	110
119	0.083040377649440	12.04233444387240	13.26250176446	0.0754005554730	0.080418403368	0.078194639307	13.0397961723070	0.076688315276	110
120	0.082745752572890	12.08521245992840	13.72510318909	0.0728591972110	0.075655704893	0.073295987266	13.6292055944670	0.073371847909	110
121	0.082475566262400	12.12480308190230	13.22063887127	0.0756393098500	0.079879200781	0.079223768836	13.1764930806960	0.075892727593	110
122	0.081937830936560	12.20437481160920	13.51314681433	0.0740020080990	0.079945706692	0.079444848504	13.1388273309560	0.076110293165	110
123	0.081457105292460	12.27639909003100	13.46364115889	0.0742741126410	0.080004866810	0.079777659368	13.3277975216760	0.075031151874	110
124	0.081157577700550	12.32170831526890	14.04590583624	0.0711951234520	0.078329776476	0.077202596397	13.2785390306590	0.075309489823	110
125	0.080852343328710	12.36822532074900	13.38497391311	0.0747106424330	0.077616454004	0.077202593312	13.5420412164290	0.073844111387	110
126	0.080531741712450	12.41746395564950	13.81448058727	0.0723878102900	0.077914390805	0.078005077383	13.4398807164130	0.074405422273	110
127	0.080246842401970	12.46154951481910	14.33706327122	0.0697492911260	0.077916794004	0.077033480671	13.6510051758060	0.073254678840	120
128	0.079985218277920	12.50231007091020	13.50049132585	0.0740713782830	0.073446152754	0.077637568627	13.6260859199700	0.073388646297	120
129	0.079657677067100	12.55371781878100	13.93001985826	0.0717874066350	0.077815738384	0.077961632644	13.3441859285890	0.074939003799	120
130	0.079350475497080	12.60231893678740	13.96284081745	0.0716186636430	0.077157472740	0.076895687881	13.8011049856590	0.072457966303	120

In Table 4.2, Table 4.3, Table 4.4, Table 5.1 and Table 5.2, n denotes the number of non-overlap equal circles inside the circle container. The value of n is within the interval $[10, 200]$. The *Packonomia* denotes the best known results in Packonomia website (Hungarian, 2009). This website has two possibilities for each number of circles within the circle container, firstly the best known results of the maximum radius of n non-overlap equal circles inside the unit circle container is denoted by *Packonomia* r. Secondly, the best known results of the minimum radius of circle container to accommodate n unit circles without overlap is denoted by *Packonomia* R. $VNS\ R$ denotes the minimum radius of circle container to accommodate n equal circles without overlap by using `GLOB-VNS` with the single formulation (4.3). Next column, in Tables 4.2, 4.3, 4.4, 5.1 and 5.2, is the one divided by $VNS\ R$ i.e. $\frac{1}{VNS\ R}$. $VNS\ r$ denotes the maximum radius for n equal circles within the unit circle container without overlap, where it has been found by `GLOB-VNS` with the single formulation (4.3) only.

The next two columns have been found by applying the `RD-VNS` algorithm, where $RD(R \rightarrow r)$ means that `RD-VNS` algorithm starts with the formulation (4.3) first, then it switches to the formulation (4.1), and vice versa for $RD(r \rightarrow R)$. The last column CPU gives the running time in seconds.

The next table compares our results(`GLOB-VNS` or `RD-VNS`) with the Packonomia website (Hungarian, 2009) by finding the average of the results. For finding the average, the n interval, $n \in [10, 200]$, has been divided into eight subintervals, $[10, 25]$, $[10, 50]$, $[10, 75]$, $[10, 100]$, $[10, 125]$, $[10, 150]$, $[10, 175]$ and $[10, 200]$. These averages are presented in Table 4.5.

Table 4.5: The average of the CPP-1 results, where $n \in [10, 200]$

n	*Packonomia* r	*Packonomia* R	R	$1/R$	r	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R))$	CPU
$10-25$	0.21192	4.79695	5.02251	0.20365	0.21085	0.20544	4.89662	0.20781	10.9375
$10-50$	0.17142	6.09999	6.74878	0.15834	0.1706	0.16654	6.23845	0.16785	20.12195
$10-75$	0.14939	7.14493	8.07279	0.13545	0.14834	0.14561	7.41922	0.144979	23.86364
$10-100$	0.13476	8.04369	9.15007	0.12120	0.13337	0.13147	8.45485	0.12974	30.989010
$10-125$	0.12407	8.84363	10.02838	0.11139	0.12237	0.12072	9.38265	0.11869	45.25862
$10-150$	0.11576	9.57317	10.79506	0.10401	0.11392	0.11268	10.2083	0.11028	75.95745
$10-175$	0.10906	10.24513	11.48404	0.09815	0.10715	0.10612	10.95063	0.10363	132.10843
$10-200$	0.10350	10.87146	12.12016	0.09331	0.10150	0.10070	11.64770	0.09811	202.93194

The data in Table 4.5 are average results between two fixed numbers of n, where n is the number of equal size circles within the container without overlap. In Table 4.5, *Packonomia* denotes the best results known in Packomania website (Hungarian, 2009), where *Packonomia* r is the best known results for maximising the radius of the n small circles and *Packonomia* R is the best known results for minimising the radius of the circle container. r denotes applying `GLOB-VNS` results to the formulation (4.1) only. R denotes applying `GLOB-VNS` results to the formulation (4.3). $RD(R \rightarrow r)$ denotes applying `RD-VNS` between (4.1) and (4.3), where the `RD-VNS` algorithm starts with the formulation (4.1). Also, $RD(r \rightarrow R)$ denotes applying `RD-VNS` to (4.1) and (4.3), where the `RD-VNS` algorithm starts with the formulation (4.3). CPU is computational time in seconds. The results in Table 4.5 have been rounded into five decimal numbers.

As shown in Table 4.5 and after comparing the average results for all different four cases, it can be noted that using `GLOB-VNS` with the formulation (4.1) for CPP-1 outperforms `GLOB-VNS` with the formulation (4.3) and using `RD-VNS` between (4.1) and (4.3).

Furthermore, Table 4.6 can prove that using `GLOB-VNS` with the formulation (4.1) for CPP-1 is the best among four variant cases by finding the percentage of the average difference between the best known results (Hungarian, 2009) and our `GLOB-VNS` and `RD-VNS` results for CPP-1.

In Table 4.6, n is the number of non-overlap equal circles inside the circle container. Besides, the value of n has been divided into eight subintervals as in Table 4.5. The percentage of the averages difference in Table 4.6 have been found by giving the VNS R as an example. It is calculated by deducting the `GLOB-VNS` results (which has been found by using the formulation (4.3) for minimising the radius of container R to accommodate n unit circles without overlap) from the best known results at the same n. After that the average between two fixed n is applied i.e. for each subinterval. Then the percentage for each average interval has been calculated. VNS r is between the best known results and `GLOB-VNS` results for the formulation (4.1). $RD(r \rightarrow R)$ denotes the percentage of the average difference between the best known results and `RD-VNS` (starting with the formulation (4.1)) and then switches to the formulation (4.3) after the mid of the time. $RD(R \rightarrow r)$ denotes the percentage of the

Table 4.6: The percentage of the averages difference compares our CPP-1 results with the best known results

n	$VNS\ R$	$VNS\ r$	$RD(R \rightarrow r)$	$RD(r \rightarrow R)$
$10-25$	0.377	0.070	0.203	0.443
$10-50$	0.451	0.080	0.193	0.407
$10-75$	0.480	0.117	0.205	0.376
$10-100$	0.444	0.129	0.219	0.375
$10-125$	0.421	0.127	0.228	0.376
$10-150$	0.404	0.126	0.226	0.369
$10-175$	0.394	0.127	0.225	0.357
$10-200$	0.375	0.131	0.217	0.348

average difference between the best known results and `RD-VNS` (starting with the formulation (4.3)) and then switches to the formulation (4.1) after the mid of the time.

As shown in Table 4.6, the best results are given by applying `GLOB-VNS` with the formulation (4.1) only (coloured in red). Furthermore, `RD-VND` between (4.1) and (4.3), starting with formulation (4.3) gives better results after the formulation (4.1).

4.4.2 CPP inside a Square container(CPP-2)

In this subsection all the CPP-2 results for $n = 10, \ldots, 200$ in four different cases will be provided: using `GLOB-VNS` with the formulation (4.4) only, using `GLOB-VNS` with the formulation (4.5), `RD-VNS` between formulation (4.4) and formulation (4.5), where in this case the `RD-VNS` algorithm starts with formulation (4.4) and then switches to the formulation (4.5) after the mid of CPU time. Besides, the last case is the same as the third one, but `RD-VNS` algorithm starts with the formulation (4.5) and switches to the formulation (4.4) after the half of the computational time. The CPP-2 results for applying each formulation, (4.4) or (4.5), independently and RD between them are given in Table 4.7, Table 4.8, Table 4.9 and Table 5.3, Table 5.4 in Appendix B.

Table 4.7: Circle packing problem inside a square container CPP-2 for $n = 10, \ldots, 200$

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \to r)$	$RD(r \to L)$	$1/(RD(r \to L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
10	0.1482043226	6.747839655	0.148195578	0.148171855	0.148066986	6.950243416	0.143879853	2	2	4	4
11	0.1423992377	7.134713573	0.140159796	0.14237902	0.142397616	7.098854368	0.1408678	2	2	8	4
12	0.1399588440	7.16006363	0.139663563	0.139941145	0.133272028	7.312903302	0.136744595	10	2	20	4
13	0.1339935135	7.593389901	0.131693488	0.132048574	0.133971723	7.535613186	0.132703202	10	2	20	4
14	0.1293317937	8.006838023	0.124893247	0.129067161	0.128275168	7.953424584	0.125732003	10	2	20	4
15	0.1271665475	7.95971688	0.125632609	0.127146832	0.127157229	8.103411381	0.123404817	10	2	20	8
16	0.1250000000	8.982164209	0.111331743	0.124997394	0.115934294	8.874801794	0.112678573	10	2	20	8
17	0.1171967428	9.000890927	0.111100113	0.115432763	0.11124249	9.00325685	0.111070918	10	2	20	8
18	0.1155214325	9.0000197	0.111110868	0.112677665	0.111360954	9.025109392	0.110801981	10	5	20	8
19	0.1122654376	9.000305964	0.111107334	0.112245212	0.111557897	9.059579644	0.110380397	10	5	20	12
20	0.1113823475	9.140446309	0.109403848	0.111336453	0.111368195	9.462355293	0.105681933	10	5	20	12
21	0.1068602124	9.820007057	0.101832921	0.106350449	0.105999875	9.667141247	0.103443197	10	5	20	12
22	0.1056652968	9.522485316	0.105014601	0.103511249	0.105652864	10.03274256	0.099673643	10	5	20	12
23	0.1028023234	9.984003825	0.100160218	0.101221759	0.101927982	10.10652656	0.098945963	10	5	20	12
24	0.1013818004	10.58405307	0.094481764	0.101332992	0.101277244	10.16095698	0.098415927	10	5	20	12
25	0.1000000000	10.74825868	0.093038326	0.099989586	0.097237987	10.66366298	0.093776407	10	5	20	12
26	0.0963623390	11.00624309	0.090857524	0.095700106	0.094354489	10.56411095	0.094660119	10	5	20	12
27	0.0954200017	11.00000538	0.090909046	0.09432853	0.093852119	11.00323291	0.090882381	10	5	20	15
28	0.0936728338	11.00398808	0.090876144	0.093370917	0.092041487	11.02449032	0.090707141	10	5	20	15
29	0.0924631440	11.82630241	0.084557283	0.09209188	0.090969663	11.00877119	0.09083666	10	5	20	15
30	0.0916710580	11.07561306	0.090288456	0.089320894	0.090377801	11.59004824	0.086280918	10	5	20	15
31	0.0893383334	11.66391037	0.085734541	0.088367566	0.088803797	11.74300331	0.085157091	15	5	30	20
32	0.0878581571	11.69682375	0.085493295	0.087593769	0.087105588	11.9023194	0.084017238	15	10	30	20
33	0.0872300141	11.94550595	0.083713491	0.085493242	0.085730103	11.95332229	0.08365875	15	10	30	20
34	0.0852703444	12.43130808	0.080442058	0.084558073	0.082428014	12.10410566	0.082616595	15	10	30	20
35	0.0842907121	13.07505198	0.076481531	0.083658373	0.082842978	12.73788121	0.078505992	15	10	30	20
36	0.0833333333	13.00005946	0.076922725	0.082149078	0.082401338	13.0007977	0.076918357	15	10	30	30
37	0.0820897664	13.76090906	0.072669618	0.080343024	0.080484742	13.01147007	0.076855267	15	10	30	30
38	0.0817097761	13.02153973	0.076795834	0.081600124	0.081610748	13.03070013	0.076741847	15	10	30	30
39	0.0813675270	13.02661574	0.076765909	0.081275027	0.0769235	13.0114176	0.076855576	15	10	30	30
40	0.0791867525	13.03106363	0.076739707	0.078771597	0.077429555	13.00708063	0.076881203	15	10	30	30
41	0.0784502101	14.00193261	0.071418713	0.077864465	0.075638522	13.15657762	0.076007608	20	15	30	40
42	0.0778015029	14.20796194	0.070383071	0.076867814	0.075876661	13.0252388	0.076774024	20	15	30	40
43	0.0763398106	14.30711577	0.06989529	0.076046432	0.075176401	13.59357214	0.073564181	20	15	60	40
44	0.0757819860	14.03871428	0.071231594	0.075109538	0.073447421	14.14662185	0.070688254	20	15	40	40
45	0.0747273437	13.9056049	0.071913448	0.074188267	0.072422442	14.03471223	0.071251906	30	15	40	40

Table 4.8: Circle packing problem inside a square container CPP-2 for $n = 10, \ldots, 200$, continued table

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \to r)$	$RD(r \to L)$	$1/(RD(r \to L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
46	0.0742721999	14.77515175	0.0676812	0.074204211	0.070439223	14.21158766	0.070365115	30	15	40	50
47	0.0731131513	14.26686365	0.07009249	0.072590087	0.070627633	14.31068911	0.069877837	30	15	50	50
48	0.0724322913	14.19323299	0.070456111	0.071310411	0.068723934	14.52711917	0.068836773	30	15	50	50
49	0.0716926817	14.76322073	0.067735897	0.068635767	0.071237125	15.03531559	0.066510077	30	15	50	50
50	0.0713771039	15.96097982	0.062652795	0.070283447	0.069590695	15.15135038	0.066000718	30	20	50	50
51	0.0710431381	15.0249248	0.066556074	0.070654029	0.069121705	15.00311283	0.066652835	40	20	50	60
52	0.0709576931	15.00059516	0.066664022	0.069008463	0.068382921	15.02126747	0.066572278	40	20	50	60
53	0.0699472526	15.09150442	0.066262446	0.068754144	0.066666688	15.05132093	0.066439351	40	20	60	60
54	0.0686455401	15.10692152	0.066194823	0.067299931	0.066667114	15.31488591	0.065295948	40	20	60	60
55	0.0680553606	15.25448497	0.065554491	0.066741686	0.067371866	15.22698443	0.065672885	40	20	120	60
56	0.0675325963	16.01060865	0.062458587	0.067404523	0.066492461	15.07350058	0.06634159	40	25	120	70
57	0.0670048731	16.01898587	0.062425924	0.066585404	0.065564227	15.42480246	0.064830652	40	25	120	70
58	0.0662329837	16.14300639	0.06194633	0.06531636	0.065164184	15.63860855	0.063944308	40	25	120	70
59	0.0658074969	16.35449788	0.061145258	0.065112102	0.064487033	15.91165975	0.062846995	50	25	120	70
60	0.0650304126	16.35452947	0.06114514	0.064628568	0.063973743	15.85436022	0.063074131	50	30	120	70
61	0.0646662689	16.13116763	0.061991793	0.062200914	0.063399975	15.96306125	0.062644626	50	30	120	90
62	0.0642521833	16.71303811	0.059833526	0.062321874	0.062075676	16.13174627	0.061989569	50	30	120	90
63	0.0640115282	22.88747704	0.043692015	0.061877546	0.061915248	16.36371142	0.061110831	50	30	120	90
64	0.0634589868	17.99794177	0.055561909	0.061968585	0.059402623	17.07119566	0.058578205	50	40	120	120
65	0.0632039571	18.00175138	0.055550151	0.061711445	0.061427955	17.10638568	0.058457702	50	30	120	120
66	0.0628622569	17.15828412	0.058280886	0.060903348	0.061900967	17.01182221	0.058782651	60	30	120	120
67	0.0625874295	18.00022439	0.055554863	0.058461734	0.058823729	17.00827252	0.058794919	60	30	120	120
68	0.0625200780	18.02596078	0.055475545	0.057515047	0.058182533	17.25253677	0.057962491	60	30	120	120
69	0.0613835717	17.24702587	0.057981011	0.05715713 3	0.058823915	17.24441352	0.057989795	70	40	120	120
70	0.0605966936	18.0158278	0.055506747	0.059139574	0.05882371	17.05373864	0.058638169	70	40	120	120
71	0.0600965314	18.13462982	0.055143116	0.058173955	0.057342423	17.29843878	0.057808685	70	40	120	120
72	0.0598010021	18.1819867	0.05499949	0.057637782	0.05589421	18.98755924	0.052666063	70	40	120	120
73	0.0593660506	18.14538542	0.05511043	0.057994786	0.056774711	17.72434007	0.05641959	70	40	120	120
74	0.0590823763	18.10828688	0.055223335	0.05576106	0.054165417	17.85728823	0.055999544	70	40	120	180
75	0.0584945353	18.37074734	0.054434367	0.058122633	0.057490409	17.92985319	0.055772905	70	60	180	180
76	0.0581985249	18.32500381	0.054570248	0.057162318	0.054759018	18.34770569	0.054502727	80	60	180	180
77	0.0578525779	17.94102642	0.055738171	0.05637166	0.056393707	17.9088569	0.055838293	80	70	180	180
78	0.0577024767	18.49753242	0.054061265	0.056849368	0.054862708	18.26159432	0.054759731	80	70	180	180
79	0.0575084978	18.22139441	0.054880542	0.056232042	0.056305716	18.87701638	0.052974473	120	70	180	180
80	0.0573706841	18.8310417	0.053103807	0.05547098	0.055283934	19.17412725	0.052153612	180	70	240	180
81	0.0568699211	19.98045427	0.050048912	0.05551851	0.056456785	19.11672507	0.052310215	240	70	240	180
82	0.0565122717	18.99401531	0.052648162	0.055499471	0.05483226	19.75638308	0.050616552	240	70	240	180
83	0.0561293730	19.01761501	0.052582829	0.055025203	0.054529659	19.23344684	0.051992761	240	70	240	240
84	0.0558566659	19.03131211	0.052544985	0.0554636	0.052631715	19.0455498	0.052505704	240	70	240	240
85	0.0556801818	20.00260932	0.049993478	0.054073108	0.054787494	19.42754929	0.051473296	240	70	240	240
86	0.0555729994	19.82146095	0.050450368	0.052850438	0.053200205	19.0141985	0.052592277	240	70	240	240
87	0.0546952597	19.9257493	0.050186318	0.052985623	0.052302219	19.87339943	0.050318518	240	70	240	240
88	0.0544066369	19.01111622	0.052600677	0.052392103	0.049259367	21.92528552	0.04560944	240	70	240	240
89	0.0539470409	19.23545113	0.051987343	0.051367372	0.050909286	20.11443753	0.049715534	240	70	240	240
90	0.0537499483	19.57718653	0.051079863	0.051765781	0.049028601	20.30778427	0.049242201	240	90	240	240

Table 4.9: Circle packing problem inside a square container CPP-2 for $n = 10, \ldots, 200$, continued table

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \to r)$	$RD(r \to L)$	$1/(RD(r \to L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
91	0.0534967199	19.51368298	0.051246092	0.052239964	0.051208018	19.14314127	0.052238031	240	90	300	300
92	0.0533170852	20.20217173	0.049499629	0.049419518	0.052632147	19.25012715	0.051947709	240	90	300	300
93	0.0529264334	19.23963584	0.051976036	0.05192664	0.049158888	19.93377672	0.050166108	300	90	300	300
94	0.0527953627	20.36203568	0.049111003	0.050308714	0.049364339	20.22767597	0.049437217	300	120	300	360
95	0.0524203665	20.29604051	0.049270694	0.051121613	0.04927334	19.94762595	0.050131279	300	120	300	360
96	0.0522754255	20.41251369	0.048989557	0.051086203	0.049955307	19.85728606	0.050359349	360	120	300	360
97	0.0521147955	20.47639417	0.048836723	0.050983104	0.04914519	20.02433501	0.049939236	360	120	300	360
98	0.0520329877	20.82067846	0.048029175	0.051129318	0.049228265	19.28351616	0.051857762	360	150	300	360
99	0.0519786064	20.16646049	0.049587284	0.050223923	0.048048854	22.37957756	0.044683596	360	120	300	360
100	0.0514010718	21.60280296	0.046290289	0.049228209	0.049336528	21.56927502	0.046362244	360	120	300	360
101	0.0510783563	21.01114241	0.047593795	0.04974958	0.048943927	21.02389315	0.04756493	360	120	360	360
102	0.0507744210	21.99759215	0.045459521	0.049584592	0.047619258	21.02471124	0.047563079	360	120	360	360
103	0.0505344976	21.08083351	0.047436455	0.049262837	0.047620233	21.08539218	0.047426199	360	120	360	360
104	0.0503487928	21.00073856	0.047617373	0.049367426	0.04762012	21.16426289	0.04724946	420	120	360	360
105	0.0502429484	21.99796423	0.045458752	0.04829607	0.047572398	21.19708733	0.047176293	420	120	360	420
106	0.0500152004	22.00783623	0.045438361	0.049142783	0.044278786	21.07983164	0.047438709	420	180	360	420
107	0.0495061669	21.44976132	0.046620565	0.048767864	0.047600791	21.30981183	0.04692674	480	180	360	420
108	0.0492435208	21.99952592	0.045455525	0.048152668	0.047619303	21.06894595	0.047463219	480	180	480	420
109	0.0490600565	21.23211011	0.047098475	0.047673256	0.047732856	21.33017902	0.046881932	480	180	480	420
110	0.0487794693	21.50164429	0.046508071	0.047861131	0.04549412	21.25718548	0.047042916	480	180	480	480
111	0.0486439363	21.99836143	0.045457931	0.047568355	0.046668592	21.55668016	0.046389332	480	180	480	480
112	0.0484454059	22.09648371	0.045256069	0.047509746	0.044782511	21.85055842	0.045765421	480	180	480	480
113	0.0482575885	22.38301411	0.044676735	0.046867604	0.046587777	26.75682502	0.037373642	480	180	600	480
114	0.0481677555	22.36201643	0.044718686	0.046797774	0.045006184	26.71337998	0.037434424	480	240	600	480
115	0.0479142475	22.08130193	0.045287185	0.046652406	0.044969174	22.49380306	0.044456689	560	240	600	600
116	0.0477759610	22.5137676	0.044417266	0.047442859	0.046426155	22.12561034	0.045196493	560	240	600	600
117	0.0476435266	22.03981088	0.04537244	0.046969458	0.047083852	22.0609439	0.045328976	560	240	600	600
118	0.0475723627	22.57550857	0.044295791	0.046386924	0.042194819	22.04011937	0.045371805	560	240	600	600
119	0.0475446462	21.96000991	0.04553732	0.046405183	0.044820533	21.94349901	0.045571584	660	240	660	600
120	0.0475299216	22.19905684	0.045046959	0.045624574	0.044330694	22.64585646	0.044158189	660	240	660	600
121	0.0468919996	23.64006672	0.042301065	0.045556198	0.04489027	26.14458123	0.038248844	660	240	660	600
122	0.0466266911	23.02795538	0.043425479	0.045649248	0.043478613	23.75641022	0.042093902	660	240	660	780
123	0.0463487952	24.00185151	0.041663452	0.045158053	0.043973707	23.55699185	0.042450242	660	240	660	780
124	0.0460997522	23.01695112	0.043446241	0.043705068	0.044351271	24.55785093	0.040720176	660	240	780	780
125	0.0459783365	23.99822019	0.041669757	0.045096775	0.044590479	23.09880265	0.043292287	660	300	780	900
126	0.0458287149	24.0141892	0.041642047	0.044886056	0.044014252	23.35029945	0.042826003	660	300	780	900
127	0.0456638697	23.02610709	0.043428965	0.044921122	0.043192766	23.78900477	0.042036227	660	300	780	900
128	0.0454732720	24.01061722	0.041648242	0.044295096	0.044081796	24.00007026	0.041666545	660	300	780	900
129	0.0452003511	24.06035889	0.04156214	0.044467635	0.042113788	23.26647272	0.042980301	660	300	780	900
130	0.0450431620	24.0486264	0.041582417	0.043948709	0.041734348	23.65985323	0.042265689	660	300	780	900

In Table 4.7, Table 4.8, Table 4.9 and Table 5.3, Table 5.4, n denotes the number of non-overlap equal circles inside the circle container. The n is within the interval $[10, 200]$. The $Pack\ r$ denotes the best known results in Packonomia website (Hungarian, 2009) for finding the maximum radius of n equal circles inside the unit square container without overlap. $VNS\ L$ denotes the minimum length of the side container of the square to accommodate n nonoverlaping unit circles by using `GLOB-VNS` with single formulation (4.5), where the next column in Tables 4.7, 4.8, 4.9, 5.3 and 5.4 is the one divided by $VNS\ L$ i.e. $\frac{1}{VNS\ L}$. $VNS\ r$ denotes the maximum radius for n equal circles within the unit square container without overlap, where it has been found by `GLOB-VNS` with the single formulation (4.3) only.

The next two columns show the results of applying the `RD-VNS` between the formulation (4.4) and the formulation (4.5), where $RD(L \rightarrow r)$ means that `RD-VNS` algorithm starts with the formulation (4.5) first, then it switches to the formulation (4.4) after half of CPU time, and vice versa for $RD(r \rightarrow L)$. In the last columns, CPU give the running time in seconds, where $CPU1$ denotes the computational time for using the formulation (4.5), and $CPU2$ denotes the computational time for using the formulation (4.4). $CPU3$ denotes the computational time for finding RD between (4.4) and (4.5) starting with (4.5), and $CPU4$ denotes the computational time for finding RD between (4.4) and (4.5) starting with (4.4).

The next table compares our results(`GLOB-VNS` or `RD-VNS`) with the best known results (Hungarian, 2009) by finding the average of the results. For finding the average, the n interval, where $n = 10, \ldots, 200$, has been divided into eight subintervals, $[10, 25]$, $[10, 50]$, $[10, 75]$, $[10, 100]$, $[10, 125]$, $[10, 150]$, $[10, 175]$ and $[10, 200]$. These averages are presented in Table 4.10.

Table 4.10: The average of the CPP-2 results, where $n = 10, \ldots, 200$

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \rightarrow r)$	$RD(r \rightarrow L)$	$1/(RD(r \rightarrow L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
$10-25$	0.11995	8.77407	0.11618	0.11924	0.11792	8.81316	0.11551	9.00	3.50	18.25	8.50
$10-50$	0.096985	11.39929	0.092476	0.09618	0.09505	11.29015	0.09292	14.73171	8.07317	27.60976	22.14634
$10-75$	0.08459	13.55834	0.079791	0.08342	0.08254	13.24880	0.08083	29.45455	16.83333	59.57576	51.33333
$10-100$	0.07637	15.21270	0.07192	0.07508	0.074181	15.01562	0.07262	88.61538	36.38462	113.09890	109.75824
$10-125$	0.07035	16.72008	0.06614	0.06909	0.06808	16.63159	0.0666	181.93103	71.47414	204.06897	201.44826
$10-150$	0.06568	18.11324	0.06164	0.064413	0.06342	18.058336	0.06199	314.35461	117.09929	345.758865	366.86525
$10-175$	0.06189	19.47829	0.05795	0.06062	0.05965	19.34503	0.05832	466.16867	177.89759	571.87952	597.27711
$10-200$	0.05875	20.62928	0.055	0.05744	0.05658	20.55585	0.055268	923.47644	253.87958	970.32461	891.03665

Table 4.10 shows the average results between two fixed numbers of n, where n is the number of equal size circles within the container. The average is calculated between two fixed n, where $n = 10, \ldots, 200$. In Table 4.10, $Pack\ r$ denotes the best known results in Packomania website for maximising the radius of n equal circles within the unit square container without overlap (Hungarian, 2009). $VNS\ L$ denotes the results of applying `GLOB-VNS` to the formulation (4.5) only, and $VNS\ r$ denotes the results of applying `GLOB-VNS` to the formulation (4.4). $RD(L \rightarrow r)$ denotes the results of applying `RD-VNS` algorithm between (4.4) and (4.5), where the `RD-VNS` algorithm starts with the formulation (4.5) and then switches to the formulation (4.4) at the mid was of the CPU time. Also, $RD(r \rightarrow L)$ denotes applying `RD-VNS` between (4.4) and (4.5), where the `RD-VNS` algorithm starts with formulation (4.4) and then switches to the formulation (4.5) at the mid was of CPU time.

CPU is the computational time in seconds, where $CPU1$ is the running time for using `GLOB-VNS` with the formulation (4.5). $CPU2$ is the running time for using `GLOB-VNS` with the formulation (4.4). $CPU3$ is the running time spent for using the `RD-VNS` between the formulation (4.4) and (4.5), starting with (4.5). The last $CPU4$ is computational running time for using the `RD-VNS` between the formulation (4.4) and (4.5), starting with (4.4).

As shown in Table 4.10, applying `GLOB-VNS` with the formulation given in (4.4) provides the best for CPP-2 problem as compared with the other cases. Moreover, it needs less computational time to run. These results have been coloured in red in Table 4.10. All results have been rounded to five decimal numbers.

Furthermore, Table 4.11 can prove that using `GLOB-VNS` with the formulation (4.4) for CPP-2 is the best among four different cases by finding the average difference. These average differences are calculated between the Packonamia website (Hungarian, 2009) and our `GLOB-VNS` and `RD-VNS` results for CPP-2.

Table 4.11: The percentage of the averages difference compares our CPP-2 results with the best known results

n	$VNS\ L$	$VNS\ r$	$RD(L \rightarrow r)$	$RD(r \rightarrow L)$
$10-25$	0.827	0.106	0.648	0.411
$10-50$	1.309	0.083	0.489	0.358
$10-75$	1.393	0.105	0.378	0.441
$10-100$	1.356	0.14	0.329	0.502
$10-125$	1.268	0.170	0.335	0.538
$10-150$	1.176	0.184	0.308	0.548
$10-175$	1.092	0.191	0.295	0.543
$10-200$	1.019	0.200	0.279	0.539

In Table 4.11, n is the number of non-overlap equal circles inside the circle container. Beside the n has been divided into eight subintervals as in Table 4.10. The percentage of the averages difference in Table (4.11) have been found by giving the $VNS\ r$ as an example. It is calculated by deducting the `GLOB-VNS` results (which have been found using the formulation (4.4) for maximising the radius r of n equal circles without overlap inside the unit square container) from the best known results at the same n. After that the average between two fixed n is applied i.e. for each subinterval. $VNS\ L$ denotes the same as above between the best known results and `GLOB-VNS` results for the formulation (4.5). $RD(r \rightarrow L)$ denotes the percentage of the average difference between the best known results and `RD-VNS` starting with the formulation (4.4) and then switches to the formulation (4.5) at mid was of the time. $RD(L \rightarrow r)$ denotes the percentage of the average difference between the best known results and `RD-VNS` starting with the formulation (4.5) and then switches to the formulation (4.4) at mid was of the time.

As shown in Table 4.11, the best results are given by the formulation (4.4) (in red). Furthermore, `RD-VND` between (4.4) and (4.5), starting with formulation (4.5) gives better

results for bigger n. All results have been rounded into five decimal numbers before finding the percentage.

4.5 Conclusion and future research

In this chapter we suggest a method that alternates two different formulations for solving the circle packing problem (CPP). Each formulation is solved by the variable neighbourhood search (VNS) global optimisation technique. Computer results show that our approach is comparable with some of the very best methods in the literature with more reasonable computational time.

Future research may include the use of other, more sophisticated global optimisation softwares for solving CPP such as LINGO (Systems, 2004), NMinimze (Research, 2005) and MathOptimizer Professional (Pintér, 1996). Moreover, it may include an extension of our approach to other types of containers where circles should be packed: rectangles, triangles and strips (Birgin and Gentil, 2010). In addition, new neighbourhood structures may be attempted within variable neighbourhood approach, as well as other types of formulations (Mladenović et al., 2007).

Chapter 5

Conclusion

The research reported in this thesis focuses on the development of novel variable neighbourhood search (VNS) based metaheuristics for censored quantile regression problems (CQR) and circle packing problems (CPP). The aim of this thesis is twofold. The first is to design continuous variable neighbourhood search for censored quantile regression problems. Circle packing problems have been studied as the second part of this thesis, where the reformulation descent within variable neighbourhood search by using two Cartesian formulations has been applied. Neighbourhood structures are implicity defined and updated according to the set of parameters, where these parameters depend on the mathematical formulation of the used problem. The `Glob` package within the visual studio C++ solver is used with VNS to be applied on problems. However, the aim of this thesis is not only to apply a general continuous variable neighbourhood search on problems. The change of VNS components have been applied to adapt the problems such as the change of the neighbourhood structure.

Chapter 1 is a survey of local search based metaheuristic methods. This chapter provided the theoretical and practical aspects of neighbourhood search, where it covered from the simplest local search to highly technical ones such as large-scale neighbourhood search and reformulation descent. Furthermore, it covered the concepts and components of each local search based metaheuristic method. This chapter presented different classes of metaheuristics, in particular, the important single-solution metaheuristic methods. Also, the important aspects of metaheuristics, intensification and diversification, were discussed.

Chapter 2 focuses on variable neighbourhood search metaheuristic, which this thesis depends on. This chapter gives a brief explanation of the most types of variable neighbourhood search.

Chapter 3 showed how continuous variable neighbourhood search within the `Glob` package can be successfully employed as a new censored quantile regression technique. The Powell estimator for CQR has been used as the objective function. This function is not convex, nor concave and not even differentiable. The methods in the literature usually linearised the CQR model and then solved it exactly. In this chapter, the CQR by Powell estimator has been found by applying an approximation method on an exact CQR model. Different metric functions within the VNS for defining the neighbourhood structure have been developed as the basic idea of our research in this thesis. This VNS approach was applied on groups of test instances and on real data. This new approach is a competitive one with state-of-the-art methods from the literature, where the VNS results outperforms the other results from the literature. This means using the nonlinear model with the approximate solution method is better than using an exact solution method with a linearised model. Future research in this direction may include extending this approach on the semi censored quantile regression or using another global optimisation techniques.

In Chapter 4, a new approach for solving circle packing problems was proposed. CPP in $2-dimensional$ space was considered within two types of containers: a circle and a square. The reformulation descent variable neighbourhood search `RD-VNS` has been applied on two Cartesian formulations for each type of container. In addition to applying `RD-VNS`, `GLOB-VNS` has been applied on each formulation independently. The results showed that our approach is comparable with some of the very best methods in the literature with more reasonable computational time. Future research may include applying `RD-VNS` with two Cartesian formulations on different types of containers such as a rectangle, a triangle and a strip. Moreover, different types of formulation like polar coordinates could be used with a new neighbourhood structure. The idea of reformulation descent could be applied on $3-dimensional$ containers.

To sum up, the research reported in this thesis may be useful in the scientific and indus-

trial fields. An up to date convergence of exciting metaheuristic methodologies proposed in this thesis could help researchers to develop an efficient computational heuristic approach for solving continuous optimisation problems with large instances especially in the econometrical field. In addition, the practitioner in the industrial field, such as circular cutting, container loading and cylinder packing, or in real life, such as extramarital affairs or exams results, may benefit from the application of these techniques. These techniques could be used in solving variant applications in the real world. In this thesis, the author would like to encourage adopting the metaheuristic approaches for solving the continuous optimisation problems especially in econometric, for example censored quantile regression problems and their real applications.

Bibliography

E. Aarts and J. Korst. *Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing.* John Wiley & Sons Inc., 1988.

E. Aarts, J. Korst, and W. Michiels. *Simulated annealing. InSearch methodologies: Introductory tutorials in optimization and decision support techniques*, chapter 7, pages 187–210. E. K. Burke & G. Kendall, Nottingham, UK, 2 edition, 2005.

E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. A quantitative analysis of the simulated annealing algorithm: A case study for the travelling salesman problem. *Journal of Statistical Physics*, 50(1):187–206, 1988.

B. Addis, M. Locatelli, and F. Schoen. Disk packing in a square: a new global optimization approach. *INFORMS Journal on Computing*, 20(4):516–524, 2008.

I. Althofer and K.-U. Koschnick. On the convergence of threshold accepting. *Appl. Math. Optim.*, 24:183–195, 1991.

T Amemiya. Two stage least absolute deviations estimators. *Econometrica*, 50:689–711, 1982.

A. Andreatta and C. Ribeiro. Huersitics for the phylogeny problem. *Journal of Heuristics*, 8(4):429–447, 2002.

F. Arito and G. Leguizamón. Incorporating tabu search principles into aco algorithms. *Hybrid Metaheuristics*, 5818:130–140, 2009.

C. Audet, J. Brimberg, P. Hansen, S. L. Digabel, and N. Mladenović. Pooling problem: alternate formulation and solution methods. *Managment Science*, 50(6):761–776, 2004.

S. Babu, T. Senthil Kumar, and V. Balasubramanian. Optimizing pulsed current gas tungsten arc welding parameters of aa6061 aluminium alloy using hooke and jeeves algorithm. *Transactions of Nonferrous Metals Society of China*, 18(5):1028–1036, 2008.

I. Barrodale and F. Roberts. Solution of an overdetermined system of equations in the ℓ_1 norm. *Communications of the ACM*, 17:319–320, 1974.

S. K. Bath, J. S. Dhillon b, and D. P. Kothari. Fuzzy satisfying stochastic multi-objective generation scheduling by weightage pattern search methods. *Electric Power Systems Research*, 69:311–320, 2004.

R. Battiti and G. Tecchiolli. The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research*, 63:153–188, 1996.

R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*. Springer Science + Business Media, LLC, 2008.

M. S. Bazarra, H. D. Sherali, and C. M. Shetty. *Unconstrained optimization: In Nonlinear programming: Theory and algorithms*, chapter 8, pages 265–359. John Wiley & Sons, Inc., Canada, 2 edition, 1993a.

M. S. Bazarra, H. D. Sherali, and C. M. Shetty. *The Fritz John and Karush-Kuhn-Tucker optimality conditions: In Nonlinear programming: Theory and algorithms*, chapter 4, pages 131–183. John Wiley & Sons, Inc., Canada, 2 edition, 1993b.

G. S. Becker. A theory of marriage: Part i. *Journal of Political Economy*, 81:813–846, 1973.

P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147:629–643, 2003.

Y. Bilias, S. Chen, and Z. Ying. Simple resampling methods for censored regression quantiles. *Journal of Econometrics*, 99:373–386, 2000.

E. G. Birgin and J. M. Gentil. New and improved results for packing identical unitary radius circles within triangles, rectangles and strips. *Computers & Operations Research*, 37:1318–1327, 2010.

E. G. Birgin and F. N. C. Sobral. Minimizing the object dimensions in circle and sphere packing problems. *Computers & Operations Research*, 35:2357–2375, 2008.

E. G. Birgin, J. M. Martínez b, and D. P. Ronconi. Optimizing the packing of cylinders into a rectangular container: A nonlinear approach. *European Journal of Operational Research*, 160:19–33, 2005.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

R. Blundella and J. L. Powell. Censored regression quantiles with endogenous regressors. *Journal of Econometrics*, 141:65–83, 2007.

D. W. Boll, J. Donovan, R. L. Graham, and B. D. Lubachevsky. Improving dense packings of equal disks in a square. *The Electronic Journal of Combinatorics*, 7(1):1–9, 2000.

J. Brimberg, P. Hansen, N. Mladenović, and É Taillard. Improvement and comparison of heuristics for solving the multisource webber problem. *Operations Research*, 48(3):444–460, 2000.

J. Brimberg, N. Mladenović, D. Urosević, and E. Ngai. Variable neighborhood search for the heaviest k-subgraph. *Computers & Operations Research*, 36(11):2885–2891, 2009.

M Buchinsky. Change in the u.s. wage structure 1963-1987: Application of quantile regression. *Econometrica*, 62:405–458, 1994.

M. Buchinsky and J. Hahn. An alternative estimator for the censored quantile regression model. *Econometrica*, 66(3):653–671, 1998.

G. Caporossi and P. Hansen. Variable neighborhood search for extremal graph 1. the autographix system. *Discrete Mathematics*, 212:29–44, 2000.

G. Caporossi and P. Hansen. Variable neighborhood search for extremal graph 5. three ways to automate finding conjecture. *Discrete Mathematics*, 276(1):81–94, 2004.

L. G. Casado, I. Garcya, P. G. Szabo, and T. Csendes. *Packing Equal circles packing in square. II. New results for up to 100 circles using the TAMSASS-PECS algorithm. In Optimization Theory: Recent Developments from Mátraháza*, chapter 15, pages 207–224. Kluwer Academic Publishers,, Dordrecht, The Netherlands, 1 edition, 2001.

I. Castillo, F. J. Kampas, and J. D. Pintér. Solving circle packing problems by globla optimization: numerical results and industrial application. *European Journal of Operational Research*, 191:786–802, 2008.

V. Cerny. Thermodynamical approach to traveling salesman problem: an sfficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

P. Chardaire, J. L. Lutton, and A. Sutter. Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. *European Journal of Operational Research*, 86:565–579, 1995.

P. Chaudhuri, K. Doksum, and A. Samarov. On average derivative quantile regression. *The Annals of Statistics*, 25:715–744, 1997.

R. Chelouah and P. Siarry. Tabu search applied to global optimization. *European Journal of Operations Research*, 123:256–270, 2000.

S Chen and S Khan. Estimating censored regression models in the presence of nonparametric multiplicative heteroskedasticity. *American Statistical Association*, 98:283–316, 2000.

V. Chernozhukov and H. Hong. Three-step censored quantile regression and extramarital affairs. *American Statistical Association*, 97(459):872–882, 2002.

V. Chernozhukov and H. Hong. Tan mcmc approach to classical estimation. *Journal of Econometrics*, 115(2):293–346, 2003.

K. W. Chu, Y. Deng, and J. Reinitzy. Parallel simulated annealing by mixing of state. *Journal of Computational Physics*, 148:646–662, 1999a.

K. W. Chu, Y. Deng, and J. Reinitzy. Parallel simulated annealing by mixing of states. *Journal of Computational Physics*, 148:646–662, 1999b.

B. W. Clare and D. L. Kepert. The closest packing of equal circles on a sphere. In *In Proceedings of the Royal Society of London. Series A, Mathematical and PhysicalSciences, Vol. 405, No. 1829, 329-344*, The Royal Society, Jun. 9, 1986.

Sergio Consoli. *The development and application of metaheuristics for problems in graph theroy: a computational study.* PhD thesis, School of Information Systems, Computing and Mathematics, Burnel University, 2008.

A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimusing multimodal functions of continuous varaibles with the simulated annealing algorithm. *ACM Transcations on Mathematical Software*, 13(3):262–280, 1987.

M. H. Correia, J. F. Oliveira, and J. S. Ferreira. A new upper bound for the cylinder packing problem. *International Transactions in Operational Research*, 8:571–583, 2001.

M.C. Costa, F.R. Monclar, and M. Zrikem. Variable neighborhood decomposition search for the optimization of power plant cable layout. *Journal of Intelligent Manufacturing*, 13(5): 353–365, 2002.

T. G. Crainic and M. Toulouse. *Parallel strategies for meta-heuristics. In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics*, chapter 17, pages 475–514. Kluwer Academic Publishers, Norwell, 1 edition, 2003.

T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10:293–314, 2004.

T.G. Crainic and M. Gendreau. *Towards an evolutionary method-Cooperative multi-thread parallel tabu search heuristic hybrid. In S. Voss, S. Martello, I. H. Osman and C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 23, pages 331–344. Kluwer Academic, Boston, Untied state of America, 1 edition, 1999.

T.G. Crainic, M. Gendreau, P. Soriano, and M. Toulouse. A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research*, 41:359–383, 1993.

T.G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal of Computing*, 9:61–72, 1997.

T.G. Crainic, M. Gendreau, and J.M. Farvolden simplex-based tabu search for the multicommodity capacitated fixed charge network design problem. *INFORMS Journal of Computing*, 12:223–236, 2000.

Y. Cui. Generating optimal t-shape cutting patterns for circular blanks. *Computers & Operations Research*, 32:143–152, 2005.

C. de Groot, R. Peikert, and D. Wurtz. The optimal packing of ten equal circles in a square. Technical Report 90-12, 1990.

C. de Groot, M. Monagan, R. Peikert, and D. Wurtz. Packing circles in a square: Review and new results, in: System modeling and optimization. In *In Proceedings of the 15th IFIP Conference , ed. P. Kall, Zurich,Lecture Notes in Control and Information Services, Vol. 180, 45–54, 1992*, Zurich, 1992.

A. Dekkers and E. Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50:367–393, 1991.

K. A. Dowsland. Optimising the palletisation of cylinders in cases. *OR Spectrum*, 13:204–212, 1991.

K. A. Dowsland and W. B. Dowsland. Packing problems. *European Journal of Operational Research*, 56:2–14, 1992.

M. Drazić, V. Kovacevic-Vujcić, M. Cangalović, and N. Mladenović. Glob- a new vns-based software for global optimization. *Nonconvex Optimization and Its application: Global Optimization, Springer US*, 84:135–154, 2006.

G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

B. Efron and R. J. Tibshirani. *An introduction to bootstrap.* Chapman and Hall, London, UK, 1993.

R. C. Fair. Cowles foundation for research in economics. *Journal of Political Economy*, 86: 45–61, 1976.

O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, 15:267–283, 2003.

B. Fitzenberger. *Parctical methods of optimization: constrainrd optimization.* Maddala G S and Rao C R, Handbook of statistics, Amsterdam, North-Holland, 1997.

C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph colouring. *Annals of Operations Research*, 63:437–461, 1996.

H. J. Fraser and J. A. George. Integrated container loading software for pulp and paper industry. *European Journal of Operational Research*, 77:466–474, 1994.

J. Gao, L. Sun, and M. Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research*, 35(9): 2892–2907, 2008.

F. Garclá-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8: 375–388, 2002.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

M. Gendreau, G. Laporte, and G. Laporte. A tabu search heurstics for the vehicle routing problem. *Managment Science*, 40:1276–1290, 1994.

J. A. George, J. M. George, and B. W. Lamar. Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, 84:693–712, 1995.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

F. Glover. Tabu search-part i. *ORSA Journal of computing*, 1:190–206, 1989a.

F. Glover. Tabu search-part ii. *ORSA Journal of computing*, 2:4–32, 1989b.

W. L. Goffe, G. D. Ferrier, and J. Rogers. Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60:65–99, 1994.

M. Goldberg. Packing of 14, 16, 17 and 20 circles in a circle. *Mathematics Magazine*, 44(3): 134–139, 1971.

R. L. Graham. Sets of points with given minimum separation (solution to problem el921). *American Mathematics*, 75:192–193, 1968.

R. L. Graham and B. D. Lubachevsky. Repeated patterns of dense packings of equal disks in a square. *The Electronic Journal of Combinatorics*, 3(1):1–17, 1996.

R. L. Graham, B. D. Lubachevsky, K. J. Nurmela, and P. R. J. Ostergard. Dense packings of congruent circles in a circle. *TDiscrete Mathematics*, 81:139–154, 1998.

I. Griva. Numerical experiments with an interior-exterior point method for nonlinear programming. *Computational Optimization and Applications*, 29:173–195, 2004.

A. Grosso, A. R. M. J. U. Jamali, M. Locatelli, and F. Schoen. Solving the problem of packing equal and unequal circles in a circular container. *Journal Glob Optim*, 47:63–81, 2010.

T. Grunert. *Lagrangean tabu search: In C.C. Ribeiro and P. Hansen (Eds.), Essays and Surveys in Metaheuristics*, pages 379–397. Kluwer Academic Publishers, Boston, USA, 1 edition, 2002.

J. Hahn. Bootstrapping quantile regression estimators. *Econometric Theory*, 11:105–121, 1995.

P. Hansen and N. Mladenović. J-means: A new local search heuristic for minimum sum-of-squares clustring. *Pattern Recognition*, 34:405–413, 2001a.

P. Hansen and N. Mladenović. *An Introduction to Variable Neighborhood Search. In S. Voss and S. Martello and I. H. Osman and C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 30, pages 433–458. Kluwer Academic Publishers, Boston, United state of America, 1 edition, 1999.

P. Hansen and N. Mladenović. *Variable neighbourhood search. In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics*, chapter 6, pages 145–184. Kluwer Academic Publishers, Norwell, 1 edition, 2003.

P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154:802–817, 2006.

P. Hansen and N. Mladenović. *Indudtrial applications of the variable neighborhood search. In: C. Riberio, P. Hansen (eds.), Essays and surveys in metaheuristics*, pages 415–440. Kluwer academic publishers, Boston, United state of America, 1 edition, 2001b.

P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

P. Hansen, N. Mladenović, and D. Urosević. Variable neighborhood search and local branching. *Computers and Operations Research*, 33(10):3034–3045, 2006.

P. Hansen, J. Brimberg, D. Urošević, and N. Mladenović. Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS Journal on Computing*, 19(4): 552–564, 2007a.

P. Hansen, J. Lazić, and N. Mladenović. Variable neighbourhood search for colour image quantization. *IMA Journal of Management Mathematics*, 18(2):207–221, 2007b.

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6:319–360, 2008.

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of OR*, 175:367–407, 2010.

A. Hertz and M. Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science*, 35(4):425–434, 2001.

A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

M. Hifi and R M'Hallah. Approximate algorithms for constrained circular cutting problems. *Computers & Operations Research*, 31:675–694, 2004.

M. Hifi and R M'Hallah. Adaptive and restarting techniques-based algorithms for circular packing problems. *Comput Optim Appl*, 156:17–35, 2008.

M. Hifi, V. T. Paschos, and V. Zissimopoulos. A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*, 159:430–448, 2004.

D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the Association for Computing Machinery*, 32: 130–136, 1985.

I. Hong, A. B. Kahng, and B. R. Moon. Improved large-step markov chain variants for the symmetric tsp. *Journal of Heuristics*, 3(1):63–81, 1997.

B. Honoré, S. Khan, and J. L. Powell. Quantile regression under random censoring. *Journal of Econometrics*, 109(1):67–105, 2002.

R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *J. Ass. for Comput.*, 8:212–229, 1961.

M. Hosseinkouchack. Further improvements in the calculation of censored quantile regressions. *Journal of Computational and Applied Mathematics*, 235:1429–1445, 2011.

W. Y. Hsiang. On the sphere packing problem and the proof of keplers conjecture. *Internat. J. Math*, 93:739–831, 1993.

B. Hu and G. R. Raidl. Variable neighborhood descent with self-adaptive neighborhood-ordering. In *In Proceedings of th EU/Meeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics, Malaga, Spain, 2006*, Malaga, Spain, 2006.

B. Hu, M. Leitner, and G.R. Raidl. he generalized minimum edge biconnected network problem: Efficient neighborhood structures for variable neighborhood search. Technical Report TR-186-1-07-02, 2009.

W. Q. Huang, Y. Li, B. Jurkowiak, and C.M. Li. A two-level search strategy for packing unequal circles into a circle container. In *In Francesca Rossi (ed) Proceedings of Principles and Practice of Constraint Programming - CP2003, LNCS 2833, 868-872.*, Kinsale, Ireland. Springer, 2003.

W. Q. Huang, Y. Li, and C. M. Li. Greedy algorithms for packing unequal circles into a rectangular container. *Journal of Operational Research in Society*, 56:539–548, 2005.

W. Q. Huang, Y. Li, C. M. Li, and R. C. Xu. New heuristics for packing unequal circles into a circlar container. *Computer & Opterationns Resaerch*, 33:2125–2142, 2006.

Hungarian. The best known results for circle packing problems @ONLINE, June 2009. URL `http://www.packomania.com/`.

D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(13):79–100, 1988.

P. Kilby, P. Prosser, and P. Shaw. *Guided local search for the vehicle routing problem with time windows. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol (eds), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 32,

pages 473–486. Kluwer Academic Publishers, Boston, United state of America, 1 edition, 1999.

S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:957–986, 1984.

S. Kirkpatrick, D. C. Gelatt, Jr., and Vecchi. Optimization by simulated annealing. *Science*, 220:3671–680, 1983.

R. Koenker and G. S. Bassett. Regression quantiles. *Econometrica*, 46:33–50, 1978.

R. Koenker and B. J. Park. An interior point algorithm for nonlinear quantile regression. *Journal of Econometrics*, 71:1097–1100, 1996.

S. Kravitz. Packing cylinders into cylindrical containers. *Mathematics Magazine*, 40:65–71, 1967.

J. Lazić, S. Hanafi, N. Mladenović, and D. Urosević. Variable neighbourhood decomposition search for 0-1 mixed integer programs. *Computers & Operations Research*, 37(6):1055–1067, 2010.

M. A. Lejeune. A variable neighborhood decomposition search method for supply chain management planning problems. *European Journal of Operational Research*, 175(2):959–976, 2006.

L. Liberti and M. Drazic. Variable neighbourhood search for the global optimization of constrained nlps. In *In Proceedings of GO Workshop, Almeria, Spain, 2005*, Almeria, Spain, 2005.

M. Locatelli and U. Raber. Packing equal circles in a square: A deterministic global optimization approach. *Discrete Applied Mathematics*, 122:139–166, 2002.

H. R. Lourenco, O. Martin, and T. Stutzle. *Iterated local search. In F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics*, chapter 11, pages 321–353. Kluwer Academic Publishers, Norwell, 1 edition, 2003.

Z. Lu and W. Huang. Perm for solving circle packing problem. *Computers & Operations Research*, 35:1742–1755, 2008.

B. D. Lubachevsky and R. L. Graham. Curved hexagonal packings of equal disks in a circle. *Discrete & Computational Geometry*, 18:179–194, 1997.

D. G. Luenberger and Y. Ye. *Penalty and barrier methods. In F. S. Hillier (ed), Linear and nonlinear programming*, chapter 13, pages 401–430. Standford University, Stanford, CA, USA, 3 edition, 2008a.

D. G. Luenberger and Y. Ye. *Primal-dual methods. In F. S. Hillier (ed), Linear and nonlinear porgramming*, chapter 15, pages 469–499. Standford University, Stanford, CA, USA, 3 edition, 2008b.

C. D. Maranas, C. A. Floudas, and P. M. Pardalos. New results in the packing of equal circles in a square. *Discrete Mathematics*, 142(1):287–293, 1995.

M. C. Markót. Optimal packing of 28 equal circles in a unit square-the first reliable solution. *Numerical Algorithm*, 37:253–261, 2004.

M. C. Markót. Interval methods for verifying structural optimality of circle packing configurations in the unit square. *Journal of Computational and Applied Mathematics*, 199(2): 353–357, 2007.

M. C. Markót and T. Csendes. A new verified optimization technique for the packing circles in a unit square problems. *SIAM Journal on Optimization*, 16(1):193–219, 2005.

M. C. Markót and T. Csendes. A reliable area reduction technique for solving circle packing problems. *Computing*, 77(2):147–162, 2006.

O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letter*, 11:219–224, 1992.

H. Melissen. Densest packing of eleven congruent circles in a circle. *Geom. Dedicata*, 50: 15–25, 1994.

P. Mills and E. Tsang. Guided local search for solving sat and weighted max-sat problems. *Journal of Automated Reasoning*, 24:205–223, 2000.

P. Mills, E. Tsang, and J. Ford. Applying an extended guided local search to the quadratic assignment problem. *Annals of Operations Research*, 118:121–135, 2003.

N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

N. Mladenović, J. Petrović, V. Kovacević-Vujcić, and M. Cangalović. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. *European Journal of Operational Research*, 151:389–399, 2003.

N. Mladenović, F. Plastria, and D. Urosević. Reformulation descent applied to circle packing problems. *Computers & Operations Research.*, 32:2419–2434, 2005.

N. Mladenović, F. Plastria, and D. Urosević. Formulation space search for circle packing problems. *Lecture Notes in Computer Science*, pages 212–216, 2007.

N. Mladenović, M. Drazić, V. Kovacevic-Vujcic, and M. Cangalovic. General variable neighbourhood search for the continuous optimization. *European Journal of Operational Research*, 191:753–770, 2008.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

A. G. Nikolaev, S. H. Jacobson, and A. W. Johnson. *The theory and practice of simulated annealling. In F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics*, chapter 10, pages 278–320. Kluwer Academic Publishers, Norwell, 1 edition, 2003.

K. J. Nurmela and P. R. J. Ostergard. Packing up to 50 equal circles in a square. *Discrete & Computational Geometry*, 18(1):111–120, 1997.

K. J. Nurmela and P. R. J. Ostergard. More optimal packings of equal circles in a square. *Discrete Comput. Geometry*, 22:439–457, 1999.

Z. Ognjanović, U. Midić, and N. Mladenović. A hybrid genetic and variable neighborhood descent for probabilistic sat problem. *Lecture Notes in Computer Science*, pages 36–42, 2005.

I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

I. H. Osman and G. Laporte. Metaheuristics: A bibiography. *Annals of Operations Research*, 63:513–623, 1996.

L. Pang, W. Lu, and H. J. Wang. Variance estimation in censored quantile regression via induced smoothing. *Computational Statistics and Data Analysis*, page Available online, 2010.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization - Algorithms and complexity*. Originally published: Englewood Cliffs, N. J., 1998.

M. I. Parzen, L. J. Wei, and Z. Ying. A resampling method based on pivotal estimation functions. *Biometrika*, 22(2):341–350, 1994.

C. A. P. Pinkse. On the computation of semiparametric estimates in limited dependent variable modelss. *Journal of Econometrics*, 58:185–205, 1993.

J. D. Pintér. Global optimization in action. *Kluwer Academic Publishers*, 1996.

U. Pirl. Der mindestabstand von n in der einheitskreisscheibe gelegenen punkten. *Math. Nachr.*, 40:111–124, 1969.

S. Portnoy. Asymptotic behavior of regression quantiles in non-stationary, dependent cases. *Journal of Multivariate analysis*, 38:100–113, 1991.

S. Portnoy. Censored regression quantiles. *Journal of American Statistical Association*, 98: 1001–1010, 2003.

S. Portnoy and G. Lin. Asymptotics for censored regression quantiles. *Journal of Nonparametric Statistics*, 22(1):115–130, 2010.

J. L. Powell. Least absolute deviations estimation for the censored regression model. *Journal of Econometrics*, 25:303–325, 1984.

J. L. Powell. Censored regression quantiles. *Journal of Econometrics*, 32:143–155, 1986.

W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Minimization or maxmization functions. In Numerical recipes in pascal: The art of scientific computing*, chapter 10, pages 309–374. Press syndicate of the University of Cambridge, Melbourne 3166, Australia, 1 edition, 1989.

J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classiffication. *In Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, ser. Lecture Notes in Computer Science, J. Mira and J. A lvarez, Eds. Berlin Heidelberg: Springer-Verlag*, 3562:41–53, 2005.

J. Puchinger and G. R. Raidl. Bringing order into the neighborhoods: Relaxation guided variable neighborhood search. *Journal of Heuristics*, 14(5):457–472, 2008.

J. Qian and L. Peng. Censored quantile regression with partially functional effects. *Biometrika*, 97(4):839–850, 2010.

R. S. Rajab, M. Drazić, and N. Mladenović. Reformulation descent for solving circle packing problems. In *Proceedings volume from the Information Control Problems in Manufacturing International Symposium*, Serbia, October 2011.

C. R. Rao and L. C. Zhao. Recent contributions to censored regression models. *American Sociological Review*, 42:203–213, 1995.

G. E. Reis. Dense packing of equal circle within a circle. *Mathematics Magazine*, 48(1): 33–37, 1975.

I Reiss. *Family systems in america*. Reinhart and Winston, New York, Holt, 1980.

S. Remde, P. Cowling, K. Dahal, and N. Colledge. Exact/heuristic hybrids using rvns and hyperheuristics for workforce scheduling. *Evolutionary Computation in Combinatorial Optimization,C. Cotta and J. van Hemert (Eds.), Springer-Verlag Berlin Heidelberg 2007*, 4446:188–197, 2007.

Wolfram Research. Mathematica release version 5.2. *Wolfram Research, Inc.*, 2005.

A. W. Roberts and D. E. Varberg. *Convex functions*. Academic press Inc., 1973.

H. E. Romeijn and R. L. Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5:101–126, 1994.

Z. Sevkli and F. Sevilgenr. A hybrid particle swarm optimization algorithm for function optimization. *Lecture Notes in Computer Science*, 4974:585–595, 2008.

D. M. Simmons. *Classical unconstrained optimization. In Nonlinear programming for operations research*, chapter 4, pages 101–143. Prentice-Hall, Inc., Englewood cliffs, N.J., 1 edition, 1975a.

D. M. Simmons. *Algorithm for nonlinear constrained problems. In Nonlinear programming for operations research*, chapter 8, pages 371–433. Prentice-Hall, Inc., Englewood cliffs, N.J., 1 edition, 1975b.

S. J. South and K. M. Lloyd. Spousal alternative and marital dissolution. *American Sociological Review*, 60:21–35, 1995.

M. C. De Souza and P. Martins. Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research*, 191(3):677–690, 2008.

T. Stutzle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174:1519–1539, 2006.

Lindo Systems. Lingo (release version: 9.0). *Lindo Systems, Inc.*, 2004.

P. G. Szabó, M. C. Markót, and T. Csendes. Global optimization in geometrycircle packing into the square. Technical Report 7, 2005.

P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L.G. Casado, and I. Garcia. *New approaches to circles packing in a square: with program code.* Springer science+business media, LLc, USA, 2007.

L. Tang and X. Wang. Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem. *International Journal of Advanced Manufacturing Technology*, 29:1246–1258, 2006.

Y. S. Teh and G. P. Rangaiah. Tabu search for global optimization of continuous functions with application to phase equilibrium calculations. *Computers and Chemical Engineering*, 27:1665–1679, 2003.

J. Tobin. Estimation of relationships for limited dependent variables. *Mathematical Analysis and Applications*, 26(1):24–36, 1958.

A. D. Toksari and E. Guner. Solving the unconstrained optimization problem by a variable neighbourhood search. *Mathematical Analysis and Applications*, 328:1178–1187, 2007.

A. M. Turing. On computable numbers with the application to the entscheidungsproblem. In *In Proceedings of London Mathematical society*, London, UK, 1936.

D. Urosević, J. Brimberg, and N. Mladenović. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31 (8):1205–1213, 2004.

E. R. van Dam, B. Husslage, D. den Hertog, and H. Melissen. Maximin latin hypercube designs in two dimensions. *Operations Research*, 55(1):158–169, 2007.

D. Vanderbilt and S. G. Louie. A monte carlo simulated annealing approach to optimization over continuous variable. *Journal of Computational Physics*, 65:259–271, 1984.

P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196:118–127, 2009.

S. VoB. Meta-heuristics: The state of art. *A. Nareyek (Ed.): Local search for plannning and scheduling. Springer-Verlag Berlin Heidlberg*, 2148:1–23, 2001.

C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

C. Voudouris and E. P. K. Tsang. *Guided Local Search. In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics*, chapter 7, pages 185–218. Kluwer Academic Publishers, Norwell, 1 edition, 2003.

C. Voudouris and E. P. K. Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of Second International Conference on Practical Application of Constraint Technology*, London, Uk, April 1996.

C. Voudouris and E. P. K. Tsang. Solving the radio link frequency assignment problem using guided local serach. In *Proceedings of NATO on symposium on frequency assignment, sharing and conservation in system (AEROSPACE)*, AGARD, 1998.

H. Wang, W. Huang, Q. Zhang, and D. Xu. An improved algorithm for the packing of unequal circles within a larger container circle. *Discrete optimization*, 141:440–453, 2002.

H. Wenqi and X. Ruchu. Two personification strategies for solving circles packing problem. *Science in China*, 42(6):179–186, 1999.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

R. S. Womersley. Censored discrete linear. *SIAM, Journal of Scientific and Statistical Computing*, 7:105–122, 1986.

M. Yazdani, M. Zandieh, and M. Amiri. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37:678–687, 2010.

K. Yu., Z. Lu, and J. Stander. Quantile regression: application and current. *The statistician*, 52:331–350, 2003.

S. H. Zanakis, J. R. Evans, and A. A. Vazacopoulos. Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, 43:88–110, 1989.

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.

Q. H. Zhao, D. Urosević, N. Mladenović, and P. Hansen. A restarted and modifed simplex search for unconstrained optimization. *Computer & Operations Research*, 36:3263–3271, 2009.

Y. Zhong and M. H. Cole. A vehicle routing problem with backhauls and time windows: a guided local search solution. *Transportation Science*, 41(2):131–144, 2005.

Appendix A

The idea of complexity theory has arisen in the beginning of 1930's. It was first introduced by Turing (1936). Moreover, the most important question that the computability theory is trying to answer is "if the problem can be solved algorithmically by using the computer or not". If the answer is "Yes", then the next question is "what are the minimum time and memory bit required?". Before going deep in this idea, some initial definitions should be introduced.

Let's assume that A is a finite alphabet and A^* is the set of infinite strings of elements of A. If $x \in A^*$, the length of x is given by $\lg(x)$.

Definition 4 *The deterministic algorithm A is specified by $D \subseteq R$, where D is a countable domain set and R is a countable range set, $\triangle$ is a finite alphabet where $\triangle^* \cap R = \phi$, E is an encoding function $E : D \to \triangle^*$ and τ is the transition function $\tau : \triangle^* \to \triangle^* \cap R$.*

The computation of A on input $x \in D$ is the unique sequence $y_1, y_2, \ldots$ such that $y_1 = E(x)$ and $y_{i+1} = \tau(y_i)\ \forall i$. Moreover, If the sequence is finite and ends with y_k, then $y_k \in R$.

Furthermore, the instantaneous description is a string occurring as an element of computation. The running time length $t(x)$ for algorithm A on input x can be defined when the computation of A on x is finite of length.

Definition 5 *A is terminating if all its computation are finite, where A is computing the function $f_A : D \to R$, like $f_A(x)$ is the last element of the computation of A on x.*

Definition 6 *The recognition algorithm A is happened if $R = \{accept, reject\}$. Moreover, if $D = \sum^*$ then A is called a string recognition algorithm, where the langue recognized by A*

is $\{x \in \sum^* \mid f_A(x) = accept\}$. *If* $D = R = \sum^*$ *the algorithm* A *is called a string mapping one.*

Definition 7 *The terminating algorithm* A *with* $D = \sum^*$ *operates on polynomial time if there is a polynomial* $p(.)$ *such that for each* $x \in \sum^*$ *then* $t(x) \leq p(\lg(x))$.

There are many well known classes of recognition algorithms like Markov algorithms, one-tap Turing machines and multitap and multihead Turing machines. Moreover, the class of language that recognize by polynomial algorithms is invariant under a wide range of changes in the class of algorithms.

Definition 8 P *is the class of languages recognizable by one-tap Turing machines which operate in polynomial time.*

The complexity class, denotes by P, is the most important class between complexity classes. The problems from class P can be solved by polynomial-time algorithms such as Turing machines.

Definition 9 *The function* $\prod : \sum^* \to \sum^*$ *is defined one-tape Turing machines which operate in polynomial time.*

Remark 1 *If* $f \in \prod$, *where* $f : \sum^* \to \sum^*$, *there is a polynomial* $p(.)$ *satisfied* $\lg(f(x)) \leq p(\lg(x))$.

Definition 10 *The* L *is reducible to* M $(L \propto M)$, *where* L, M *are languages, if there is a function* $f \in \prod$ *satisfied* $f(x) \in M \Leftrightarrow x \in L$.

Lemma 2 *If* $L \propto M$ *and* $M \in P$ *then* $L \in P$.

If P_2 denote the class of subset $\sum^* \times \sum^*$ which are recognizable in polynomial time. The language L is defined as follow $L = \{x \mid$ *there exists* y *that* $< x, y > \in L_2$ *and* $\lg(y) \leq p(\lg(x))\}$, where $L_2 \in P_2$ and p is a polynomial. Furthermore, L as a language is derived from L_2 by p-bounded existential quantification.

Definition 11 *NP is the set of language derived from elements of P_2 by polynomial-bounded existential quantification.*

Definition 12 *A nondeterministic algorithm A is specified by D the domina countable set, $\bigtriangledown$ a finite alphabet where ($\bigtriangledown^* \cap \{accept, reject\} = \phi$. The encoding function $E : D \rightarrow \bigtriangledown^*$ and the transition function $\tau \subseteq \bigtriangledown^* \times (\bigtriangledown^* \cup \{accept, reject\})$. For any $y_0 \in \bigtriangledown^*$ there is a set $\{< y_0, y > | < y_0, y > \in \tau\}$ has fewer than k_A element. k_A is a constant.*

Moreover, if $D = \sum^*$ then A is nondeterministic string recognition algorithm. That means the NP complexity class is for nondeterministic polynomial. Furthermore, the language from this class can be used by polynomial nondeterministic Turing machine.

Definition 13 *The language L is polynomial complete if*

- $L \in NP$.
- *Satisfiability* $\propto L$.

Theorem 3 *Either all complete language are in P, or non of them are. The former alternative holds if $P = NP$.*

Definition 14 *If D is a countable domain and $T \subseteq D$. e is one-one encoding function $e : D \rightarrow \sum^*$. Then T is complete if and only if $e(D)$ is complete.*

Appendix B

The continued Tables results for CPP-1 are presented in Table 5.1 and Table 5.2, where n denotes the number of non-overlap equal circles inside the circle container. The n is within the interval $[10, 200]$. The *Packonomia* denotes the best known results in Packonomia website (Hungarian, 2009). This website has two possibilities for each number of circles within the circle container, firstly the maximum radius of n non-overlap equal circles inside the unit circle container is denoted by *Packonomia* r. Secondly, the minimum radius of circle container to accommodate n unit circles without overlap is denoted by *Packonomia* R. VNS R denotes the minimum radius of circle container to accommodate n equal circles without overlap by using `GLOB-VNS` with the single formulation (4.3), where the next column in Tables is the one divided by VNS R. VNS r denotes the maximum radius for n equal circles within the unit circle container without overlap, where it has been found by `GLOB-VNS` with the single formulation (4.3) only.

The next two columns have been found by applying the `RD-VNS` algorithm, where $RD(R \rightarrow r)$ means that `RD-VNS` algorithm starts with the formulation (4.3) first, then it switches to the formulation (4.1) after the half of CPU time, and vice versa for $RD(r \rightarrow R)$. The last column CPU gives the running time in seconds.

Table 5.1: Circle packing problem inside a circle container from $n = 10$ till $n = 200$, continued table

n	*Packonomia r*	*Packonomia R*	*VNS R*	$1/R$	*VNS r*	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R)$	*CPU*
131	0.079053755252 48	12.649620461497 20	13.86268029177	0.072136122233	0.076825549693	0.077463461087	13.651785172635	0.073250493423	120
132	0.078818126663 85	12.687436790585 80	14.41272200112	0.069383146357	0.076496942486	0.078692741472	13.775183018823	0.072594316797	130
133	0.078522069613 22	12.735273088519 00	14.28621872017	0.069997528358	0.077011764096	0.077214701403	14.069511550390	0.071075672842	180
134	0.078299667964 46	12.771446239770 40	14.07530197739	0.071046433079	0.075997985875	0.076599832724	14.032832492579	0.071261450639	180
135	0.078038092564 34	12.814254771482 40	13.98304597953	0.071515176412	0.075862225345	0.076322372389	13.852014951008	0.072191663345	240
136	0.077725687011 40	12.865759550678 80	14.06973088320	0.071074564844	0.075151271789	0.075911506439	13.967781136795	0.071593332556	240
137	0.077430991968 43	12.914725416507 10	14.12810373760	0.070780907231	0.075141642885	0.075825392603	14.086812556170	0.070988379806	240
138	0.076869934722 46	12.962702608082 20	14.46693815603	0.069123126761	0.075127135484	0.075642278155	13.993024550596	0.071464178197	240
139	0.076869934722 46	13.008987240727 30	14.42535420443	0.069322387917	0.074214931049	0.075443681603	14.290134088767	0.069978349663	240
140	0.076563246067 77	13.061097215168 80	14.14405724274	0.070701071329	0.075335656007	0.075305524891	14.039955169249	0.071225298653	240
141	0.076293623454 62	13.107255294995 60	14.42646906100	0.069317030784	0.073961876585	0.074594108741	14.357154004225	0.069651687215	240
142	0.076066384385 64	13.146411625538 50	14.49359145813	0.068996011298	0.074007332781	0.074778002460	14.096272773439	0.070940738454	240
143	0.075772495907 46	13.197400824980 90	14.37733914828	0.069553899347	0.073110005846	0.074795793553	14.043050047212	0.071209601663	240
144	0.075484292738 20	13.247789225080 80	15.03771697901	0.066499456094	0.072520036785	0.073091846001	14.279198080586	0.070031943976	300
145	0.075320099332 05	13.276668629863 50	14.67990686150	0.068120323203	0.071741291073	0.073492326655	14.109037142114	0.070876558756	300
146	0.075011811649 19	13.331233815238 50	14.53339033614	0.068807069574	0.073640162388	0.073752882181	14.312750238521	0.069867774071	300
147	0.074866480341 07	13.357112494727 50	14.42438568633	0.069327042534	0.071976689422	0.073732513794	14.326171000331	0.069802321917	300
148	0.074699673575 15	13.386939355148 70	14.76199754234	0.067741509720	0.072917010098	0.068900143293	14.498755397840	0.068971437379	300
149	0.074429413773 86	13.435548518285 10	14.69430588315	0.068053571768	0.071167730606	0.073237133839	14.600127930789	0.068492550527	300
150	0.074289754450 12	13.460806370969 70	15.98356018918	0.062564284062	0.071958439572	0.072580299935	14.738252696693	0.067850648281	300
151	0.074204941313 10	13.476191508333 60	14.74498585737	0.067819664913	0.072122545873	0.072404819577	14.426571476292	0.069316538697	300
152	0.073900225253 74	13.531758483366 00	14.75664585292	0.067766077059	0.071035566861	0.071848321108	14.666117385955	0.068184371752	300
153	0.073575197788 35	13.591536687086 70	14.87037221270	0.067247812341	0.069591188753	0.072121891963	14.879223215500	0.067207809542	300
154	0.073327659015 59	13.637418859743 70	15.00138893549	0.066660494192	0.071714002231	0.071805942882	14.722883834565	0.067921475931	420
155	0.073133048706 55	13.673708640433 10	15.06864776888	0.066362955412	0.071273011357	0.067614993120	14.686250493383	0.068090899066	420
156	0.072888422190 63	13.719600039970 70	15.80008962278	0.063290780234	0.070440228850	0.070258263025	14.925628611518	0.066998853183	420
157	0.072605865685 13	13.772991900360 70	15.18276910091	0.065864138047	0.070628993521	0.069799899112	14.926176312139	0.066996394729	420
158	0.072342013798 32	13.823225916655 60	15.20831557504	0.065753501436	0.070229297570	0.070086561490	14.715547389562	0.067955338223	420
159	0.072128248469 74	13.864193588723 60	15.15978992322	0.065963974769	0.070589996835	0.069097522402	14.786726609671	0.067628219984	420
160	0.071836300857 98	13.920538614273 10	15.15609461022	0.065980057905	0.069862428003	0.070386089074	15.197310655905	0.065801115911	420
161	0.071585034482 45	13.969400269622 20	14.96776076071	0.066810260799	0.068900391192	0.070108380122	14.858991671154	0.067299317621	420
162	0.071370819596 48	14.011328518487 50	15.27512756336	0.065465901732	0.069779148235	0.069938151525	15.101403330455	0.066219011447	420
163	0.071075123894 43	14.069620216003 50	15.50796550353	0.064482991000	0.068749783846	0.065845403893	15.162582705181	0.065951824926	420
164	0.070868206161 86	14.110700046733 30	15.25724775260	0.065542620544	0.068156699074	0.069517375018	15.153908532040	0.065985976081	420
165	0.070699661492 23	14.144339292344 00	15.48223716224	0.064590148667	0.066747568692	0.069813844110	15.265733334126	0.065506188148	420
166	0.070493632407 11	14.185678420213 80	15.30647866895	0.065331812864	0.069231186521	0.069711148936	15.200124876437	0.065788933192	420
167	0.070321537176 84	14.220394492874 20	15.46365501757	0.064667764436	0.068090967308	0.069353628368	15.304969534052	0.065338254857	420
168	0.070128511372 51	14.259535359555 30	15.41572662715	0.064868820276	0.067741509459	0.068520282088	15.193667889601	0.065816892094	480
169	0.069952776560 23	14.295358228403 10	15.55213242186	0.064299864023	0.067410377119	0.068024900822	15.509140601013	0.064478105249	480
170	0.069776124656 32	14.331549723139 60	16.26486945088	0.061482202671	0.067012269679	0.066891170052	15.710918087149	0.063650004058	480

Table 5.2: Circle packing problem inside a circle container from $n = 10$ till $n = 200$, continued table

n	*Packonomia r*	*Packonomia R*	*VNS R*	$1/R$	*VNS r*	$RD(R \to r)$	$RD(r \to R)$	$1/(RD(r \to R)$	*CPU*
171	0.069600749130 16	14.367661447578 80	15.776153974 75	0.063386805276	0.067452976249	0.067988808506	15.459538305371	0.064684984781	600
172	0.069363635406 18	14.416776083665 90	15.956476101 68	0.062670478972	0.068452864992	0.066347261931	15.703997631396	0.063678053415	600
173	0.069197728416 11	14.451344136912 30	15.523012819 32	0.064420484067	0.067222190871	0.067113852043	15.534387885386	0.064373312124	600
174	0.068999546681 58	14.492848838772 50	15.547111330 08	0.064320630294	0.066903789202	0.065863450717	15.623569487875	0.064005859914	600
175	0.068792158147 08	14.536540601939 00	16.001608772 23	0.062493716365	0.065311081127	0.067063341685	15.718948352761	0.063617487478	600
176	0.068614703724 41	14.574135654894 50	15.789919175 70	0.063331546468	0.066551802912	0.063423936585	15.759400315881	0.063454191147	600
177	0.068412582430 25	14.617194154592 20	15.901948331 08	0.062885376004	0.063774211220	0.066842655542	15.831963011739	0.063163361313	600
178	0.068217889861 16	14.658911350602 70	15.964994904 49	0.062637038470	0.066582033669	0.067115716588	15.893903719393	0.062917205090	600
179	0.068016599535 37	14.702293364135 60	15.861927886 99	0.063044038980	0.065219347682	0.066120631881	15.866689843145	0.063025118023	600
180	0.067829361242 65	14.742878035112 80	16.020128128 64	0.062421473285	0.065487786953	0.066019972036	15.845419552649	0.063109720552	600
181	0.067655200908 09	14.780829656518 00	15.975343245 05	0.062596464105	0.065893155303	0.066485886721	16.071929911826	0.062220281291	600
182	0.067481252520 33	14.818930631121 80	16.258977733 60	0.061504481794	0.063481913424	0.066023622773	15.891401219263	0.062927112984	600
183	0.067252213487 68	14.869399059751 60	16.123908934 66	0.062019700313	0.064932927284	0.066060312367	16.247684843651	0.061547230244	600
184	0.067089452837 79	14.905472584756 80	16.350566036 74	0.061159962154	0.064513026547	0.066287082127	16.299839235985	0.061350298339	600
185	0.066942716851 00	14.938144835469 30	16.205722463 39	0.061706597917	0.065297000861	0.065726434870	16.336194992020	0.061213764924	600
186	0.066839622261 17	14.961185688521 10	16.135278918 00	0.061975997135	0.064897480400	0.065477414577	16.226282415678	0.061628410894	600
187	0.066714391021 83	14.989269701537 10	16.135278918 00	0.061975997135	0.064499184757	0.065682031902	16.411563201467	0.060932647775	600
188	0.066714391021 83	15.028785151972 80	16.384108738 67	0.061034751170	0.062921577862	0.065346742890	16.178914868484	0.061808842443	600
189	0.066358407244 02	15.069680565458 10	16.184619995 26	0.061787054642	0.063565586433	0.064834845175	16.435255786503	0.060844809049	600
190	0.066209373949 94	15.103601504464 10	16.220246575 26	0.061651343915	0.062104371990	0.064821338444	16.152257877884	0.061910849094	660
191	0.066039914725 34	15.142357529668 60	16.333073487 71	0.061225463827	0.063491871954	0.064345321158	16.324587649745	0.061257290013	660
192	0.065922032429 18	15.169435212335 70	16.486084246 39	0.060657217630	0.063510214440	0.063984124946	16.428281692685	0.060870638738	660
193	0.065792181138 71	15.199374495454 00	16.474896945 17	0.060698406996	0.063042597248	0.062093058615	16.455352624657	0.060770499594	780
194	0.065574830071 95	15.249753585373 40	16.603584928 60	0.060227957053	0.063944032543	0.064132548056	16.472140842504	0.060708562995	780
195	0.065414321826 15	15.287172167857 20	16.710744793 33	0.059841737299	0.062606892054	0.063838153506	16.623120926262	0.060157175324	780
196	0.065261031800 23	15.323079829032 50	17.027218536 19	0.058729498178	0.059816092971	0.063375378772	16.496579728181	0.060618626193	780
197	0.065066146625 47	15.368975294574 60	16.846281081 83	0.059360282257	0.064208154605	0.063986854923	16.561514287094	0.060380952047	810
198	0.064972158738 91	15.391207855940 52	16.654469551 62	0.060043941772	0.063283188001	0.062936598224	16.804761359455	0.059506944408	840
199	0.064925535151 15	15.402260415288 50	16.758121485 94	0.059672559412	0.063061159778	0.060703114830	16.666319276341	0.060001250631	840
200	0.064669354186 26	15.463274878542 00	17.192206906 73	0.058165889081	0.062826910321	0.063545183777	16.625736734130	0.060147710504	840

The continued Tables results for CPP-2 are presented in Table 5.3 and Table 5.4, where n denotes the number of non-overlap equal circles inside the circle container. The n is within the interval $[10, 200]$. The *Pack* r denotes the best known results in Packonomia website (Hungarian, 2009) for finding the maximum radius of n equal circles inside the unit square container without overlap. $VNS\ L$ denotes the minimum length of the side container of the square to accommodate n nonoverlaping unit circles by using `GLOB-VNS` with single formulation (4.5), where the next column in Tables is the one divided by $VNS\ L$. $VNS\ r$ denotes the maximum radius for n equal circles within the unit square container without overlap, where it has been found by `GLOB-VNS` with the single formulation (4.3) only.

The next two columns have the results of applying the `RD-VNS` between the formulation (4.4) and the formulation (4.5), where $RD(L \rightarrow r)$ means that `RD-VNS` algorithm starts with the formulation (4.5) first, then it switches to the formulation (4.4) at the middle of the computational time, and vice versa for $RD(r \rightarrow L)$. The last columns CPU give the running time in seconds, where $CPU1$ denotes the computational time for using the formulation (4.5), and $CPU2$ denotes the computational time for using the formulation (4.4). $CPU3$ denotes the computational time for finding RD between (4.4) and (4.5) starting with (4.5), and $CPU4$ denotes the computational time for finding RD between (4.4) and (4.5) starting with (4.4).

Table 5.3: Circle packing problem inside a square container CPP-2 for $n = 10, \ldots, 200$, continued table

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \to r)$	$RD(r \to L)$	$1/(RD(r \to L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
131	0.0448872324	24.18424685	0.041349231	0.043870854	0.043688769	23.24466087	0.043020632	660	300	840	900
132	0.0447463709	24.34592477	0.041074636	0.043749819	0.042801685	24.03256876	0.0416102	660	300	840	900
133	0.0446096416	24.00475183	0.041658419	0.043261106	0.042905652	23.49810013	0.042556632	720	300	840	900
134	0.0445642442	24.0642323	0.04155545	0.043352312	0.042073789	25.24403144	0.039613324	720	300	960	960
135	0.0444311981	24.33547551	0.041092273	0.043060991	0.042401784	27.95152115	0.035776228	720	300	960	900
136	0.0443533026	24.39023305	0.041000018	0.043161828	0.042420167	26.33351393	0.037974423	780	300	960	1020
137	0.0442930235	24.08703153	0.041516116	0.043205738	0.040751435	24.12473043	0.04145124	780	300	960	1280
138	0.0440321362	24.59575175	0.040657428	0.041024184	0.042840687	23.36839988	0.042792832	780	300	960	1280
139	0.0439457424	24.38416045	0.041010229	0.042692218	0.041418274	24.44168312	0.040913713	900	300	1080	1280
140	0.0438048391	24.40253577	0.040979348	0.042902122	0.040560096	23.91045252	0.041822713	900	360	1080	1280
141	0.0436584692	24.41518342	0.04095812	0.042685473	0.04109611	24.97401924	0.040041612	900	360	1140	1280
142	0.0435766327	24.65511815	0.04055953	0.041856527	0.041237864	24.02128265	0.04162975	1800	360	1080	1280
143	0.0435305119	24.56539694	0.040707667	0.043452712	0.041098856	23.95582212	0.041743506	1200	360	1080	1280
144	0.0430037655	25.00364457	0.03999417	0.041800768	0.04058585	25.10886492	0.039826571	1200	360	1080	1280
145	0.0428713567	25.00230525	0.039996312	0.041774465	0.039167089	25.19934158	0.039683577	1200	360	1200	1280
146	0.0426512861	26.00023991	0.038461184	0.040206169	0.042563342	26.01485484	0.038439576	1200	360	1200	1280
147	0.0424469914	25.54232465	0.039150704	0.041161253	0.039604421	26.39950752	0.037879494	1200	360	1200	1280
148	0.0423302208	25.02658259	0.039957513	0.04106549	0.040867469	25.54073615	0.039153139	1200	360	1200	1400
149	0.0422518627	26.25242264	0.038091723	0.040603009	0.039829766	25.91375113	0.03858955	1200	360	1200	1400
150	0.0421454659	26.01946593	0.038432764	0.040926021	0.042004747	25.61730497	0.039036113	1200	420	1320	1400
151	0.0419770971	25.05129696	0.039918093	0.041142603	0.039904489	27.73571626	0.036054594	1200	420	1320	1400
152	0.0417798031	25.75798762	0.038822909	0.040718968	0.040000142	25.63266068	0.039012727	1200	420	1380	1520
153	0.0416489333	26.00955829	0.038447404	0.040111672	0.038378831	25.32241227	0.039490708	1200	420	1320	1520
154	0.0415519299	24.99996094	0.040000063	0.040724225	0.039119413	26.05128127	0.038385828	1200	420	1320	1520
155	0.0414380393	25.38771314	0.039389133	0.039547927	0.039515604	25.8891135	0.038626274	1200	480	1320	1520
156	0.0412993685	26.01103867	0.038445216	0.040253604	0.040597059	25.85946034	0.038670567	1200	480	1780	1800
157	0.0412203492	26.18000518	0.038197089	0.039783088	0.040000514	25.40269723	0.039365898	1200	480	1620	1800
158	0.0411596935	26.16448602	0.038219746	0.04014969	0.038768216	26.25146255	0.038093116	1200	480	1620	1800
159	0.0411141834	26.06435208	0.038366578	0.039851739	0.038764834	26.15612393	0.038231964	1200	480	1620	1800
160	0.0410445958	26.46886919	0.037780231	0.039733908	0.036399015	26.10806198	0.038302345	1200	480	1620	1800
161	0.0410103478	24.99999964	0.040000001	0.039802607	0.036208179	25.95681065	0.038525534	1200	480	1780	1800
162	0.0407991126	26.12011793	0.038284666	0.039592499	0.03933761	26.04583819	0.03839385	1320	480	1900	1920
163	0.0406812044	26.34728869	0.03795457	0.037056974	0.038272223	26.58015349	0.037622055	1320	480	1900	1920
164	0.0405250524	41.43109882	0.024136459	0.039616855	0.037809377	26.0672999	0.038362239	1800	480	2140	1920
165	0.0403775441	29.73107764	0.033634839	0.03842006	0.038018047	26.49215453	0.037747024	1320	480	2020	1920
166	0.0402377276	26.67774513	0.037484427	0.039695974	0.037624366	26.1393237	0.038256537	1320	600	2020	1920
167	0.0401484414	27.01845955	0.037011733	0.038771457	0.036566527	26.7128929	0.037435107	1320	600	2020	1920
168	0.0401024007	24.999375	0.040001	0.039117374	0.038562409	26.36603641	0.037927582	1320	600	2140	1920
169	0.0398053930	28.20187026	0.035458641	0.038552163	0.038207864	27.75364212	0.036031307	1320	600	2140	2140
170	0.0395995814	28.0702391	0.035624919	0.038448119	0.038472457	27.59189581	0.036242526	1320	600	2140	2140

Table 5.4: Circle packing problem inside a square container CPP-2 for $n = 10, \ldots, 200$, continued table

n	$Pack\ r$	$VNS\ L$	$1/L$	$VNS\ r$	$RD(L \to r)$	$RD(r \to L)$	$1/(RD(r \to L))$	$CPU1$	$CPU2$	$CPU3$	$CPU4$
171	0.0394857333	28.05720046	0.035641475	0.038523594	0.03654344	27.40090685	0.036495143	1500	600	2140	2140
172	0.0393433077	26.99992188	0.037037144	0.038583497	0.03842254	27.21591492	0.036743207	1500	600	2140	2320
173	0.0392515335	28.24641981	0.0354027717	0.037946503	0.037624462	27.67295985	0.036136359	1500	600	2260	2320
174	0.0391795430	26.9996875	0.037037466	0.037297099	0.037637301	28.45008625	0.035149278	1500	600	2260	2320
175	0.0390610992	27.43337429	0.03645195	0.037765422	0.038583101	28.19414767	0.035468354	1500	660	2260	2320
176	0.0389908134	28.05045629	0.035650044	0.037349292	0.037152874	27.36239696	0.036546506	2700	660	2260	2320
177	0.0388595533	28.1003785	0.0355867709	0.03740119	0.038118889	27.95061532	0.035777388	2700	660	3600	2320
178	0.0387305625	28.26977169	0.035373473	0.037977391	0.038222475	27.65249966	0.036163096	2700	660	2380	2320
179	0.0386825683	28.05272779	0.035647157	0.037483868	0.038565037	28.04468951	0.035657375	3600	660	2380	2700
180	0.0386201119	28.03712173	0.035666999	0.037626283	0.033780416	28.10434823	0.035581683	3600	720	2380	2700
181	0.0384828248	28.0044273	0.03570864	0.037703148	0.037606356	27.82492838	0.035938996	3600	720	3600	2700
182	0.0384176810	28.0270189	0.035679856	0.037091433	0.036738131	28.06492534	0.035631664	3600	720	3600	2700
183	0.0383548074	28.00082586	0.035713232	0.037636978	0.037406093	27.78640165	0.035988827	3600	720	3600	2700
184	0.0382816728	27.96680155	0.035756681	0.036511437	0.037413102	27.83618668	0.035924461	3600	720	3600	2700
185	0.0382452488	28.13535257	0.035542473	0.035377263	0.036859383	27.85207828	0.035903963	3600	720	3600	2700
186	0.0382338904	28.42721221	0.035177561	0.036083152	0.036488284	28.56049701	0.035013396	3600	780	3600	2700
187	0.0382010645	28.22848234	0.035425213	0.036700757	0.034781194	28.51225823	0.035072634	3600	780	3600	2700
188	0.0381886231	26.9996875	0.037037466	0.036982065	0.036206572	28.97844603	0.034508407	3600	780	3600	2700
189	0.0378665527	28.18498562	0.0354798883	0.037072043	0.036137305	28.06277854	0.03563439	3600	780	3600	2700
190	0.0376960000	28.10983869	0.035574733	0.03510182	0.03431802	28.55169951	0.035024185	3600	780	3600	2700
191	0.0375441276	26.99984375	0.037037251	0.036021519	0.035982691	28.83984047	0.034674256	3600	780	3600	3000
192	0.0374443353	28.5602534	0.035013695	0.035727841	0.035904898	28.8423244	0.034671269	3600	780	3600	3000
193	0.0373240521	26.99984375	0.037037251	0.035677205	0.036137168	28.79115621	0.034732888	3600	780	3600	3180
194	0.0372576486	26.99992188	0.037037144	0.034978076	0.035293005	28.99936734	0.034483511	4500	780	3600	3180
195	0.0372133686	28.38251697	0.035232957	0.035774974	0.035842804	28.63625712	0.034920765	5400	780	3600	3180
196	0.0370473729	29.17940691	0.034270745	0.034526554	0.034482898	29.9590901	0.033378851	5400	840	3600	3180
197	0.0368716728	29.86007945	0.033489529	0.034872559	0.035102117	29.86191152	0.033487474	5400	840	3600	3180
198	0.0367716872	30.00416764	0.033328703	0.035486892	0.035614497	30.24052393	0.03306821	5400	840	5400	3180
199	0.0366674250	29.00524084	0.034476528	0.035745422	0.035999174	29.54877033	0.033842356	5400	840	5400	3300
200	0.0366127989	30.21075778	0.033100792	0.034179713	0.035579861	30.02805252	0.033302193	5400	840	5400	3300