General variable neighborhood search for the continuous optimization

Nenad Mladenović a,*, Milan Dražić b, Vera Kovačević-Vujčić c, Mirjana Čangalović c

a School of Mathematics, Brunel University, West London, UK
b Faculty of Mathematics, University of Belgrade, Serbia and Montenegro
c Faculty of Organizational Sciences, University of Belgrade, Serbia and Montenegro

Received 15 April 2006; accepted 15 December 2006
Available online 18 February 2007

Abstract

We suggest a new heuristic for solving unconstrained continuous optimization problems. It is based on a generalized version of the variable neighborhood search metaheuristic. Different neighborhoods and distributions, induced from different metrics are ranked and used to get random points in the shaking step. We also propose VNS for solving constrained optimization problems. The constraints are handled using exterior point penalty functions within an algorithm that combines sequential and exact penalty transformations. The extensive computer analysis that includes the comparison with genetic algorithm and some other approaches on standard test functions are given. With our approach we obtain encouraging results.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Global optimization; Nonlinear programming; Metaheuristics; Variable neighborhood search

1. Introduction

Here we consider the continuous global (nonlinear) optimization problems (CGOP for short) in a general form

\[ \text{global } \min_{x \in S} f(x) \quad (1) \]

with the feasible set \( S \) defined by

\[ g_i(x) \leq 0, \quad i = 1, 2, \ldots, m, \quad (2) \]

\[ h_i(x) = 0, \quad i = 1, 2, \ldots, r, \quad (3) \]

\[ a \leq x \leq b, \quad (4) \]
where \( x \in \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}, g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, 2, \ldots, m, h_i : \mathbb{R}^r \to \mathbb{R}, i = 1, 2, \ldots, r, \) are possibly nonlinear continuous functions, and \( a, b \in \mathbb{R}^n \) are the variable bounds.

Problems of global optimization naturally arise in many applications, e.g. in the advanced engineering design, data analysis, financial planning, risk management, scientific modelling, etc. Most cases of practical interest are characterized by multiple local optima and, therefore, in order to find a globally optimal solution, a global scope search effort is needed. Generally speaking there are two types of methods for tackling CGOP problem (1)–(4) (for a general survey see Pardalos and Romeijn, 2002; Pardalos and Rosen, 1987; Pinter, 1996; Torn and Zilinskas, 1989):

(i) **The exact methods** include complete enumerative search strategies (applicable to certain well-structured problems of type (1)–(4), such as the concave programming), homotopy methods and related approaches, successive approximation methods (applicable again to certain structured global optimization problems), branch and bound algorithms (applicable to broad classes of problem (1)–(4), such as the concave programming, D.C. programming, Lipschitz optimization, etc.), and others.

(ii) **The heuristic methods** do not offer a guarantee of locating the global optimum, but give satisfactory results for much larger range of global optimization problems. Recently, so-called metaheuristics, or frameworks for building heuristics (originally developed for solving combinatorial optimization problems) became popular for solving CGOP as well. Let us mention some of them. The multistart local search (MLS) consists of generating a set of random solutions and using them as starting points of some conventional minimization technique converging to a local minimum. In order to achieve a sufficiently high probability of finding a global minimum, a large number of starting points must be tried out. This strategy is time consuming and soon becomes intractable as the dimensionality of the problem increases. Some of the best known metaheuristic approaches use different techniques in order to avoid entrapments in poor local minima. The methods of this type are the simulated annealing, genetic algorithms, tabu search, variable neighborhood search, etc. The simulated annealing and the genetic algorithms imitate some spontaneous optimization processes in physical systems and nature respectively, while the tabu search and the variable neighborhood search (VNS) belong to intelligent problem-solving algorithms.

In this paper we develop a general VNS heuristic for solving CGOP. We call it continuous general VNS (CGVNS for short). It differs from our previous work on the VNS methodology for CGOP (Kovacevic-Vujicic et al., 2004; Drazic et al., 2006a,b) in the following two basic issues:

(i) **General vs. basic VNS.** In our previous approach we made a list of basic VNS heuristics, each of them using neighborhoods induced from the single metric function; such VNS heuristics were iterated until some stopping condition was met. Here we suggest a general VNS scheme for solving CGOP; neighborhoods having the same distance from the incumbent, but induced from the different metric functions are considered simultaneously; in that way we use the same ingredients as before, but within one, general VNS scheme. In Section 5 we show that our new CGVNS compares favorably with the old basic VNS approach as well as with other approaches from the literature.

(ii) **Constrained VNS vs. unconstrained VNS.** Before we were restricted to unconstrained and box-constrained nonlinear programs (1) and (4). The aim of this paper is to develop the CGVNS heuristic for solving general constrained nonlinear programming problem (1)–(4) and to investigate its potential. Comparative analysis given in Section 5 shows that VNS idea works very well in solving constrained global optimization problem as well.

The paper is organized as follows. In Section 2 we give an outline of VNS methodology for continuous optimization. Sections 3 and 4 describe general VNS algorithms for box-constrained and generally constrained CGOP, respectively. Extensive computer analysis that includes comparison with publicly available GENOCOP III software (based on genetic search), and with some other approaches are reported in Section 5. It also includes comparison with our previous VNS version on box constrained instances. Section 6 concludes the paper.

2. VNS for continuous optimization

**Basic VNS.** Variable neighborhood search (Mladenovic and Hansen, 1997) is one among metaheuristics designed for solving optimization problems. It exploits systematically the idea of neighborhood change, both in the descent to local minima and in the escape from the valleys which contain them.
It has been applied in many areas, mostly for solving combinatorial optimization problems. The basic steps of VNS metaheuristic are given in Fig. 1.

For the numerous successful applications of VNS see e.g. the survey papers (Hansen and Mladenović, 2001a,b, 2003). General VNS is the variant of VNS that we develop here for solving CGOP. General VNS is successfully applied in solving several optimization problems. Let us mention two recent ones: mixed integer programming problem (Hansen and Mladenović, 2006) and vertex weighted $k$-cardinality tree problem (Brimberg et al., 2006). There the deterministic VNS (variable neighborhood descent or VND for short) is used as the local search in step 2(b).

Previous work on continuous VNS. The multisource Weber was the first continuous optimization problem suggested to be solved by VNS in Brimberg and Mladenović (1996) and Brimberg et al. (2000). Another continuous problem attacked by VNS was the general bilinear programming problem (Hansen and Mladenović, 2001a) (and its special case pooling problem, the typical one from the oil industry (Audet et al., 2004)). However, although both of those problems are defined in continuous variables, the basic move is in fact combinatorial. For example, in the multisource Weber problem the neighborhood points (of the continuous solution) are defined by re-locating facility points or by re-allocating customers; thus the number of neighboring solutions is always finite.

The rules of VNS for solving a “pure” continuous optimization problem are for the first time suggested in Mladenović et al. (2003b). There we consider polyphase radar code design, the unconstrained nonlinear problem that has specific minimax objective function. Later we developed the software package GLOB for general box-constrained nonlinear programs (Mladenović et al., 2003a; Kovacević-Vujčić et al., 2004). For the local search phase of VNS we included several nonlinear programming methods: steepest descent, Rosenbrock, Nelder-Mead, Fletcher-Reeves, etc. It is specified by the user what method among those will be run. In the shaking step we used rectangular norm for defining neighborhoods in $R^n$. The advanced version of GLOB, again for solving box-constrained CGOP, was suggested in Dražić et al. (2006a). There we consider several basic VNS heuristics, each using different metric function in designing neighborhoods for the shaking step. Users are allowed to choose any combination of those heuristics. For each heuristic (metric) we perform the search with $k_{\text{max}}$ (a VNS parameter) neighborhoods. In each neighborhood a random starting point for a local search is generated according to the chosen metric. For finding three dimensional structure of the molecule, that is shown to be an unconstrained NLP problem in Lavor and Maculan (2004), we observed that the uniform distribution for generating points at random in the shaking step is not necessarily the best choice (Dražić et al., 2006b); the specially designed distribution allows us to get more initial points for descents closer to axial directions and much better results in terms of computational efforts. Therefore, we have added another important parameter to the continuous VNS: probability distribution used
for obtaining the random point in the shaking step. Throughout the paper we denote this latest published VNS version as GLOB.

In the next two sections we describe how we design the general VNS for solving unconstrained and constrained global optimization problems, respectively. This new VNS version proposed in this paper we denote with CGVNS.

3. Continuous general VNS for unconstrained optimization

Basic notation. The idea of VNS is to define a set of neighborhood structures \( N_k, k = 1, \ldots, k_{\text{max}} \), that can be used in a systematic way to conduct a search through the solution space. Whereas in local search a single neighborhood is typically defined \( (k_{\text{max}} = 1) \), the VNS expands the search over an increasing radius to escape from a “local optimum trap”.

The neighborhood \( N_k(x) \) denotes the set of solutions in the \( k \)th neighborhood of \( x \), and using the metric \( \rho_k \), it is defined as

\[
N_k(x) = \{ y \in S | \rho_k(x, y) \leq r_k \},
\]

or

\[
N_k(x) = \{ y \in S | r_{k-1} \leq \rho_k(x, y) \leq r_k \},
\]

where \( r_k \) is the radius (size) of \( N_k(x) \) monotonically nondecreasing with \( k \). Observe that the same value of the radius can be used in several successive iterations. In other words each neighborhood structure \( N_k \) is defined by pair \( (\rho_k, r_k), k = 1, \ldots, k_{\text{max}} \). Note that we are allowed to use the same metric for different \( k \), e.g., \( \rho_3 \) could be equal to \( \rho_7 \). Another important observation is that while in the discrete case \( N_k(x) \) is finite, here it contains an infinite number of points. The metric functions are defined in an usual way, i.e., as \( \ell_p \) distance:

\[
\rho_k(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty)
\]

or

\[
\rho_k(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p = \infty).
\]

Parameters. In designing general VNS heuristic for solving CGOP one should take into account the following:

- Maximal running time \( t_{\text{max}} \) allowed for the search.
- Number of neighborhood structures \( k_{\text{max}} \) used in the search;
- Values of radii \( r_k \), \( k = 1, \ldots, k_{\text{max}} \). Those values may be defined by the user or calculated automatically during the search.
- Geometry of neighborhood structures \( N_k \), defined by the choice of metrics. Usual choices are \( \ell_1 \), \( \ell_2 \), and \( \ell_{\infty} \).
- Distributions used for obtaining the random point \( y \) from \( N_k(x) \) in the shaking step. Uniform distribution in \( N_k(x) \) is the obvious choice, but other distributions may lead to much better performance on some problems.
- Local optimizer used in the local search step. Usually the choice of the local optimizer depends on the properties of the objective function. A lot of local optimization algorithms are available both for smooth and non differentiable functions.
- Ordering of neighborhoods and distributions in the shaking step.

In designing VNS heuristic it is possible to reduce the number of parameters which the user must provide. This can be accomplished by fixing some parameters in advance (e.g. the number of points generated at random, geometry, distributions and their ordering) and automatically calculating values of radii (e.g. uniformly distributed from 0 to the diameter of the set \( S \)). In that case the user should specify only three parameters: \( t_{\text{max}}, k_{\text{max}} \) and the choice of local search procedure.
Algorithm. The general VNS algorithm presented below incorporates various VNS-based heuristics for unconstrained (i.e. box-constrained) continuous global optimization developed in Kovacˇevic-Vujcˇic´ et al. (2004). As stated before, in our previous work we made a list of VNS heuristics, each of them using neighborhoods induced from the single metric function; such VNS heuristics were iterated until some stopping condition was met. Here we use the same ingredients as before, but in different order and within one heuristic: neighborhoods having the same distance from the incumbent, but induced from the different metric functions are considered simultaneously. The pseudo-code of continuous general VNS is outlined in Fig. 2.

In the implementation of CGVNS we have used the neighborhood structures defined by metrics \( \ell_1 \) and \( \ell_\infty \) and predefined radii \( r_1 < \cdots < r_{k_{\text{max}}} \). The pairs \((\ell_1, r_k), (\ell_\infty, r_k), k = 1, \ldots, k_{\text{max}}\) determine neighborhood structures according to (5) or (6), where \( \rho_k \) is either \( \ell_1 \) or \( \ell_\infty \).

We have implemented four different distribution types for generating a random point in the shaking step. The random point is obtained in two steps: first a random direction \( d \) is generated by using one of the distributions listed below and after that a random radius \( r \in [0, r_k] \) (or \([r_{k-1}, r_k]\)) with density function proportional to the measure of the \( n \)-dimensional sphere is determined. Finally, the direction \( d \) is scaled to \( r \) in metric defining the geometry of \( N_k(x) \). If the obtained random point lies out of box constraints, it is mapped symmetrically w.r.t. the boundary into the box region. Now we give four possible ways to generate points from the continuous region:

- **Distribution \( D_1 \):** The direction \( d \) is generated using the uniform distribution in \( \ell_\infty \) unit ball.
- **Distribution \( D_2 \):** The direction \( d \) is generated using the uniform distribution on \( \ell_1 \) unit sphere.
- **Distribution \( D_3 \):** The direction \( d \) is generated using specially designed distribution on \( \ell_1 \) unit sphere as follows: (i) The coordinate \( d_1 \) is taken uniformly on \([-A_k, A_k]\), \( d_k \) is taken uniformly from \([-A_k, A_k], \ k = 2, \ldots, n-1 \) and the last \( d_n \) takes \( A_n \) with random sign; (ii) coordinates of \( d \) are permuted randomly. The fact that in the random direction \( d \) only few coordinates \( d_i \) are significantly different from 0 can considerably speed up CGVNS when the problem dimension \( n \) is large.
- **Distribution \( D_4 \):** First the direction \( d \) is generated using uniform distribution in \( \ell_\infty \) unit ball. Then, with the current point \( x \) and box-constraint vectors \( a \) and \( b \), coordinates \( d_i \) are scaled as follows: (i) if \( d_i > 0 \) the scaling factor is \((b_i - x_i)/r_{k_{\text{max}}} \) (ii) if \( d_i < 0 \) the scaling factor is \((x_i - a_i)/r_{k_{\text{max}}} \). Thus, the point is always

```cpp
/* Initialization */
01 Select the set of neighborhood structures \( N_k, k = 1, \ldots, k_{\text{max}} \) and the array of random distributions types;
02 Choose an arbitrary initial point \( x \in S \)
03 Set \( x^* \leftarrow x, f^* \leftarrow f(x) \);

/* Main step */
04 repeat the following steps until the stopping condition is met
05 Set \( k \leftarrow 1 \);
06 repeat the following steps until \( k > k_{\text{max}} \)
07 for all distributions from the array do
08 Generate at random a point \( y \in N_k(x^*) \);
09 Apply some local search method from \( y \) to obtain a local minimum \( y' \);
10 if \( f(y') < f^* \) then
11 Set \( x^* \leftarrow y', f^* \leftarrow f(y') \) and goto line 05;
12 end
13 next
14 Set \( k \leftarrow k + 1 \);
15 end
16 end
17 Stop. Point \( x^* \) is an approximate solution of the problem.
```

Fig. 2. The pseudo-code of continuous general VNS.
within bounds, $\mathcal{N}_{k_{\text{max}}}(x)$ is equal to the whole box region $S$ and $\mathcal{N}_k(x)$ is the box region $S$ contracted towards the point $x$ with the contraction factor $r_k/r_{k_{\text{max}}}$.

In this paper we report numerical experiments with neighborhood structures defined by the pairs $(\ell_{\infty}, r_1), \ldots, (\ell_{\infty}, r_{k_{\text{max}}})$ and with the distribution array $(\mathcal{D}_3, \mathcal{D}_2, \mathcal{D}_1, \mathcal{D}_4)$. In the sequel we assume that neighborhoods are defined by (6). This choice is motivated by extensive computational experience. In that way we reduced the number of parameters to only three: $k_{\text{max}}$, $r_{\text{max}}$ and the choice of the local minimizer. Moreover, in order to get user-friendly heuristic, in our default version (CGVNS default) we fixed $k_{\text{max}}$ to 15.

4. Exterior point continuous general VNS for constrained optimization

Penalization methods are one of classical approaches to find local minima of the problem (1)–(4). These methods solve a constrained optimization problem by solving a sequence of unconstrained problems. Unconstrained problems involve an auxiliary function which incorporates the objective function together with penalty terms that measure violation of the constraints. Penalization methods include two major groups: exterior point penalty methods, which impose a penalty for violating a constraint, and interior point penalty methods, which impose a penalty for reaching the boundary of an inequality constraint.

Since the problem (1)–(4) has both inequality and equality constraints it is more natural to tackle it using exterior point methods. Here we use an exterior point penalty local minimization technique and combine it with VNS strategy in order to reach a near optimal solution of the problem (1)–(4). The corresponding unconstrained problem minimizes so called penalty function and has the form

$$
\min_{a \leq x \leq b} F_{\mu,q}(x) = f(x) + \frac{1}{\mu} P_q(x)
$$

with penalty

$$
P_q(x) = \sum_{i=1}^{m} \left( \max \{0, g_i(x)\} \right)^q + \sum_{i=1}^{r} |h_i(x)|^q,
$$

where $\mu$ is a positive penalty parameter and penalty exponent $q \geq 1$. If $q = 1$ it can be proved that the penalty function is exact, i.e. for some sufficiently small value of $\mu$ any (local) solution of problem (7) is a local minimum of (1)–(4). However, the exact penalty function is not differentiable at all points. For that reason the values $q > 1$, which guarantee differentiability, are also used. In this case the solutions of (7) under mild conditions converge to a local minimum of (1)–(4) when $\mu \to 0$ (see Avriel, 1976).

The main feature of exterior point CGVNS is to apply VNS methodology developed in Section 3 to problem (7) with appropriate variation of penalty parameter. Following CGVNS framework, the basic notions of our exterior point CGVNS algorithm for the problem (1)–(4) are specified as follows.

The initial solution $x$ in the initialization step is found as a feasible solution of the problem (7) for fixed penalty exponent $q$ and $\mu = \mu_0$, where $\mu_0$ is the given initial value of penalty parameter $\mu$. Point $y$ in the shaking step is randomly generated as a feasible solution of problem (7) for the current value of $\mu$ and then, in the local search step, a local search method is applied in order to find a local minimum $y'$ of this problem. If $y'$ is better than $x$ w.r.t. the function $F_{\mu,q}$, the algorithm moves to $y'$ and continues with the same $\mu$. Otherwise, the algorithm stays in $x$ and the value of $\mu$ is possibly decreased. Namely, if $\mu$ is greater than some given minimal value $\mu_{\text{min}}$, $\mu$ is updated multiplying it with a constant factor $\alpha$, $0 < \alpha < 1$, and the penalty function is appropriately changed. For each point $y'$ generated by the local search procedure, the corresponding penalty is calculated. If it is not greater than some very small feasible tolerance factor $\varepsilon$, the point is considered feasible for the problem (1)–(4) and the evidence of the best such point is kept.

The exterior point CGVNS can be summarized as in Fig. 3.

Additional parameters. As can be seen from the pseudo code above, the exterior point CGVNS uses additional parameters that are typical for the penalty function methods:

- $q$ – the penalty function exponent;
- $\alpha \in (0, 1)$ – the penalty decreasing rate;
- $\mu_{\text{min}}$ – the minimum penalty parameter.

The algorithm for the initialization step is specified as follows.
01 */ Initialization */
02 Select the set of neighborhood structures \( \mathcal{N}_k \), \( k = 1, \ldots, k_{\text{max}} \), the penalty function parameters \( (q, \mu_0, \mu_{\text{min}}) \) and the array of random distributions types;
03 Choose an arbitrary initial point \( x \) feasible for the box constrained problem (7);
04 Set \( x^* \leftarrow x \), \( f^* \leftarrow F_{\mu,q}(x) \), \( f_p \leftarrow \infty \); \( \mu \leftarrow \mu_0; \)
05 */ Main step */
06 repeat the following steps until the stopping condition is met
07 Set \( k \leftarrow 1; \)
08 repeat the following steps until \( k > k_{\text{max}} \)
09 for all distributions from the array do
10 Generate at random a point \( y \in \mathcal{N}_k(x^*) \)
11 Apply some local search method from \( y \) to obtain a local minimum \( y' \);
12 if \( F_{\mu,q}(y') \leq \epsilon \) and \( f(y') < f_p \) then set \( x^*_p \leftarrow y' \), \( f_p \leftarrow f(y') \);
13 else
14 Set \( \mu \leftarrow \max(\mu_{\text{min}}, \alpha \mu) \);
15 endif
16 next
17 Set \( k \leftarrow k + 1; \)
18 end
19 if \( f_p = \infty \) (i.e., \( x^*_p \) is not found) then
20 The feasible solution \( x^*_p \) is obtained by a local minimizer applied to \( F_{\mu,q}(x) \) with \( q = 1, \mu = \mu_{\text{min}} \), starting from \( x^* \);
21 endif
22 Stop. The point \( x^*_p \) is an approximate feasible solution of the problem.

Fig. 3. The exterior point CGVNS.

- \( \epsilon \) – small number, i.e., the tolerance for checking feasibility;
- \( \mu_0 \) and \( \mu_{\text{min}} \) – initial and minimal value of the penalty parameter, respectively.

In this paper, we report numerical experiments obtained with \( q = 2, \alpha = 0.9, \epsilon = 10^{-10}, \mu_0 = 1 \) and \( \mu_{\text{min}} = 10^{-8} \). Most of the suggested parameter values are obtained as a result of an extensive computational analysis. For example, the value of the penalty factor \( q = 2 \) has been suggested after running our method on a set of test instances with \( q = 1, 1.1, \ldots, 1.9, 2 \).

5. Computational results

Parameters. CGVNS currently implements neighborhood structures defined by metrics \( \ell_1 \) and \( \ell_{\infty} \) and distributions \( D_1 = D_4 \) defined in Section 3. The number \( k_{\text{max}} \) of different balls in one metric is an input parameter. Sizes of balls can be explicitly defined by the user or automatically generated by the software as follows: after finding the largest distance \( R \) from the current point \( x \) to the boundary of the box-region, radii are calculated as \( k_{\text{max}} \cdot R \). The random point \( y \) in the shaking step can optionally be chosen from the disk, i.e. such that it satisfies the additional requirement \( y \in \mathcal{N}_k(x) \cup \mathcal{N}_{k-1}(x) \).

Initial solution is an arbitrary point satisfying box constraints (4) provided by the user. Stopping criteria are the maximal allowed CPU time \( t_{\text{max}} \) or the maximal number of VNS iterations. Experiments were performed on Intel Pentium III processor, 930 MHz, 1 GB of RAM. In all tests the sizes of \( \mathcal{N}_k(x) \), \( k = 1, \ldots, k_{\text{max}} \) were generated automatically.

CGVNS default. In order to reduce the number of parameters, if not otherwise stated, CGVNS uses default set of parameters which gives good average performance in most experiments: \( k_{\text{max}} = 15 \), sequence of
neighborhood structures defined by pairs \((\ell_\infty, r_1), \ldots, (\ell_\infty, r_{15})\) and distribution type array \((D_3, D_2, D_1, D_4)\). The choice of local minimizer is specific for the problem. Thus, the minimizer that we actually use for each test problem will always be indicated.

**Standard test instances.** The power of CGVNS is first tested on a set of standard benchmark test functions from the literature (Battiti and Tecchiolli, 1996; Chelouah et al., 1999; Chelouah and Siarry, 2000; Cvijović and Klinowski, 1995; Dreo and Siarry, 2007; Siarry et al., 1997) for box constrained problems. In the following table, we give the list of problems with their dimensions and minimal objective function values \(f_{\min}\).

<table>
<thead>
<tr>
<th>Test function</th>
<th>Name</th>
<th>(n)</th>
<th>(f_{\min})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branin</td>
<td>BR</td>
<td>2</td>
<td>0.3979</td>
</tr>
<tr>
<td>Goldstein–Price</td>
<td>GP</td>
<td>2</td>
<td>3.0000</td>
</tr>
<tr>
<td>Hartman</td>
<td>HT3</td>
<td>3</td>
<td>-3.8628</td>
</tr>
<tr>
<td></td>
<td>HT6</td>
<td>6</td>
<td>-3.3224</td>
</tr>
<tr>
<td>Shubert</td>
<td>SB</td>
<td>2</td>
<td>-186.7309</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>RO2</td>
<td>2</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>RO10</td>
<td>10</td>
<td>0.0000</td>
</tr>
<tr>
<td>Griewank</td>
<td>GR</td>
<td>10</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shekel</td>
<td>SH5</td>
<td>4 ((m=5))</td>
<td>-10.1532</td>
</tr>
<tr>
<td></td>
<td>SH10</td>
<td>4 ((m=10))</td>
<td>-10.5364</td>
</tr>
<tr>
<td>Martin &amp; Gaddy</td>
<td>MG</td>
<td>2</td>
<td>0.0000</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>RA(n)</td>
<td>[2–500]</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

**G and GGP test instances.** The first set of constrained instances contains 10 test problems from Michalewicz (1994). The second set of test instances contains 10 general geometric programming (GGP) test problems from Floudas et al. (1999). In the next two tables for each problem we specify its dimension \(n\), the number of inequality constraints \(m\), the number of equality constraints \(r\), as well as the minimal objective function value \(f_{\min}\). Both sets of test instances are presented in the Appendix.

<table>
<thead>
<tr>
<th>Name</th>
<th>(n)</th>
<th>(m)</th>
<th>(r)</th>
<th>(f_{\min})</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>-15.0000</td>
</tr>
<tr>
<td>G2</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>7049.3309</td>
</tr>
<tr>
<td>G3</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>680.6301</td>
</tr>
<tr>
<td>G4</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0.0539</td>
</tr>
<tr>
<td>G5</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>24.3062</td>
</tr>
<tr>
<td>G7</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>-213.0000</td>
</tr>
<tr>
<td>G9</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-2.4714</td>
</tr>
<tr>
<td>G10</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-4.5142</td>
</tr>
<tr>
<td>G11</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>-11.0000</td>
</tr>
<tr>
<td>G12</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>-1.0000</td>
</tr>
</tbody>
</table>

**Local minimizers** used for local search are well known methods: Nelder–Mead (NM), Hooke–Jeeves (HJ), Rosenbrock (RO), steepest descent (SD), Fletcher–Powell (FP) and Fletcher–Reeves (FR). The first three do not require gradients and can be used for non-smooth objective functions. The other three methods use information on the gradient, which either can be user supplied or approximately calculated by the finite difference method. Local minimizers SD, FP and FR determine step-sizes by the golden section (GS) or the quadratic approximation method (QA).

In order to show that the choice of local minimizer may influence the efficiency and effectiveness of the CGVNS, we tested the default version of CGVNS using all six local optimizers. Tests were performed on
one small (RO2) and on one large (RA50) instance. Average and best results from 100 runs are reported in Table 1. Columns “comp. effort” report the average number of function (plus \( n \) times gradient, if any) evaluations until the global minimum is reached. Those columns show in fact the efficiency of the methods.

It appears: (i) all local minimizers (within CGVNS) are successful in solving RO2; (ii) the gradient type local minimizers (i.e., SD, FP, FR) were effective, but not as efficient as direct search methods NM and RO; (iii) in solving RA50, both NM and RO fail to find optimal solutions.

5.1. Problems with box constraints

Table 2 shows the average performance of CGVNS with default parameters in 100 runs. The first and the second column contain the test function code and the local minimizer code, respectively. In all runs optimal solutions were reached. To provide a consistent comparison with the published results, the computational effort in Table 2 is measured by the average number of function evaluations during the search process used to obtain the first global minimum with the same precision as in Siarry et al. (1997), i.e. to satisfy the test 

\[ |f(x) - f_{\text{min}}| \leq 10^{-4} f_{\text{min}} + 10^{-6}. \]

Here the number of function evaluations is computed as the number of objective function evaluations plus \( n \) number of gradient evaluations (if any). If input parameters are varied, computational effort can be reduced. The best results obtained by CGVNS (with parameter values given in columns 4 and 5) are also presented in the table.

The performance of CGVNS is then compared with six global optimization methods listed below.

**Table 1**

<table>
<thead>
<tr>
<th>Local minimizer</th>
<th>Objective value RO2</th>
<th>Comp. effort</th>
<th>Objective value RA50</th>
<th>Comp. effort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Best</td>
<td>Average</td>
<td>Best</td>
</tr>
<tr>
<td>SD</td>
<td>( 2.60 \times 10^{-5} )</td>
<td>( 9.44 \times 10^{-10} )</td>
<td>25,211</td>
<td>( 2.15 \times 10^{-9} )</td>
</tr>
<tr>
<td>FP</td>
<td>( 1.82 \times 10^{-5} )</td>
<td>( 7.44 \times 10^{-10} )</td>
<td>26,270</td>
<td>( 1.76 \times 10^{-9} )</td>
</tr>
<tr>
<td>FR</td>
<td>( 1.82 \times 10^{-5} )</td>
<td>( 7.44 \times 10^{-10} )</td>
<td>22,966</td>
<td>( 2.75 \times 10^{-8} )</td>
</tr>
<tr>
<td>NM</td>
<td>( 6.89 \times 10^{-8} )</td>
<td>( 1.55 \times 10^{-8} )</td>
<td>331</td>
<td>( 1.05 \times 10^{2} )</td>
</tr>
<tr>
<td>HJ</td>
<td>( 3.83 \times 10^{-5} )</td>
<td>( 1.18 \times 10^{-9} )</td>
<td>291,968</td>
<td>( 8.96 \times 10^{-1} )</td>
</tr>
<tr>
<td>RO</td>
<td>( 1.23 \times 10^{-9} )</td>
<td>( 5.57 \times 10^{-13} )</td>
<td>997</td>
<td>( 1.91 \times 10^{2} )</td>
</tr>
</tbody>
</table>

The figures in Table 3 show that CGVNS gives satisfactory results for all of the functions. It always detects the global minimum. In terms of computational effort it gives the best results for the functions GP, HT3, HT6,
RO2, GR, SH5, SH10 and MG. It is outperformed by CRTS for the function BR, by ECTS for SB and by ESA for RO10.

Rastrigin function with dimension $n = 50, 100, 150, 200$ (see Appendix) was another challenging test problem for CGVNS. Table 4 summarizes the average performance of the default CGVNS in 10 runs, and compares it with average behavior of multistart local search (MLS) and our previous heuristic based on VNS (GLOB). For a local minimizer we used the steepest descent method. In all cases CGVNS reached the optimal objective function value 0, MLS always failed, while GLOB was successful only for $n = 50$ and 100. CGVNS was also much more efficient than GLOB in cases where both methods found optimal solutions.

5.2. Problems with general constraints

The power of CGVNS was tested on two sets of constrained test problems and compared with the results of GENOCOP III, a well known public domain global optimization software based on genetic algorithm. As sta-
ted before, the default version uses the following values for the additional parameters: \( q = 2, \ \alpha = 0.9, \ \varepsilon = 10^{-10}, \ \mu_0 = 1 \) and \( \mu_{\text{min}} = 10^{-8} \).

The average results of experiments with the default versions of both CGVNS and GENOCOP III in 10 runs are summarized in Tables 5 and 6. The first and the second columns contain the test function name and the local minimizer code, respectively. For each problem the % deviation of the average and the best objective function values from \( f_{\text{min}} \left( \frac{f}{f_{\text{min}}} \right) \times 100 \), and the average computational effort needed to obtain the first global minimum with the precision \( 10^{-4} \) are calculated, and compared to the results obtained by GENOCOP III. For both methods the same stopping condition is used, i.e., the maximal allowed number of function evaluations.

According to Table 5, it can be seen that for G1, G3, G4, G7, G9, G11 and G12 CGVNS finds optimal solutions in all experiments, while GENOCOP III succeeds only for G1 and G12. Table 5 also shows that CGVNS outperforms GENOCOP III in both quality of function value and computational effort in all cases. This is probably due to the fact that GENOCOP III does not use a local minimizer within the search. In the case of G4 GENOCOP III failed to reach any feasible solution.

For the GGP test instances the average results for 10 runs are reported in Table 6. It can be seen that for GGP2 and GGP10 GENOCOP III failed to produce any feasible solution, and that it is outperformed by CGVNS in all cases.

6. Conclusions

In this paper we suggest a new heuristic for solving continuous (unconstrained and constrained) optimization problems. It is based on variable neighborhood search (VNS) metaheuristic and we call it continuous

<table>
<thead>
<tr>
<th>Function</th>
<th>Local minimizer</th>
<th>GGP test instances – general constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CGVNS default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aver. % error</td>
</tr>
<tr>
<td>GGP1</td>
<td>RO</td>
<td>0.52</td>
</tr>
<tr>
<td>GGP2</td>
<td>NM</td>
<td>0.00</td>
</tr>
<tr>
<td>GGP3</td>
<td>RO</td>
<td>0.16</td>
</tr>
<tr>
<td>GGP4</td>
<td>FP + QA</td>
<td>0.11</td>
</tr>
<tr>
<td>GGP5</td>
<td>NM</td>
<td>0.02</td>
</tr>
<tr>
<td>GGP6</td>
<td>FR + QA</td>
<td>0.00</td>
</tr>
<tr>
<td>GGP7</td>
<td>NM</td>
<td>0.00</td>
</tr>
<tr>
<td>GGP8</td>
<td>NM</td>
<td>0.02</td>
</tr>
<tr>
<td>GGP9</td>
<td>NM</td>
<td>0.92</td>
</tr>
<tr>
<td>GGP10</td>
<td>NM</td>
<td>3.83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Local minimizer</th>
<th>CG test instances – general constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CGVNS default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aver. % error</td>
</tr>
<tr>
<td>G1</td>
<td>HJ</td>
<td>0.00</td>
</tr>
<tr>
<td>G2</td>
<td>RO</td>
<td>0.26</td>
</tr>
<tr>
<td>G3</td>
<td>NM</td>
<td>0.00</td>
</tr>
<tr>
<td>G4</td>
<td>RO</td>
<td>0.00</td>
</tr>
<tr>
<td>G5</td>
<td>NM</td>
<td>0.72</td>
</tr>
<tr>
<td>G7</td>
<td>FP + QA</td>
<td>0.00</td>
</tr>
<tr>
<td>G9</td>
<td>RO</td>
<td>0.00</td>
</tr>
<tr>
<td>G10</td>
<td>NM</td>
<td>0.01</td>
</tr>
<tr>
<td>G11</td>
<td>SD + QA</td>
<td>0.00</td>
</tr>
<tr>
<td>G12</td>
<td>RO</td>
<td>0.00</td>
</tr>
</tbody>
</table>
general VNS (CGVNS for short). The idea of changing neighborhoods within the search has been fully explored: neighborhood structures in $\mathbb{R}^n$ are induced from the different metric functions; several points from the same neighborhood are randomly chosen according to different distributions; several local minimizers are used, etc.

Computational experience shows that our new VNS based method compares favorably to publicly available heuristic GENOCOP III based on genetic search. The comparison with other approaches from the literature on the same standard test instances shows that the default version of our method gives results of the similar quality, in terms of computational efforts (measured by the number of objective function calculations). However, fine tuning of CGVNS parameters for each instance (i.e., the choice of distributions, their order and the choice of maximal number of neighborhoods) allows us to obtain, in most cases, optimal solutions more efficiently compared to the published results.

The future work will be focused on two directions: (i) the application of CGVNS to real global optimization problems from industry; (ii) the extension of the method to allow the change of the local minimizer during the search process.

Acknowledgments

The authors are thankful to referees for several helpful suggestions for improving the presentation of the paper. This work is partly supported by Serbian Ministry of Sciences.

Appendix. Test functions

**Rastrigin** ($n \geq 1$, $f_{\text{min}} = 0$):

$$f(x) = 10n - \sum_{i=1}^{n}(x_i^2 - 10 \cos(2\pi x_i)), \quad -5.12 \leq x_i \leq 5.12, \ i = 1, \ldots, n.$$ 

**G1** ($n = 13$; $f_{\text{min}} = -15$):

$$f(x, y) = 5 \sum_{i=1}^{4}(x_i - x_i^2) - \sum_{i=1}^{9}y_i,$$

- $2x_1 + 2x_2 + y_6 + y_7 \leq 10,$
- $2x_1 + 2x_3 + y_6 + y_8 \leq 10,$
- $2x_2 + 2x_3 + y_7 + y_8 \leq 10,$
- $-8x_1 + y_6 \leq 0,$
- $-8x_2 + y_7 \leq 0,$
- $-8x_3 + y_8 \leq 0,$
- $-2x_4 - y_1 + y_6 \leq 0,$
- $-2y_2 - y_3 + y_7 \leq 0,$
- $-2y_4 - y_5 + y_8 \leq 0,$
- $0 \leq x_i \leq 1, \ i = 1, 2, 3, 4,$
- $0 \leq y_i \leq 1, \ i = 1, \ldots, 5, 9,$
- $0 \leq y_i, \ i = 6, 7, 8.$

**G2** ($n = 8$; $f_{\text{min}} = 7049.330923$):

$$f(x) = x_1 + x_2 + x_3,$$
\[1 - 0.0025(x_4 + x_6) \geq 0,\]
\[1 - 0.0025(x_5 + x_7 - x_4) \geq 0,\]
\[1 - 0.0025(x_8 - x_5) \geq 0,\]
\[x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \geq 0,\]
\[x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geq 0,\]
\[x_3x_8 - 1250000 - x_3x_5 + 2500x_5 \geq 0,\]
\[100 \leq x_1 \leq 10000,\]
\[1000 \leq x_i \leq 10000, \quad i = 2, 3,\]
\[10 \leq x_i \leq 1000, \quad i = 4, \ldots, 8,\]

**G3 (n = 7; \( f_{\min} = 680.6300573 \)):**
\[f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_4^3 + 3(x_4 - 11)^2 + 10x_6^6 + 7x_6^7 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,\]
\[127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0,\]
\[282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0,\]
\[196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0,\]
\[-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0,\]
\[-10 \leq x_i \leq 10, \quad i = 1, \ldots, 7.\]

**G4 (n = 5; \( f_{\min} = 0.0539498478 \)):**
\[f(x) = e^{x_1^2 + x_2^3 + x_3^4},\]
\[x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10,\]
\[x_3x_5 - 5x_4x_5 = 0,\]
\[x_1^2 + x_2^2 = -1,\]
\[-2.3 \leq x_i \leq 2.3, \quad i = 1, 2,\]
\[-3.2 \leq x_i \leq 3.2, \quad i = 3, 4, 5.\]

**G5 (n = 10; \( f_{\min} = 24.3062091 \)):**
\[(x) = x_1^2 + x_2^2 + x_3x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45,\]
\[105 - 4x_1 - 5x_2 + 3x_3 - 9x_8 \geq 0,\]
\[-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0,\]
\[-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0,\]
\[-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0,\]
\[8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0,\]
\[-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0,\]
\[3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0,\]
\[-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_2^2 + x_6 + 30 \geq 0,\]
\[-10 \leq x_i \leq 10, \quad i = 1, \ldots, 10,\]
G7 \((n = 6; f_{\min} = -213)\):

\[
\begin{align*}
f(x, y) &= -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5 \sum_{i=1}^{5} x_i^2, \\
6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 &\leq 6.5, \\
10x_1 + 10x_3 + y &\leq 20, \\
0 &\leq x_i, \quad i = 1, \ldots, 5, \\
0 &\leq y.
\end{align*}
\]

G9 \((n = 3; f_{\min} = -2.471428)\):

\[
\begin{align*}
f(x) &= -\frac{3x_1 + x_2 - 2x_3 + 0.8}{2x_1 - x_2 + x_3} - \frac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3}, \\
x_1 + x_2 - x_3 &\leq 1, \\
-x_1 + x_2 - x_3 &\leq -1, \\
12x_1 + 5x_2 + 12x_3 &\leq 34.8, \\
12x_1 + 12x_2 + 7x_3 &\leq 29.1, \\
-6x_1 + x_2 + x_3 &\leq -4.1, \\
0 &\leq x_i, \quad i = 1, 2, 3.
\end{align*}
\]

G10 \((n = 4; f_{\min} = -4.5142)\):

\[
\begin{align*}
f(x) &= x_1^{0.6} + x_2^{0.6} - 6x_1 - 4x_3 + 3x_4, \\
-3x_1 + x_2 - 3x_3 &= 0, \\
x_1 + 2x_3 &\leq 4, \\
x_2 + 2x_4 &\leq 4, \\
x_1 &\leq 3, \\
x_4 &\leq 1, \\
0 &\leq x_i, \quad i = 1, 2, 3, 4.
\end{align*}
\]

G11 \((n = 6; f_{\min} = -11)\):

\[
\begin{align*}
f(x, y) &= 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5, \\
x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 &\leq 16, \\
-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 &\leq -1, \\
2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 &\leq 24, \\
0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 &\leq 12, \\
-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 &\leq 3, \\
y_3 &\leq 1, \quad y_4 &\leq 1, \quad y_5 &\leq 2, \\
x &\geq 0, \quad y_i &\geq 0, \quad i = 1, \ldots, 5.
\end{align*}
\]

G12 \((n = 2; f_{\min} = -1)\):

\[
f(x) = \begin{cases} 
  x_2 + 10^{-5}(x_2 - x_1)^2 - 1, & 0 \leq x_1 < 2, \\
  \frac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3, & 2 \leq x_1 < 4, \\
  \frac{1}{3}(x_1 - 2)^3 + x_2 - \frac{11}{3}, & 4 \leq x_1 \leq 6,
\end{cases}
\]
\[
\frac{x_1}{\sqrt{3}} - x_2 \geq 0, \\
-x_1 - \sqrt{3}x_2 + 6 \geq 0, \\
0 \leq x_1 \leq 6, \quad x_2 \geq 0.
\]

**GGP1 – alkylation process design** \((n = 7; f_{\text{min}} = 1227.23):\)

\[
f(x) = c_1x_1 + c_2x_1x_6 + c_3x_3 + c_4x_2 + c_5 - c_6x_3x_5,
\]

\[
c_7x_6^2 + c_8x_1^{-1}x_3 - c_9x_6 \leq 1,
\]

\[
c_{10}x_1x_3^{-1} + c_{11}x_1x_3^{-1}x_6 - c_{12}x_1x_3^{-1}x_6^2 \leq 1,
\]

\[
c_{13}x_6^2 + c_{14}x_5 - c_{15}x_4 - c_{16}x_6 \leq 1,
\]

\[
c_{17}x_5^{-1} + c_{18}x_3^{-1}x_6 + c_{19}x_4x_5^{-1} - c_{20}x_5^{-1}x_6^2 \leq 1,
\]

\[
c_{21}x_7 + c_{22}x_3x_3^{-1}x_4^{-1} - c_{23}x_2x_5^{-1} \leq 1,
\]

\[
c_{24}x_7^{-1} + c_{25}x_3x_3^{-1}x_7^{-1} - c_{26}x_2x_3^{-1}x_4^{-1}x_7^{-1} \leq 1,
\]

\[
c_{27}x_5^{-1} + c_{28}x_5^{-1}x_7 \leq 1,
\]

\[
c_{29}x_3 - c_{30}x_7 \leq 1,
\]

\[
c_{31}x_3 - c_{32}x_1 \leq 1,
\]

\[
c_{33}x_3x_3^{-1} + c_{34}x_3^{-1} \leq 1,
\]

\[
c_{35}x_2x_3^{-1}x_4^{-1} - c_{36}x_2x_3^{-1} \leq 1,
\]

\[
c_{37}x_4 + c_{38}x_3^{-1}x_3x_4 \leq 1,
\]

\[
c_{39}x_1x_6 + c_{40}x_1 - c_{41}x_3 \leq 1,
\]

\[
c_{42}x_1^{-1}x_3 + c_{43}x_1^{-1} - c_{44}x_6 \leq 1,
\]

\[
(1500, 1, 3000, 85, 90, 3, 145) \leq (x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leq (2000, 120, 3500, 93, 95, 12, 162).
\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>(c_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.715</td>
</tr>
<tr>
<td>2</td>
<td>0.035</td>
</tr>
<tr>
<td>3</td>
<td>4.0565</td>
</tr>
<tr>
<td>4</td>
<td>10.000</td>
</tr>
<tr>
<td>5</td>
<td>3000.0</td>
</tr>
<tr>
<td>6</td>
<td>0.063</td>
</tr>
<tr>
<td>7</td>
<td>0.59553571 E–1</td>
</tr>
<tr>
<td>8</td>
<td>0.88392857</td>
</tr>
<tr>
<td>9</td>
<td>0.11756250</td>
</tr>
<tr>
<td>10</td>
<td>1.10880000</td>
</tr>
<tr>
<td>11</td>
<td>0.13035330</td>
</tr>
<tr>
<td>12</td>
<td>0.00660330</td>
</tr>
<tr>
<td>13</td>
<td>0.66173269 E–3</td>
</tr>
<tr>
<td>14</td>
<td>0.17239878 E–1</td>
</tr>
<tr>
<td>15</td>
<td>0.56595559 E–2</td>
</tr>
</tbody>
</table>

\[
(1.9120592 E–1) \leq (c_i) \leq (0.00619800).
\]

\[
(0.24623121 E–4) \leq c_{12} \leq (0.10204082 E–1).
\]

\[
(0.25125634 E–2) \leq c_{13} \leq (0.12244898 E–4).
\]

\[
(0.16118996 E+3) \leq c_{14} \leq (0.00006250).
\]

\[
(5000.0) \leq c_{15} \leq (0.00007625).
\]

\[
(0.44333333 E+2) \leq c_{16} \leq (1.22).
\]

\[
(0.33000000) \leq c_{17} \leq (1.0).
\]

\[
(0.02255600) \leq c_{18} \leq (1.0).
\]

\[
(0.00759500) \leq c_{19} \leq (0.00759500).
\]

**GGP2 – CSTR sequence design** \((n = 6; f_{\text{min}} = -0.38881):\)

\[
f(x) = -x_4,
\]
$x_1 + k_1 x_1 x_3 = 1,$
$x_2 - x_1 + k_2 x_2 x_6 = 0,$
$x_3 + x_1 + k_3 x_3 x_5 = 1,$
$x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0,$
$x_5^0 + x_6^0 \leq 4,$
$k_1 = 0.09755988,$
$k_2 = 0.99k_1,$
$k_3 = 0.0391908,$
$k_4 = 0.9k_3,$
$(0, 0, 0, 0, 10^{-5}, 10^{-5}) \leq (x_1, x_2, x_3, x_4, x_5, x_6) \leq (1, 1, 1, 16, 16).$

**GGP3 – heat exchanger design** ($n = 8; f_{\text{min}} = 7049.25$):

$f(x) = x_1 + x_2 + x_3,$
$833.33252x_1^{-1} x_4 x_6^{-1} + 100.0 x_6^{-1} - 83333.33 x_4^{-1} x_6^{-1} \leq 1,$
$1250.0 x_2^{-1} x_5 x_7^{-1} + 1.0 x_6 x_7^{-1} - 1250.0 x_2^{-1} x_6 x_7^{-1} \leq 1,$
$1250000.0 x_3^{-1} x_8^{-1} + 1.0 x_3 x_8^{-1} - 2500.0 x_3^{-1} x_5 x_8^{-1} \leq 1,$
$0.0025 x_4 + 0.0025 x_6 \leq 1,$
$- 0.0025 x_4 + 0.0025 x_3 + 0.0025 x_7 \leq 1,$
$0.01 x_6 - 0.01 x_5 \leq 1,$
$(100, 1000, 1000, 10, 10, 10, 10, 10) \leq (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \leq (10000, 10000, 10000, 1000, 1000, 1000, 1000, 1000).$

**GGP4 – optimal reactor design** ($n = 8; f_{\text{min}} = 3.9511$):

$f(x) = 0.4 x_1^{0.67} x_7^{-0.67} + 0.4 x_2^{0.67} x_8^{-0.67} + 10.0 - x_1 - x_2,$
$0.0588 x_3 x_7 + 0.1 x_1 \leq 1,$
$0.0588 x_6 x_8 + 0.1 x_1 + 0.1 x_2 \leq 1,$
$4 x_3 x_5^{-1} + 2 x_3^{-0.71} x_5^{-1} + 0.0588 x_3^{-1.3} x_7 \leq 1,$
$4 x_4 x_6^{-1} + 2 x_4^{-0.71} x_6^{-1} + 0.0588 x_4^{-1.3} x_8 \leq 1,$
$0.1 \leq x_i \leq 10, \quad i = 1, \ldots, 8.$

**GGP5 – Colville’s test problem** ($n = 5; f_{\text{min}} = 10122.6964$):

$f(t) = 5.3578 t_2^5 + 0.8357 t_1 t_5 + 37.2392 t_1,$
$0.00002584 t_5 t_5 - 0.00006663 t_2 t_5 - 0.0000734 t_1 t_4 \leq 1,$
$0.00085300 t_7 t_5 - 0.00009395 t_1 t_4 - 0.00033085 t_3 t_5 \leq 1,$
$1330.3294 t_2^{-1} t_5^{-1} - 0.42 t_1 t_5^{-1} - 0.30586 t_2^{-1} t_5^{-1} \leq 1,$
$0.00024186 t_2 t_5 + 0.00010159 t_1 t_2 + 0.00007379 t_2 \leq 1,$
$2275.1327 t_1^{-1} t_5^{-1} - 0.2668 t_1 t_5^{-1} - 0.40584 t_1 t_5^{-1} \leq 1,$
$0.00029955 t_5 t_5 + 0.00007992 t_1 t_5 + 0.00012157 t_3 t_4 \leq 1,$
$(78, 33, 27, 27, 27) \leq (x_1, x_2, x_3, x_4, x_5) \leq (102, 45, 45, 45, 45).$

**GGP6** ($n = 3; f_{\text{min}} = -83.2535$):

$f(t) = 0.5 t_1 t_2^{-1} - t_1 - 5 t_2^{-1},$
\[0.01t_2t_3^{-1} + 0.01t_4 + 0.0005t_1t_3 \leq 1, \quad 1 \leq t_1, t_2, t_3 \leq 100.\]

**GGP7** \((n = 4; \quad f_{\min} = -5.7398):\)

\[
f(t) = -t_1 + 0.4t_1^{0.67}t_3^{-0.67},
\]

\[0.05882t_3t_4 + 0.1t_1 \leq 1,
\]

\[4t_3t_4^{-1} + 2t_3^{-0.71}t_4^{-1} + 0.05882t_2^{-1}t_3 \leq 1,
\]

\[0.1 \leq t_1, t_2, t_3, t_4 \leq 10.
\]

**GGP8** \((n = 8; \quad f_{\min} = -6.0482):\)

\[
f(t) = -t_1 - t_5 + 0.4t_1^{0.67}t_3^{-0.67} + 0.4t_5^{0.67}t_7^{-0.67},
\]

\[0.05882t_3t_4 + 0.1t_1 \leq 1,
\]

\[0.05882t_7t_8 + 0.1t_1 + 0.1t_5 \leq 1,
\]

\[4t_3t_4^{-1} + 2t_5^{-0.71}t_4^{-1} + 0.05882t_2^{-1}t_3 \leq 1,
\]

\[4t_6t_5^{-1} + 2t_6^{0.71}t_5^{-1} + 0.05882t_6^{-1}t_7 \leq 1,
\]

\[0.01 \leq t_i \leq 10, \quad i = 1, \ldots, 8.
\]

**GGP9** \((n = 10; \quad f_{\min} = 1.1437):\)

\[
f(t) = t_6 + 0.4t_4^{0.67} + 0.4t_9^{0.67},
\]

\[t_3^{-1}t_2^{-1.5}t_3t_4^{-1}t_5^{-1} + 5t_3^{-1}t_2^{-1}t_3t_5^{1.2} \leq 1,
\]

\[0.05t_3 + 0.05t_2 \leq 1,
\]

\[10t_3^{-1} - t_3t_5^{-1} \leq 1,
\]

\[t_6^{-1}t_7^{-1.5}t_5^{-1}t_4^{-1} + 5t_6^{-1}t_7^{-1}t_8t_10^{1.2} \leq 1,
\]

\[t_2^{-1}t_7 + t_2^{-1}t_6 \leq 1,
\]

\[t_2^{-1}t_8 - t_6t_5^{-1} \leq 1,
\]

\[t_10 \leq 0.1.
\]

\[0.01 \leq t_i \leq 15, \quad i = 1, \ldots, 10.
\]

**GGP10** \((n = 11; \quad f_{\min} = 0.1406):\)

\[
f(t) = t_5^{-1},
\]

\[0.1t_1 + t_7t_5 \leq 1,
\]

\[10t_1t_4 + 10t_1t_4^{1.7} \leq 1,
\]

\[t_5^{-1} - 100t_2t_1 \leq 1,
\]

\[t_10t_8^{-1} - 10t_8 \leq 1,
\]

\[t_1^{-1}t_2t_3 + t_1^{-1}t_2t_3t_2^{1.7} \leq 1,
\]

\[t_5^{-1} - 10t_1^{-1}t_8t_11 \leq 1,
\]

\[10t_11 + 10t_9 \leq 1,
\]

\[t_2^{-1}t_3t_6 + t_2^{-1}t_3t_6t_2 \leq 1,
\]

\[t_6^{-1} - t_2^{-1}t_9 \leq 1.
\]

\[0.01 \leq t_i \leq 10, \quad i = 1, \ldots, 11.
\]
References


