Finding optimal separating halfspaces by
Variable Neighbourhood Search *

FRANK PLASTRIA, STEVEN DE BRUYNE
MOSI, Vrije Universiteit Brussel, Belgium.
{Frank.Plastria,Steven.De.Bruyne}@vub.ac.be

EMILIO CARRIZOSA
Universidad de Sevilla, Spain.
ecarrizosa@us.es

31st March 2006

Abstract
We consider the linear classification method consisting of separating two sets of points in \(d\)-space by a hyperplane. We investigate the situation where the two sets are nonseparable, and we wish to find the hyperplane which minimises the sum of distances from all misclassified points to the hyperplane. To this end two local descent methods are developed, one grid-based and one optimisation-theory based, and are embedded in several ways into a VNS metaheuristic scheme. Computational results show these approaches to be complementary, leading to a single hybrid VNS strategy which combines both approaches to exploit the strong points of each. Extensive computational tests show that the resulting method performs well.

Keywords. Classification, Separating halfspace, Minsum, norm-distance to hyperplane, Variable Neighborhood Search.

1 Introduction
The goal of classification is to find a simple rule to classify objects into one of given classes. The simplest rules are linear rules, and in this paper we study such rules for the separation of two classes of numerical data \(A, B \subset \mathbb{R}^d\).

Ideally a classification rule should correctly classify all objects of \(A\) and \(B\). However, such a perfect linear classification rule exists if and only if \(A\) and \(B\) are linearly separable, i.e. when their convex hulls are disjoint. The theory of Support Vector Machines (SVM) aims at determining a perfect rule of that kind which is optimal in some way. See e.g. [4].

Here, however, we focus on the quite common case where \(A\) and \(B\) are not separable. In SVM methodology, one then proposes to embed the data into another higher dimensional space in such a way that the given sets become linearly separable, and to apply the standard methodology in this new artificial space. The use of kernels makes such an approach feasible, see [14]. This embedding may be felt as artificial and ad-hoc, and we therefore do not follow it. We prefer to use criteria which exploit exclusively the given data. This means that misclassified points will appear as soon as \(A\) and \(B\) are not separable.

Several criteria may be applied, e.g. the popular minimisation of the number of misclassifications. Here we explore the criterion proposed by Mangasarian [8]: minimise the (possibly weighted) sum of misclassification distances, i.e. the distance to the separating halfplane for each misclassified point.

*Presented at Mini EURO Conference on VNS, Puerto de la Cruz, November 2005
point of $A$ and $B$. This is a nonconvex criterion, which for separable $A$ and $B$ is evidently
optimized by any separating halfplane with objective value 0, but for non separable classes turns
out to admit very many local optima. The aim of this work is to develop an efficient algorithm to
solve the resulting global optimization problem, in particular one that allows to tackle the large
scale data sets nowadays common in data mining. The work on this problem seems to be very
restricted at this moment.

Mangasarian [8] has described an exact solution algorithm when distances are measured by
the $L_1$-norm, based on solving two LP-subproblems per dimension, but this approach cannot be
generalized to other norms, in particular to the Euclidean norm. For this latter case, and for the
$L_\infty$-norm, exact solution approaches of branch-and-cut type were developed by Audet et al. [1].
To the best of our knowledge the only further work on this problem is the heuristic approach of
Caporossi et al [2], who use a Variable Neighbourhood Search (VNS, see [9, 6]) to solve problems
with any $L_p$-norm.

In this paper we report on an independent study of several heuristic approaches of VNS-
type to solve the problem under any norm. First a relatively simple adaptive grid-based VNS
approach, we called Marble, turns out to be very quick, but not to behave too well in all cases.
Then we develop a more complex local optimisation method based on two theoretical necessary
optimality conditions, which yields local optima from any starting point, but, when built into a
VNS framework, is much more time-consuming with somewhat disappointing results. Finally it
is by combining these two search strategies into a single framework that we obtain a quite stable
method which yields high quality results in acceptable times.

2 Problem statement

Let there be given two finite datasets $A, B \subset \mathbb{R}^d$. Together these form the training set of
the rule, and their number will be denoted by $p \equiv |A \cup B|$.

Any pair $\sigma \equiv (u, \beta) \in \mathbb{R}^d \setminus \{0\} \times \mathbb{R}$ defines the halfspaces and hyperplanes

$$H^\#(\sigma) = H^\#(u, \beta) \equiv \{ x \in \mathbb{R}^d \mid \langle u, x \rangle \# b \}$$

where $\# \in \{\leq, <, =, >\}$. Note that these sets all remain the same when $\sigma$ is multiplied by any
strictly positive constant, but not when the sign is inversed. In fact we will use the halfspaces
$H^\geq(\sigma)$ and $H^<(\sigma)$ to discriminate elements from $A$, which should be Above the hyperplane
$H^\geq(\sigma)$, i.e. in $H^>(\sigma)$, as opposed to the elements of $B$, which should be Below, i.e. in $H^<(\sigma)$.
Therefore we prefer to say that we separate by a halfspace (or oriented hyperplane), and the
sign of $\sigma$ is thus part of the information. We will also speak of the ‘halfspace $\sigma$’ by an abuse of
terminology.

Similarly for any subset $C \subset \mathbb{R}^d$ we define following sets

$$C^\#(\sigma) = C \cap H^\#(\sigma)$$

and therefore the set $A^\geq(\sigma)$, resp. $B^\leq(\sigma)$, is the set of correctly classified points from $A$, resp. $B$,
the set $A^\leq(\sigma)$, resp. $B^\geq(\sigma)$, is the set of misclassified points from $A$, resp. $B$, and the set $A^>(\sigma)$, resp.
$B^<(\sigma)$, contains the non-classified points from $A$, resp. $B$.

Any halfspace $\sigma^*$ minimizing the sum of distances from all misclassified points $A^<(\sigma^*) \cup B^>(\sigma^*)$
to its boundary hyperplane $H^=(\sigma^*)$ is called an optimal separating halfspace, i.e.

$$\sigma^* \in \arg\min\{ f(\sigma) \mid \sigma \in \mathbb{R}^d \setminus \{0\} \times \mathbb{R} \}$$

where

$$f(\sigma) \equiv \sum_{a \in A^<(\sigma)} d(a, H^=(\sigma)) + \sum_{b \in B^>(\sigma)} d(b, H^=(\sigma)) \quad (1)$$
The set of all halfspaces in $\mathbb{R}^d$ is denoted by $\mathcal{H}$. As mentioned above, for any $\lambda > 0$ we have $H^\#(\lambda \sigma) = H^\#(\sigma)$, so the function

$$H^\leq : \mathbb{R}^d \setminus \{0\} \times \mathbb{R} \to \mathcal{H} : \sigma = (u, \beta) \mapsto H^\leq(\sigma)$$

is surjective, with inverse image of each halfspace a ray $\mathbb{R}_0^+(u, \beta)$ for some $u \neq 0$ in $\mathbb{R}^d$. In particular each such ray contains a single $(u, \beta)$ with $\|u\| = 1$.

We will assume throughout that distance is measured by a norm $\gamma$. It is known that the distance of a point $a \in \mathbb{R}^d$ to a hyperplane $H(u, \beta)$ ($u \neq 0$) is then calculated as (see e.g. [12])

$$d_\gamma(a, H(u, \beta)) = \frac{|\langle u ; a \rangle - \beta|}{\gamma^\circ(u)}$$

where $\gamma^\circ$ is the dual norm of $\gamma$.

This expression allows to rewrite the objective function at $\sigma = (u, \beta)$ as

$$f(u, \beta) = \frac{1}{\gamma^\circ(u)} \left[ \sum_{a \in A^{<}(\sigma)} (\beta - \langle u ; a \rangle) + \sum_{b \in B^{>}(\sigma)} (\langle u ; b \rangle - \beta) \right]$$

and it can be seen there will always be an optimal solution $(u, \beta)$ for which

$$f(u, \beta) = \sum_{a \in A^{<}(\sigma)} (\beta - \langle u ; a \rangle) + \sum_{b \in B^{>}(\sigma)} (\langle u ; b \rangle - \beta)$$

by choosing $\gamma^\circ(u) = 1$.

## 3 Local descent methods

In this section we describe three local descent approaches which attempt to improve upon some starting solution. All can be considered as a simple descent strategy through a finite space of candidate solutions, a subset of all possible solutions but of quite different type for each method. The first method may be considered as a quick but ‘blind’ grid method, while the others are more involved, exploiting structural properties an optimal solution is known to satisfy.

Observe that each objective value evaluation $f(u, \beta)$ by (3) involves checking the sign of $\langle u ; c \rangle - \beta$ for each datapoint $c \in A \cup B$. Therefore all of these values have to be calculated, regardless of whether they will be used or not. However, the values not used are exactly those that are involved in the evaluation of the complementary half-space $f(-u, -\beta)$. This means that the two complementary halfspaces may be evaluated together at virtually the same cost as a single evaluation. Therefore in all our codes both halfspaces are always evaluated together and compared, so as to choose the better one.

### 3.1 Marble-descent

We consider regular grids in $\mathbb{R}^d \setminus \{0\} \times \mathbb{R}$ defined by some ‘maximum coefficient’ parameter $m \in \mathbb{N}$. This search space $\mathcal{H}(m)$ consists of all $(u, \beta)$ with integer coefficients in the interval $[-m, m]$ (excluding those with $u = 0$). It should be noted, that $\mathcal{H}(m)$ does not necessarily contain an optimal separating halfspace. But thanks to the surjectivity of the map $H^\leq$, an arbitrarily good approximation may be found in $\mathcal{H}(m)$ for sufficiently large $m$. 

On such a space we search as follows:

**Marble($m, \Delta$)**

- Choose a random solution $\sigma = (u, \beta) \in \mathcal{H}(m)$
• Sequentially for each steplength $\delta$ from $\Delta$ and halving down to 1
  
  - Cyclically for every coefficient, until a full cycle without change occurs
    
    * try adding and subtracting $\delta$ from this coefficient
    
    * check when this solution is in $\mathcal{H}(m)$ and update $\sigma$ if it is better

A one-parameter Marble($m$) corresponds to Marble($m,m$).

### 3.2 Cell-descent

By extension of the results on median hyperplanes obtained in [12], it was shown in [13] that for any norm distance measure, there always exists an optimal halfspace determined by a hyperplane satisfying the following two properties:

1. it is blocked, i.e. passes through $d$ affinely independent points of the training set $A \cup B$.

2. it balances the misclassified datapoints, i.e. both for $A$ and $B$ the number of their misclassified points cannot exceed the number of non-wellclassified points of the other class

Since in $\mathbb{R}^d$ any $d$ affinely independent points determine a unique hyperplane which passes through them, which generates two halfspaces, the set of blocked halfspaces $\mathcal{H}_b$ is finite. According to the first property we may restrict search to $\mathcal{H}_b$ without loss of optimality.

The second property may be used for any fixed $u \neq 0$ to find the best solution of type $(u, \beta)$ by translation: choose $\beta$ among the set of values $\{ u \mid A \cup B \}$ defined as $\{ \langle u; c \rangle \mid c \in A \cup B \}$ in such a way that it balances the number of values of $\langle u; A \rangle$ lower than $\beta$ against the number of values of $\langle u; B \rangle$ higher than $\beta$. This can be easily done in $O(p \log p)$ by a single sweep after sorting $\langle u; A \cup B \rangle$. Another method of linear complexity is described in [3].

Such a translation, even when started from a blocked hyperplane, does not necessarily result in a hyperplane in $\mathcal{H}_b$. Therefore we also need a blocking step, which allows to construct a blocked hyperplane starting from any $\sigma \in \mathcal{H}$, preferably one that has not higher objective value than $\sigma$. This may be obtained by a cell-move as explained below.

For any $\sigma_0 \in \mathcal{H}$ we define the cell $C(\sigma_0) \subset \mathcal{H}$ as all halfspaces that classify points similarly to $\sigma$, or, more precisely, that do not classify correctly any points of $A \cup B$ which were also misclassified by $\sigma_0$, and do not misclassify any other points:

$$C(\sigma_0) \equiv \{ \sigma \in \mathcal{H} \mid A^<(\sigma) \subset A^<(\sigma_0), A^>(\sigma) \subset A^>(\sigma_0), B^>(\sigma) \subset B^>(\sigma_0), B^<(\sigma) \subset B^<(\sigma_0) \}$$

For a halfspace $(u, \beta) \in \mathcal{H}$ to belong to this cell is expressed by the following linear inequalities:

$$\langle u; a \rangle \leq \beta \quad \forall a \in A^<(\sigma_0)$$

$$\langle u; a \rangle \geq \beta \quad \forall a \not\in A^<(\sigma_0)$$

$$\langle u; b \rangle \geq \beta \quad \forall b \in B^>(\sigma_0)$$

$$\langle u; b \rangle \leq \beta \quad \forall b \not\in B^>(\sigma_0)$$

Note that $\sigma_0 \in C(\sigma_0)$.

By (3) the constraint

$$f(u, \beta) = f(\sigma_0)$$

is a linear equality constraint not satisfied by $(0,0)$, and each halfspace in $C(\sigma_0)$ has exactly one representative satisfying this constraint. Therefore this constraint may be added in the definition of $C(\sigma_0)$ without loss of generality, and it was proven in [13] that the polyhedral subset of $\mathbb{R}_d^* \times \mathbb{R}$ we then obtain is nonempty and bounded. We will also call it $C(\sigma_0)$.

By (4) and (3) we see that minimising $f$ on $C(\sigma_0)$ is equivalent to maximising $\gamma^o(u)$ on $C(\sigma_0)$, and, by convexity of $\gamma^o$ the optimum will be reached at some extreme point $\sigma^*$ of $C(\sigma_0)$, and such
a $\sigma^*$ is always a blocked solution (for details see [13]). Furthermore, since $\sigma_0 \in C(\sigma_0)$, we will have $f(\sigma^*) \leq f(\sigma_0)$, as sought.

However, finding a maximum of $\gamma^\circ(u)$ on $C(\sigma_0)$ is a hard global optimisation problem (see e.g. [7], chapter I.2), and therefore we propose to solve the following linear approximation instead. At $\sigma_0 = (u_0, \beta_0)$ let $p_0 \in \partial \gamma^\circ(u_0)$ be any subgradient of the dual norm $\gamma^\circ$ at $u_0$. Then $\langle p_0 ; u \rangle = (p_0 ; u - u_0) + \gamma^\circ(u_0) \leq \gamma^\circ(u)$ for all $u$, because $\gamma^\circ$ is a norm for which it is well-known that $\langle p_0 \rangle ; u \rangle = \gamma^\circ(u_0)$. Therefore, maximising the linear function $\langle p_0 ; u \rangle$ on $C(\sigma_0)$ will also yield an extreme point of $C(\sigma_0)$, i.e. a blocked halfspace, with objective value no higher than $f(\sigma_0)$.

In case finding a subgradient $p_0 \in \partial \gamma^\circ(u_0)$ is not easy, one may use the direction of increase $p_0 = u_0$ of $\gamma^\circ$ at $u_0$ instead. Note that for the euclidean norm this choice is a positive multiple of a subgradient, so will perform as expected. For general norms, however, this does not fully guarantee that the new extreme point that will be obtained by solving the LP does not deteriorate the objective as compared to $\sigma_0$.

We can now describe our cell-descent as follows:

### Cell-Descent

- Choose $d$ random points of $A \cup B$, and find the $\sigma = (u, \beta) \in H_b$ passing through these points.
- Repeat until no new solution is found
  - Construct by translation the optimal solution $\sigma_0 = (u_0, \beta_0)$ with fixed $u_0 = u$.
  - Choose $p_0 \in \partial \gamma^\circ(u_0)$ or, if not available, take $p_0 = u_0$
  - Construct a new solution $(u, \beta) \in H_b$ by solving the following LP:

$$\begin{align*}
\max \quad & \langle p_0 ; u \rangle \\
\langle u ; a \rangle & \leq \beta \quad \forall a \in A^< (\sigma_0) \\
\langle u ; a \rangle & \geq \beta \quad \forall a \notin A^< (\sigma_0) \\
\langle u ; b \rangle & \geq \beta \quad \forall b \in B^> (\sigma_0) \\
\langle u ; b \rangle & \leq \beta \quad \forall b \notin B^> (\sigma_0) \\
f(u, \beta) &= f(\sigma_0) \\
u \in \mathbb{R}^d, \quad \beta \in \mathbb{R}
\end{align*}$$

### 3.3 Translation-descent

One may also consider the following much simpler local search method, solely based on translation.

### Translation-Descent

- Choose a random solution $\sigma = (u, \beta) \in H(m)$
- Construct by translation the optimal solution $\sigma_0 = (u_0, \beta_0)$ with fixed $u_0 = u$.

This descent-search will be inoperative if started on a balanced solution, and always ends with a balanced solution. Therefore it is useless to try to repeat it.

Marble-descent includes several trials of changes of the $\beta$-coefficient in a solution, which may be seen as an approximate form of Translation-descent. Cell-descent fully includes a Translation descent step in each of its loops. One may conclude that Cell-descent will certainly be more powerful than Translation-descent, while Marble descent will probably be so.
The algorithm described by Caporossi et al. [2] is based on such a translation-descent. However, it does not use the balancing property to find the best $\beta$-value for fixed $u$, but rather uses an updating method of the objective while sweeping the sorted sets $\langle u ; A \rangle$ and $\langle u ; B \rangle$.

### 3.4 Comparison of descent-methods

Both search methods were first coded in Matlab 7. For solving the LP’s in Cell-descent the built-in function ‘linprog’ could not be used, because it sometimes gave unexpected errors, and systematically malfunctioned when over 500 constraints were present. Therefore our code called CPLEX 9 by way of the CPLEXINT library [5].

Tests were then performed on the six datasets derived from the UCI Machine Learning Repository [11] for which Audet et al [1] published exact optimal solutions, unfortunately with only 4 significant digits. The details how these data sets were produced are given in that reference. For a correct interpretation of the results it is useful to know that all data sets, including the artificially produced ones discussed later, are always linearly standardised to $[0,1]$. All our tests are based on the euclidean distance $L_2$.

100 runs were done on each of the six datasets. The average results of these 100 runs are listed in table 1. Here $d$ denotes the dimension of the data-space, $p$ gives the size of the dataset, Trans descent, Marble(1000) and Cell-descent shows the resulting objective value when using Translation descent, Marble-descent with $m = \Delta = 1000$ or Cell-descent respectively, and the final column gives the global optimum value as published in [1].

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>Trans descent</th>
<th>Marble(1000)</th>
<th>Cell descent</th>
<th>Global optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>15.207</td>
<td>2.176</td>
<td>3.249</td>
<td>2.067</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>35.51</td>
<td>12.65</td>
<td>28.11</td>
<td>12.24</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>4.289</td>
<td>1.768</td>
<td>1.699</td>
<td>1.207</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>4.5548</td>
<td>0.60043</td>
<td>0.20693</td>
<td>0.03114</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>25.687</td>
<td>2.7770</td>
<td>4.2598</td>
<td>0.8971</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>11.253</td>
<td>2.1717</td>
<td>1.6327</td>
<td>0.8711</td>
</tr>
</tbody>
</table>

As expected, one observes that Translation-descent always gives worse results than both other descent methods. Marble(1000) and Cell-descent work much better, and give comparable results. They even seem to be somewhat complementary. Both methods remain however far from obtaining consistently a good approximation to the global optimum. It is therefore clear that both descent mechanisms should be extended by a global search framework.

### 3.5 Comparison of random search methods

Table 2 shows the best results obtained within the same 100 runs. This may be interpreted as results of repeated (100 trials) local search methods based on each of the three descent methods.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>Trans search</th>
<th>Marble(1000) search</th>
<th>Cell search</th>
<th>Global optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>3.277</td>
<td>2.072</td>
<td>2.079</td>
<td>2.067</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>16.65</td>
<td>12.26</td>
<td>15.42</td>
<td>12.24</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>2.182</td>
<td>1.218</td>
<td>1.318</td>
<td>1.207</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>1.2604</td>
<td>0.11016</td>
<td>0.03275</td>
<td>0.03114</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>9.9316</td>
<td>1.0377</td>
<td>0.9685</td>
<td>0.8971</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>6.5964</td>
<td>1.2494</td>
<td>0.8788</td>
<td>0.8711</td>
</tr>
</tbody>
</table>
The Translation descent-based search remains quite poor. Marble-search and Cell-search now produce quite good results, but still not systematically. Observe, however, that in most cases one of the two search methods finds a solution which is quite close to optimal, but it is not always the same one. The computation times for translation search were in the two-hundredth to one-tenth of second range, for Marble search in the ten to two-hundred seconds range, and for Cell search in the two-tenth to three seconds range.

It must be observed that our first implementations of Cell-search using the Matlab function linprog took considerably more time even than Marble-search.

4 Variable Neighbourhood Searches

4.1 General framework

We propose to use the metaheuristic framework of Variable Neighbourhood Search (VNS) [9, 6], described in general as follows. However, in all of our computational testing we use the simpler Variable Neighbourhood Descent (VND), which is a single main loop of VNS (obtained by choosing as stopping condition simply ‘True’).

\begin{itemize}
  \item Initialization:
    \begin{itemize}
      \item select the set of \textit{neighbourhood structures} $N_k$ ($k = 1, \ldots, k_{max}$) that will be used in the search
      \item find an \textit{initial solution} $x$
      \item choose a stopping condition
    \end{itemize}
  \item Main loop: Repeat the following sequence until the \textit{stopping condition} is met
    \begin{itemize}
      \item Set $k$ to 1.
      \item Repeat the following steps until $k > k_{max}$
        \begin{itemize}
          \item \textbf{Shaking}: generate a point $x$ at random from the $k$th neighbourhood of $x$ ($x \in N_k(x)$);
          \item \textbf{Local search}: apply some \textit{local descent} method with $x$ as initial solution, ending at a local optimum $x$.
          \item \textbf{Move or not} if $x$ is better than the incumbent $x$,
              then move there (i.e. set $x$ to $x$), set $k$ to 1
              otherwise, set $k$ to $k + 1$
        \end{itemize}
    \end{itemize}
\end{itemize}

By filling in the yet undefined details indicated in italics, we may obtain several different heuristics.

4.2 MarbleVNS

A first method MarbleVNS($m$), operating in the searchspace $\mathcal{H}(m)$, uses following specifications:

\begin{itemize}
  \item \textbf{Initial solution} a random halfspace with integer coefficients in the range $[-m, m]$
  \item \textbf{Neighbourhoods} $N_k(\sigma)$ contains all halfspaces having $d + 1 - k$ coefficients in common with $\sigma$ and $k$ new ones. We also take $k_{max} = d + 1$
  \item \textbf{Local descent} Marble($m, \lceil \sqrt{m} \rceil$), and using the current solution as initial solution
\end{itemize}
The particular choice of $\Delta = \lceil \sqrt{m} \rceil$ was dictated by our concern to give a more local search character to the inner Marble runs. A first series of tests with this scheme did not give satisfactory results. Furthermore it was observed that by far most improved solutions were found for very low $k$-values.

We therefore implemented four variants of this search strategy:

**MarbleNVNS($m$)** or Narrow Marble VNS: this is the standard Marble VND as described above (thus with a single main loop).

**MarbleBVNS($m$)** or Broad Marble VNS: now in the innermost loop each value of $k$ is repeated several times, in such a way that the total number of coefficients modified while working within $N_k$ is a constant (set to $k_{\text{max}}$). This means that $k = 1$ is used $k_{\text{max}}$ times, $k = 2$ is used $k_{\text{max}}/2$ times, etc.

**2S-NVNS($m^2$)** or two-stage Marble NVNS: first MarbleNVNS($m$) is executed, and followed by a Marble($m^2,m^2$) run

**2S-BVNS($m^2$)** as the previous, but using the broad version MarbleBVNS($m$)

We did a comparative test of the following methods: MarbleNVNS(100), MarbleBVNS(100), 2S-NVNS(1000), 2S-BVNS(1000), in other words, the two-stage methods work first on a coarser grid and secondly on a finer grid than in their one-step variants. Table 3 shows the average results obtained on the same data sets as before during 5 independent runs of each method, and includes again for comparison the ‘exact’ global optimal values given by [1]. The best heuristically found average value is shown in boldface. The last column indicates the relative deviation of this best value with respect to the ‘true’ optimum.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>NVNS</th>
<th>BVNS</th>
<th>2S-NVNS</th>
<th>2S-BVNS</th>
<th>Global opt.</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>2.086</td>
<td>2.081</td>
<td>2.081</td>
<td>2.067</td>
<td></td>
<td>0.5%</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>12.41</td>
<td>12.45</td>
<td>12.39</td>
<td>12.24</td>
<td></td>
<td>0.5%</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>1.267</td>
<td>1.258</td>
<td>1.286</td>
<td>1.207</td>
<td></td>
<td>2.1%</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>0.2880</td>
<td>0.1931</td>
<td>0.1688</td>
<td>0.03114</td>
<td></td>
<td>340.0%</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>2.198</td>
<td>2.450</td>
<td>2.201</td>
<td>0.8971</td>
<td></td>
<td>142.4%</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>1.122</td>
<td>1.415</td>
<td>1.157</td>
<td>0.8711</td>
<td></td>
<td>20.9%</td>
</tr>
</tbody>
</table>

We can observe that the best results were systematically obtained with MarbleBVNS. The improvement of BVNS upon NVNS indicates that the more intense search within small neighbourhoods is effective, whereas the second stage using a final finer grid-descent seems almost useless.

Observe also that the quality of the solutions found on the first three data sets is relatively good, but quite bad for the three last data sets.

These conclusions drawn from the objective values reached should be somewhat revised in view of the computational times taken by the different methods, as shown in table 4.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>NVNS</th>
<th>BVNS</th>
<th>2S-NVNS</th>
<th>2S-BVNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>205</td>
<td>608</td>
<td>193</td>
<td>442</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>349</td>
<td>700</td>
<td>271</td>
<td>406</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>27</td>
<td>56</td>
<td>17</td>
<td>33</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>69</td>
<td>197</td>
<td>58</td>
<td>126</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>875</td>
<td>1537</td>
<td>394</td>
<td>983</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>175</td>
<td>864</td>
<td>125</td>
<td>388</td>
</tr>
</tbody>
</table>
4.3 PointVNS

The second method PointVNS operates in the searchspace $\mathcal{H}_d$. Recall that this means that every halfspace is determined by $d$ affinely independent datapoints, and the choice of an orientation.

**Initial solution** a random hyperplane going through $d$ affinely independent data points

**Neighbourhoods** $N_k(\sigma)$ contains all halfspaces determined by $d-k$ data points common with $\sigma$ and $k$ new ones. Evidently $k_{\text{max}} = d$.

**Local descent** Cell descent, with the current solution as initial solution

Note that choosing a new solution in $N_k(\sigma)$ is not so simple. We have encountered many difficulties due to degenerate situations. Indeed, simply replacing $k$ datapoints that determine $\sigma$ by $k$ other datapoints often yielded a (nearly) affinely dependent set of points, which either led to numerical difficulties or did not define a single hyperplane. Therefore we had to include affine dependency tests in the code which was quite detrimental to its computational efficiency.

We have implemented two main variants of this search strategy:

**PointNVNS** or narrow Point VNS: this is the standard VND as described above.

**PointBVNS($k$)** or broad Point VNS: now in the innermost loop each value of $k$ is repeated as long as the total number of datapoints modified while working within $N_k$ does not exceed $rd$.

Each method was coded in Matlab and run 5 times on each data set. Table 5 shows the results obtained when applying the first and three instances of the second variant (taking $k = 1, 3, 10$) on the same datasets as before, using the same presentation as in table 3.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>NVNS</th>
<th>BVNS(1)</th>
<th>BVNS(3)</th>
<th>BVNS(10)</th>
<th>Global opt.</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>2.279</td>
<td>2.131</td>
<td>2.101</td>
<td>2.096</td>
<td>2.067</td>
<td>1.4%</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>16.74</td>
<td>14.93</td>
<td>14.28</td>
<td>13.49</td>
<td>12.24</td>
<td>10.2%</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>1.455</td>
<td>1.321</td>
<td>1.225</td>
<td>1.233</td>
<td>1.207</td>
<td>1.5%</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>0.03262</td>
<td>0.03154</td>
<td>0.03147</td>
<td>0.03147</td>
<td>0.03114</td>
<td>1.1%</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>1.045</td>
<td>0.9956</td>
<td>0.9575</td>
<td>0.9194</td>
<td>0.8971</td>
<td>2.5%</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>0.9918</td>
<td>0.9085</td>
<td>0.8917</td>
<td>0.8854</td>
<td>0.8711</td>
<td>1.6%</td>
</tr>
</tbody>
</table>

Here we observe that the broad VNS methods are always better than the narrow version. Usually (but not systematically) the most intensive search strategy in each neighbourhood gives the best results, as was to be expected. Concerning the quality, one may see that it is usually quite good, with a notable exception on the second data set.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>NVNS</th>
<th>BVNS(1)</th>
<th>BVNS(3)</th>
<th>BVNS(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>86</td>
<td>301</td>
<td>675</td>
<td>3724</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>47</td>
<td>149</td>
<td>349</td>
<td>1097</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>3</td>
<td>8</td>
<td>14</td>
<td>104</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>17</td>
<td>55</td>
<td>204</td>
<td>486</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>108</td>
<td>147</td>
<td>936</td>
<td>2728</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>9</td>
<td>47</td>
<td>83</td>
<td>295</td>
</tr>
</tbody>
</table>

The corresponding average computational times are given in table 6. Computation times for Point-BVNS($k$) increase almost linearly with $k$. 
5 Hybrid method: Marble-Cell

As compared to the results obtained with MarbleBVNS, shown in table 3, the results found by all PointVNS, shown in table 5, are worse for the two first data sets, but very much better for the three last ones.

This suggests that the two approaches MarbleVNS and PointVNS are complementary: when one performs badly, the other one performs well and vice-versa. This prompted us to combine their features into a single hybrid method.

We propose to mix both methods in the following way. The less expensive Marble descent method is used as the main part of the local descent mechanism working on the quite broad space $\mathcal{H}(1024)$, reserving the much more computationally involved Cell descent method as a kind of final touch-up step, pressing the solution towards a local minimum in $\mathcal{H}_p$. This resulted in the following local search scheme

Marble-Cell Descent

**Initial solution** randomly chosen.

**Descent** Repeat until stable solution is obtained
- Rescale current solution to $\mathcal{H}(1024)$
- Apply Marble(1024) starting from it
- Apply Cell-descent starting with the solution arrived at by Marble

The rescaling is obtained by first multiplying the current solution $\sigma$ by a constant factor such that the largest coefficient equals 1024 in absolute value, and then rounding all coefficients to the nearest integer. This rounding usually produces a slight deterioration of the solution’s quality, but this may be seen as a feature which enables to move out of a local minimum with reduced basin.

Table 7 summarizes the average results obtained as before during 5 runs of this method. However, in order to try to avoid the high computation times observed before with Matlab implementations, the first Descent step was executed using $m = 32$.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>MC-Descent</th>
<th>Global opt.</th>
<th>descent-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>2.072</td>
<td>2.067</td>
<td>0.2%</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>12.25</td>
<td>12.24</td>
<td>0.1%</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>1.385</td>
<td>1.207</td>
<td>14.7%</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>0.08269</td>
<td>0.03114</td>
<td>165.5%</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>0.8975</td>
<td>0.8971</td>
<td>0.04%</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>0.8722</td>
<td>0.8711</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

One may observe that the hybrid descent gives already often better results than the much more sophisticated VNS schemes in section 4.

Even better results may be expected when building this descent method into a VND. We propose the Marble-Cell VNS scheme with following characteristics.

**Initial solution** Solution obtained by Marble-Cell Descent

**Neighbourhoods** $N_k(\sigma)$ contains all halfspaces having $d + 1 - k$ coefficients in common with $\sigma$ and $k$ new ones.
Local search Marble-Cell Descent, with the current solution as initial solution

Note that by defining our neighbourhoods by modification of coefficients of $\sigma$, which are directly rescaled to belong to $\mathcal{H}(1024)$, we avoid the difficulties mentioned before of having to care about affine independency.

As the coefficient $\beta$ is one of the coefficients that is possibly modified in Marble, one may consider that Marble already includes a kind of translation step. Secondly, when an LP is solved within a cell corresponding to a balanced solution, the resulting extreme solution is often also balanced, and it is therefore useless to apply a translation step. Therefore we totally drop translations from the Cell-descent steps.

Since our Matlab codes were quite slow in general, we decided to make a completely new implementation in C. We made use of the GCC compiler, used double precision arithmetic and calls to CPLEX 9 for LP solving. All results given in the next sections were obtained with this C-code, and run on an Intel Xeon 3.40GHz processor with 2GB RAM.

6 Computational results

In order to study in detail the behaviour of our final algorithm, we decided to do more extensive tests on all the data sets used also by Caporossi et al. [2], which consist of the six UCI data sets discussed before, and several large scale data sets artificially produced as mixtures of normal distributions.

6.1 UCI data sets

We ran Marble-Cell VNS 100 times on each of the UCI data sets.

The following table gives statistics concerning the frequency with which certain solution qualities were obtained. Each column shows the number of runs that obtained a solution of given relative quality as compared to $b$, the best solution found during these 100 runs. Probably this is the global optimum, now determined up to 10 digits.

<table>
<thead>
<tr>
<th>Data set</th>
<th>$d$</th>
<th>$p$</th>
<th>best ($b$)</th>
<th>$# = b$</th>
<th>$&lt; 1.0001b$</th>
<th>$&lt; 1.001b$</th>
<th>$&lt; 1.01b$</th>
<th>$&lt; 1.1b$</th>
<th>$&lt; 2b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>2.066966216</td>
<td>69</td>
<td>69</td>
<td>96</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>708</td>
<td>12.24317523</td>
<td>4</td>
<td>52</td>
<td>83</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>1.207301887</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>97</td>
<td>98</td>
<td>100</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>0.631141906</td>
<td>14</td>
<td>14</td>
<td>40</td>
<td>68</td>
<td>83</td>
<td>100</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>0.897142644</td>
<td>11</td>
<td>45</td>
<td>81</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>0.871132909</td>
<td>47</td>
<td>95</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

These results show that the performance is quite different on different data sets. It seems that, contrary to the impression obtained from table 7, the echocardiogram data are the easiest to solve: the global optimum is found in 86% of the runs. Marble-CellVNS performs quite well on most data-sets, except on Glass.

The next table gives a summary of the observed calculation times. These times are clearly much lower than the times needed for Matlab implementations of much simpler methods. We estimate that we obtained a reduction by a factor of over 100 by moving to C.
Table 9: Calc. Times in 100 runs of Marble-Cell VNS on UCI data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>d</th>
<th>p</th>
<th>Min Time</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>StDev Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>1.438</td>
<td>2.946</td>
<td>6.281</td>
<td>1.013</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>3.938</td>
<td>8.158</td>
<td>17.719</td>
<td>2.875</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>0.140</td>
<td>0.318</td>
<td>0.625</td>
<td>0.117</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>0.625</td>
<td>1.356</td>
<td>2.860</td>
<td>0.463</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>7.047</td>
<td>11.092</td>
<td>19.016</td>
<td>2.451</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>1.844</td>
<td>2.682</td>
<td>4.421</td>
<td>0.450</td>
</tr>
</tbody>
</table>

We also prepared some graphs to check time versus quality, as shown in figure 1. One easily recognises that local optima corresponding to different values are found, most of which several times. The echocardiogram data are an exception with one clear global optimum found many times, and a small number of other local optima, some of which quite bad, found only few times. However, times needed to find these local optima do not seem to depend strongly on the quality.

6.2 Artificial data sets

The artificial data-sets used by [1, 2] consist of two series of ten sizes with ten data-sets of each size, all obtained using a Musicant’s NDC generator [10], a Matlab program which locates randomly a given number of centers, assigns them to one of two classes by splitting the set with randomly generated hyperplane, and finally produces multivariate normally distributed points from these centers using a random covariance matrix. We made use of the same data-sets made available by these authors. The first series has always $p = 2000$ points, but dimensions $d = 4, \ldots, 13$. The second series has fixed dimension $d = 6$, but numbers of points $p = 2000, \ldots, 20000$, by steps of 2000.

For each of these artificial data-set we did a single run of Marble-Cell VNS, always using euclidean distances, and obtaining the results summarized in table 10. The first two columns indicate the dimension $d$ and size $p$, the next two columns indicate the average objective values obtained over the 10 data-sets by Marble-Cell VNS and those published in [1], the next three columns indicate likewise the average CPU-times used by Marble-Cell VNS, and those published in [1] and in [2].

This shows that Marble-Cell VNS obtains very close to optimal values consistently. Calculation times of Marble-Cell VNS remain very acceptable even for high dimensions or large datasets, contrary to the performance of the exact method of [1]. They are very comparable to those of [2], although have a higher rate of increase with dimension $d$ and/or number of data points $p$.

6.3 Cross-validation

We finally performed a series of 10 tenfold cross-validations on each of the UCI databases, always using euclidean distances. The results are summarized in table 11. In this table we have also attempted to include similar results obtained in [1, 2]. However these are not directly comparable for two reasons. First [1] did only a single tenfold cross-validation, and since we observed a high degree of variation between our different tenfold cross-validation runs, the results of which were quite close to optimal, their results may not be reliable. Secondly, if we compare our results with those approximately given (only graphically) in [2], we find an almost perfect match.
Figure 1: UCI-data: Calculation time versus quality
Table 10: Marble-Cell VNS on artificial data sets: average results over 10 sets

<table>
<thead>
<tr>
<th>d</th>
<th>p</th>
<th>MC Distance</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MC-VNS</td>
<td>AHKNP-exact</td>
</tr>
<tr>
<td>4</td>
<td>2000</td>
<td>3.1130</td>
<td>3.1096</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>3.4558</td>
<td>3.4577</td>
</tr>
<tr>
<td>6</td>
<td>2000</td>
<td>4.3368</td>
<td>4.3304</td>
</tr>
<tr>
<td>7</td>
<td>2000</td>
<td>5.0493</td>
<td>5.0387</td>
</tr>
<tr>
<td>8</td>
<td>2000</td>
<td>5.9768</td>
<td>5.9641</td>
</tr>
<tr>
<td>9</td>
<td>2000</td>
<td>6.3159</td>
<td>6.2936</td>
</tr>
<tr>
<td>10</td>
<td>2000</td>
<td>6.4969</td>
<td>6.4801</td>
</tr>
<tr>
<td>11</td>
<td>2000</td>
<td>11.2930</td>
<td>11.2771</td>
</tr>
<tr>
<td>12</td>
<td>2000</td>
<td>7.2683</td>
<td>7.2610</td>
</tr>
<tr>
<td>13</td>
<td>2000</td>
<td>9.5310</td>
<td>9.4937</td>
</tr>
</tbody>
</table>

| 6  | 2000| 3.9870      | 3.9782    | 11.0    | 36.0 | 14.6 |
| 6  | 4000| 7.6811      | 7.6462    | 30.8    | 212.6| 28.3 |
| 6  | 6000| 14.2541     | 14.2193   | 36.4    | 655.5| 34.2 |
| 6  | 8000| 15.9780     | 15.9486   | 48.7    | 959.7| 54.9 |
| 6  | 10000| 23.8181    | 23.7869   | 69.7    | 1433.3| 77.7 |
| 6  | 12000| 27.1247    | 27.0759   | 79.1    | 2593.5| 69.6 |
| 6  | 14000| 35.9110    | 35.8001   | 105.5   | 2858.2| 74.9 |
| 6  | 16000| 25.9811    | 25.7808   | 90.6    | 2146.0| 112.4|
| 6  | 18000| 36.5834    | 36.5189   | 156.6   | 5915.7| 115.5|
| 6  | 20000| 25.0170    | 24.9326   | 160.4   | 5777.2| 118.5|

Table 11: Average of 10 tenfold cross-validations of Marble-Cell VNS on UCI data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>d</th>
<th>p</th>
<th>MC-VNS</th>
<th>AHKNP-exact</th>
<th>CHK-VNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>9</td>
<td>683</td>
<td>95.24%</td>
<td>95.31%</td>
<td>95.26%</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
<td>74.40%</td>
<td>73.83%</td>
<td>74.05%</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>7</td>
<td>74</td>
<td>60.30%</td>
<td>56.75%</td>
<td>60.50%</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>214</td>
<td>91.74%</td>
<td>92.06%</td>
<td>? %</td>
</tr>
<tr>
<td>Housing</td>
<td>13</td>
<td>506</td>
<td>82.23%</td>
<td>81.77%</td>
<td>82.25 %</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>150</td>
<td>78.46%</td>
<td>76.67%</td>
<td>? %</td>
</tr>
</tbody>
</table>

References


