Alternating local search based VNS for linear classification

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

June 29, 2007

Abstract

We consider the linear classification method consisting of separating two sets of points in d-space by a hyperplane. We wish to determine 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. Data Mining, Classification, Linear Classification, Heuristic Minimisation, Norm-distance, Variable Neighbourhood Search, Variable Neighborhood Search, VNS, Local Search, Grid Search, Cell 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 $A, B \subset \mathbb{R}^d$.

Several criteria may be applied, e.g. the popular minimisation of the number of misclassifieds. Here we rather explore the criterion proposed by Mangasarian [9]: minimise the (possibly weighted) sum of misclassification distances, i.e. the distance to the separating hyperplane for each misclassified point of A and B. This is a nonconvex criterion, which for separable A and B is evidently optimized by any separating hyperplane 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, so in all what follows we assume implicitly that A and B cannot be linearly separated.

Mangasarian [9] 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, exact solution approaches of branch-and-cut type were developed by Audet et al. [1]; see also Karam [7] for the L_∞-norm. To the best of our knowledge the only further algorithmic work on this problem is the heuristic approach of Karam et al [8], who use a Variable Neighbourhood Search (VNS, see [10, 5]) 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 we discuss a relatively simple adaptive grid-based VNS approach, which turns out to be very quick, but does not 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 formulation

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

We work with halfspaces and hyperplanes in $\mathbb{R}^d$. These are defined by some pair $\sigma \stackrel{\text{def}}{=} (u, \beta) \in \mathbb{R}^d \setminus \{0\} \times \mathbb{R}$ by

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

where $\# \in \{\leq, <, =, \geq, >\}$, and $\langle\ u\ ;\ x\ \rangle$ denotes the scalar product. Note that these sets all remain the same when σ is multiplied by any strictly positive constant, but not when its sign is inversed.

Any $\sigma = (u, \beta)$ defines the following linear classification rule:

For a new object $x \in \mathbb{R}^d$:
If $\langle\ u\ ;\ x\ \rangle > b$ then we classify x in A,
If $\langle\ u\ ;\ x\ \rangle < b$ then we classify x in B,
If $\langle\ u\ ;\ x\ \rangle = b$ then we consider x as non classified.

This means in fact that we use the halfspaces $H^{>}(\sigma)$ and $H^{<}(\sigma)$ to discriminate elements from A, considered to have to be **A**bove the hyperplane $H^{=}(\sigma)$, i.e. in $H^{>}(\sigma)$, as opposed to the elements of B, that should be **B**elow, i.e. in $H^{<}(\sigma)$. Therefore we say that we separate by a halfspace (or *oriented* hyperplane), and the sign of σ is thus part of the information. We will also speak of the 'rule' or 'halfspace σ' by an abuse of terminology.

In order to derive a good rule σ we must be able to differentiate between well and wrongly classified points by σ. Therefore we also need to define similarly following sets for any subset $C \subset \mathbb{R}^d$

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

The set $A^{>}(\sigma)$ (resp. $B^{<}(\sigma)$) is thus the set of *correctly classified points* from A (resp. B), the set $A^{<}(\sigma)$ (resp. $B^{>}(\sigma)$) 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).

According to the criterion proposed in [9], we say that a halfspace σ^* is an *optimal separating halfspace*, if it minimizes the sum of distances from all misclassified points $A^{<}(\sigma^*) \cup B^{>}(\sigma^*)$ to the boundary hyperplane $H^{=}(\sigma^*)$. Thus, determining an optimal rule means solving the following optimization problem

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

where

$$f(\sigma) \stackrel{\text{def}}{=} \sum_{a \in A^{<}(\sigma)} d(a, H^{=}(\sigma)) + \sum_{b \in B^{>}(\sigma)} d(b, H^{=}(\sigma)) \tag{1}$$

We denote the set of all halfspaces in $\mathbb{R}^d$ by $\mathcal{H}$. As mentioned above, for any $\lambda > 0$ we have $H^{\#}(\lambda\sigma) = H^{\#}(\sigma)$, so the function

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

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,β) with $\|u\| = 1$.

We will assume throughout that distance is measured by a given norm γ. 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. [13])

$$d_\gamma(a, H(u,\beta)) = \frac{|\langle\ u\ ;\ a\ \rangle - \beta|}{\gamma^\circ(u)} \tag{2}$$

where γ° is the dual norm of γ.

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

$$f(u,\beta) = \frac{1}{\gamma^\circ(u)} \left[\sum_{a \in A^<(\sigma)} (\beta - \langle\ u\ ;\ a\ \rangle) + \sum_{b \in B^>(\sigma)} (\langle\ u\ ;\ b\ \rangle - \beta) \right] \tag{3}$$

and it can be seen there will always be an optimal solution (u,β) 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 three can be considered as simple descent strategies 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 seen 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 to be used in the evaluation of the complementary half-space $f(-u,-\beta)$. This means that each pair of complementary halfspaces may be evaluated together at virtually the same cost as a single evaluation of each of them. Therefore in all our codes both complementary halfspaces are always evaluated together and compared, so as to always choose the better one.

3.1 Grid-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,β) 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 $\mathbf{H}^<$, an arbitrarily good approximation may be found in $\mathcal{H}(m)$ by choosing a sufficiently large m.

On such a space we search as follows:

Grid(m,Δ)

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

---

A one-parameter Grid(m) corresponds to Grid(m,m).

3.2 Cell-descent

By extension of the results on median hyperplanes obtained in [13], it was shown in [14] 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 that 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, β) by translation: choose β among the set of values $\langle\ u\ ;\ A \cup B\ \rangle \stackrel{\text{def}}{=} \{\ \langle\ u\ ;\ c\ \rangle\ \mid\ c \in A \cup B\ \}$ in such a way that it balances the number of values of $\langle\ u\ ;\ A\ \rangle$ lower than β against the number of values of $\langle\ u\ ;\ B\ \rangle$ higher than β. 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 [2].

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 that allows to construct a blocked hyperplane starting from any $\sigma \in \mathcal{H}$, preferably one that does not deteriorate the objective value. This may be obtained by a cell-move as explained next.

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 σ, or, more precisely, that do not classify correctly any points of $A \cup B$ which were also misclassified by σ_0, and do not misclassify any other points:

$$C(\sigma_0) \stackrel{\text{def}}{=} \{\ \sigma \in \mathcal{H}\ \mid\ A^{<}(\sigma) \subset A^{\leq}(\sigma_0), A^{\geq}(\sigma) \subset A^{\geq}(\sigma_0), B^{>}(\sigma) \subset B^{\geq}(\sigma_0), B^{\leq}(\sigma) \subset B^{\leq}(\sigma_0)\ \}$$

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

$$\begin{aligned}
\langle\ u\ ;\ a\ \rangle &\leq \beta \quad \forall a \in A^{<}(\sigma_0) \\
\langle\ u\ ;\ a\ \rangle &\geq \beta \quad \forall a \notin A^{<}(\sigma_0) \\
\langle\ u\ ;\ b\ \rangle &\geq \beta \quad \forall b \in B^{>}(\sigma_0) \\
\langle\ u\ ;\ b\ \rangle &\leq \beta \quad \forall b \notin B^{>}(\sigma_0)
\end{aligned}$$

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

By (3) the constraint

$$f(u, \beta) = f(\sigma_0) \tag{4}$$

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 [14] that the polyhedral subset of $\mathbb{R}_0^d \times \mathbb{R}$ 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 $\gamma^\circ(u)$ on $C(\sigma_0)$, and, by convexity of γ° the optimum will be reached at some extreme point σ^* of $C(\sigma_0)$, and such a σ^* is always a blocked solution (for details see [14]). 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. [6], 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 γ° at u_0. Then $\langle\ p_0\ ;\ u\ \rangle = \langle\ p_0\ ;\ u - u_0\ \rangle + \gamma^\circ(u_0) \leq \gamma^\circ(u)$ for all u, because γ° is a norm for which it is well-known that $\langle\ p_0\ ;\ u_0\ \rangle = \gamma^\circ(u_0)$. 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 γ° 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 σ_0.

We can now describe our cell-descent as follows:

Cell-Descent

- Choose d random points of $A \cup B$, and find the $\sigma = (u, \beta) \in \mathcal{H}_b$ 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 $(u, \beta) \in \mathcal{H}_b$ by solving the following LP:

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

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 $\sigma = (u, \beta) \in \mathcal{H}(m)$
- 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.

Grid-descent includes several trials of changes of the β-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 Grid descent will probably be so.

The algorithm described by Karam et al [8] is based on such a translation-descent. However, it does not use the balancing property to find the best β-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

All three 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 [4].

Tests were performed on the six datasets derived from the UCI Machine Learning Repository [12] 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, Grid(1000) and Cell-descent shows the resulting objective value when using Translation descent, Grid-descent with $m = \Delta = 1000$ or Cell-descent respectively, and the final column gives the global optimum value as published in [1].

Table 1: Descent methods: average results over 100 trials (Matlab)

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

As expected, one observes that Translation-descent always gives worse results than both other descent methods. Grid(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 2: Random search methods (100 trials): results (Matlab)

Data set	d	p	Trans search	Grid(1000) search	Cell search	Global optimum
Cancer	9	683	3.277	2.072	2.079	2.067
Diabetes	8	768	16.65	12.26	15.42	12.24
Echocardiogram	7	74	2.182	1.218	1.318	1.207
Glass	9	214	1.2604	0.11016	0.03275	0.03114
Housing	13	506	9.9316	1.0377	0.9685	0.8971
Hepatitis	16	150	6.5964	1.2494	0.8788	0.8711

The Translation descent-based search remains quite poor. Grid-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 Grid 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 Grid-search.

4 Variable Neighbourhood Searches

4.1 General framework

We propose to use the metaheuristic framework of Variable Neighbourhood Search (VNS) [10, 5], described in general below. However, in all of our computational testing we use the simpler Variable Neighbourhood Descent (VND), which consists of a single main loop of VNS (obtained by choosing as stopping condition simply 'True').

VNS

Initialization:

- select the set of *neighbourhood structures* N_k ($k = 1, \ldots, k_{max}$) that will be used in the search
- find an *initial solution* x
- choose a stopping condition

Main loop: Repeat the following sequence until the *stopping condition* is met

- Set k to 1.
- Repeat the following steps until $k > k_{max}$

  Shaking: generate a point x at random from the kth neighbourhood of x ($x \in N_k(x)$);

  Local search: apply some *local descent* method with x as initial solution, ending at a local optimum x.

  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 $k + 1$

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

4.2 GridVNS

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

Initial solution a random halfspace with integer coefficients in the range $[-m, m]$

Neighbourhoods $N_k(\sigma)$ contains all halfspaces having $d + 1 - k$ coefficients in common with σ and k new ones. We also take $k_{max} = d + 1$

Local descent Grid(m,$\lceil \sqrt{m} \rceil$), and using the current solution as initial solution

The particular choice of $\Delta = \lceil \sqrt{m} \rceil$ was dictated by our concern to give a more local search character to the inner Grid 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:

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

GridBVNS(m) or *Broad* Grid 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_{max}). This means that $k = 1$ is used k_{max} times, $k = 2$ is used $k_{max}/2$ times, etc.

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

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

We did a comparative test of the following methods: GridNVNS(100), GridBVNS(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 3: GridVNS: average results over 5 runs (Matlab)

Data set	d	p	NVNS	BVNS	2S-NVNS	2S-BVNS	Global opt.	error
Cancer	9	683	2.086	**2.078**	2.081	2.081	2.067	0.5%
Diabetes	8	768	12.41	**12.30**	12.45	12.39	12.24	0.5%
Echocardiogram	7	74	1.267	**1.232**	1.258	1.286	1.207	2.1%
Glass	9	214	0.2880	**0.1370**	0.1931	0.1688	0.03114	340.0%
Housing	13	506	2.198	**2.175**	2.450	2.201	0.8971	142.4%
Hepatitis	16	150	1.122	**1.053**	1.415	1.157	0.8711	20.9%

We can observe that the best results were systematically obtained with GridBVNS. 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 4: GridVNS: average computation times (s) over 5 runs (Matlab)

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

4.3 PointVNS

The second method PointVNS operates in the searchspace of blocked halfspaces $\mathcal{H}_b$. 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 σ and k new ones. Evidently $k_{max} = d$.

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 σ 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(r) 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 5: PointVNS: average results over 5 runs (Matlab)

Data set	d	p	NVNS	BVNS(1)	BVNS(3)	BVNS(10)	Global opt.	error
Cancer	9	683	2.279	2.131	**2.096**	2.101	2.067	1.4%
Diabetes	8	768	16.74	14.93	14.28	**13.49**	12.24	10.2%
Echocardiogram	7	74	1.455	1.321	**1.225**	1.233	1.207	1.5%
Glass	9	214	0.03262	0.03191	0.03154	**0.03147**	0.03114	1.1%
Housing	13	506	1.045	0.9956	0.9575	**0.9194**	0.8971	2.5%
Hepatitis	16	150	0.9918	0.9085	0.8917	**0.8854**	0.8711	1.6%

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 6: PointVNS: average computation times (s) over 5 runs (Matlab)

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

The corresponding average computational times are given in table 6. Computation times for Point-BVNS(k) increase almost linearly with k.

5 Hybrid method: Grid-Cell

As compared to the results obtained with GridBVNS, 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 GridVNS 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 Grid descent method is used as the main part of the local descent mechanism working on the quite large search 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 $\mathcal{H}_b$. This resulted in the following local search scheme

---

Grid-Cell Descent

Initial solution randomly chosen.

Descent Repeat until a stable solution is obtained

- Rescale the current solution to $\mathcal{H}(1024)$
- Apply Grid(1024) starting from it
- Apply Cell-descent starting with the solution arrived at by Grid

---

The rescaling is obtained by first multiplying the current solution σ 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 7: Grid-Cell Descent: average results over 5 runs (Matlab)

Data set	d	p	GC-Descent	Global opt.	descent-error
Cancer	9	683	2.072	2.067	0.2%
Diabetes	8	768	12.25	12.24	0.1%
Echocardiogram	7	74	1.385	1.207	14.7%
Glass	9	214	0.08269	0.03114	165.5%
Housing	13	506	0.8975	0.8971	0.04%
Hepatitis	16	150	0.8722	0.8711	0.1%

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 **Grid-Cell VNS** scheme with following characteristics.

Initial solution Solution obtained by Grid-Cell Descent

Neighbourhoods $N_k(\sigma)$ contains all halfspaces having $d + 1 - k$ coefficients in common with σ and k new ones.

Local search Grid-Cell Descent, with the current solution as initial solution

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

As the coefficient β is one of the coefficients that is possibly modified in Grid, one may consider that Grid 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 Karam et al. [8], 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 Grid-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 8: Grid-Cell VNS on UCI data sets: 100 runs

Data set	d	p	best (b)	# =b	$< 1.0001b$	$< 1.001b$	$1.01b$	$< 1.1b$	$< 2b$
Cancer	9	683	2.066966216	69	69	96	100	100	100
Diabetes	8	768	12.24317323	4	52	83	100	100	100
Echocardiogram	7	74	1.207301887	88	88	88	97	98	100
Glass	9	214	0.031141906	14	14	40	68	83	100
Housing	13	506	0.897142644	11	45	81	100	100	100
Hepatitis	16	150	0.871132909	47	95	99	100	100	100

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 88% of the runs. Grid-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 Grid-Cell VNS on UCI data sets

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

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

Table 10: Grid-Cell VNS on artificial data sets: average results over 10 sets

d	p	MC Distance		Time (s)		
		GC-VNS	AHKNP-exact	GC-VNS	AHKNP-exact	CHK-VNS
4	2000	3.1130	3.1096	3.5	5.0	10.5
5	2000	3.4658	3.4577	5.9	11.4	11.1
6	2000	4.3368	4.3304	12.2	34.5	13.9
7	2000	5.0493	5.0387	14.6	118.3	21.8
8	2000	5.9768	5.9641	20.6	557.4	29.2
9	2000	6.3159	6.2936	28.0	1796.3	44.3
10	2000	6.4969	6.4801	40.6	3194.4	56.8
11	2000	11.2930	11.2771	66.6	38530.5	64.0
12	2000	7.2683	7.2610	74.3	NA	78.5
13	2000	9.5310	9.4937	124.9	NA	93.8
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	635.5	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	25.8911	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

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, 8] consist of two series of ten sizes with ten data-sets of each size, all obtained using Musicant's NDC generator [11], a Matlab program which locates randomly a given number of centers, assigns them to one of two classes by splitting the set using a randomly generated separating 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 Grid-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 Grid-Cell VNS and those published in [1], the next three columns indicate likewise the average CPU-times used by Grid-Cell VNS, and those published in [1] and in [8].

This shows that Grid-Cell VNS obtains very close to optimal values consistently. Calculation times of Grid-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 [8], 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

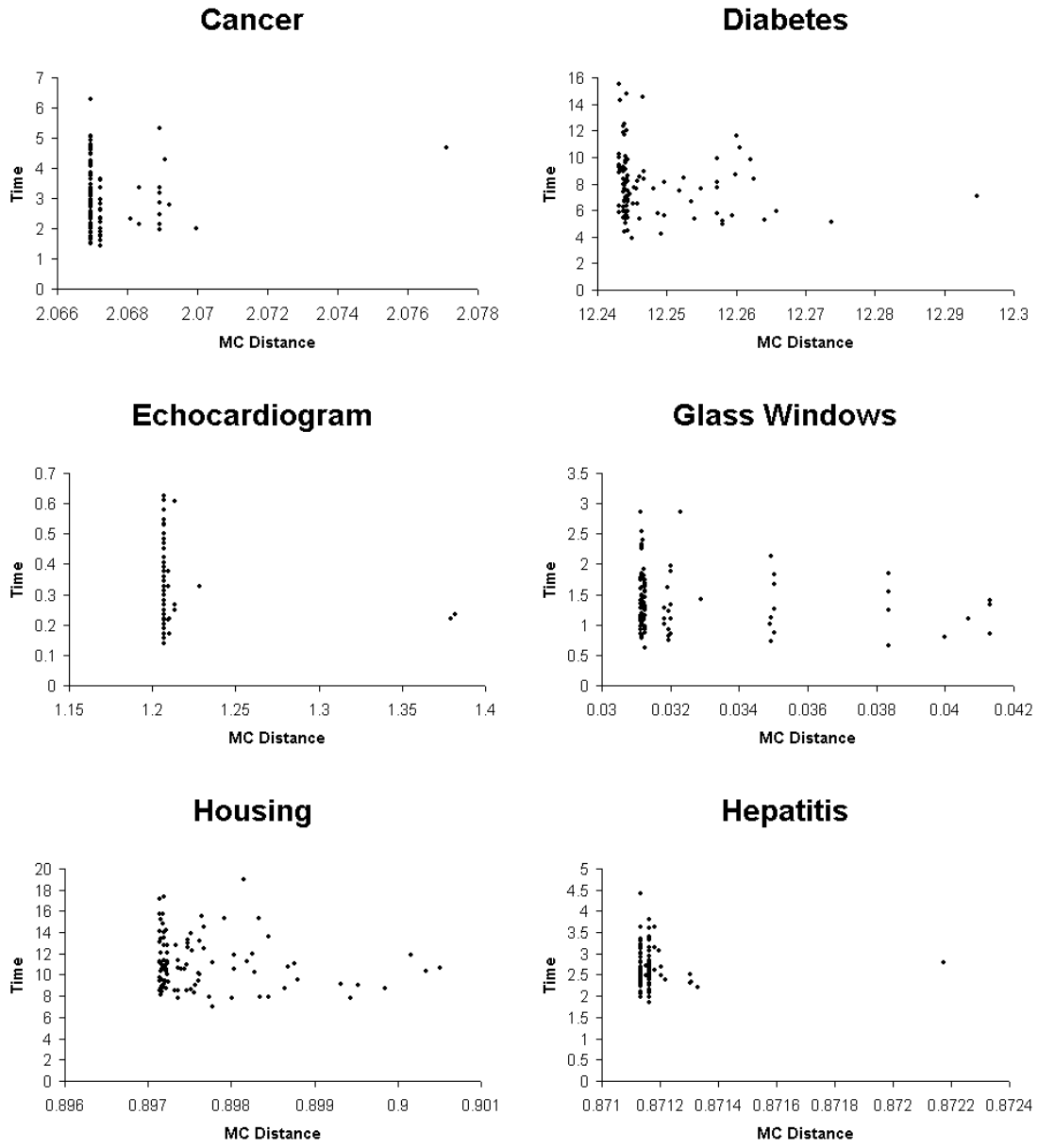

Figure 1: UCI-data: Calculation time versus quality

attempted to include similar results obtained in [7, 8]. However these are not directly comparable for two reasons. First [7, see fig2.2 p.34] 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 [8], we find an almost perfect match.

As a reference we also added the averages of 10 tenfold cross-validations of two other popular classification algorithms. The first one is the SMO algorithm from Weka[16], which we used to find SVM classifiers using linear kernels. The second one is the J48 algorithm from Weka[16], which we used to build C4.5 classification trees.

We do want to emphasise that it is not the goal of this paper to prove that the minimisation of the sum of misclassification distances is in any way superior to other criteria, but only that given this criterion it is possible to efficiently find a linear classifier that is not inferior to the guaranteed global optimum from a classification point of view. However, we would also like to mention that preliminary tests have indicated that by combining this criterion with a dimension reduction technique, it can be very competitive with the aforementioned industry standards. This approach will be reported elsewhere.

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

Data set	d	p	GC-VNS	AHKNP-exact	CHK-VNS	SMO	J48
Cancer	9	683	95.24%	95.31%	95.26%	97.00%	95.45%
Diabetes	8	768	74.40%	73.83%	74.05%	76.80%	74.49%
Echocardiogram	7	74	60.30%	56.75%	60.50%	70.95%	70.95%
Glass	9	214	91.74%	92.06%	NA	92.20%	93.27%
Housing	13	506	82.23%	81.77%	82.25%	86.42%	81.90%
Hepatitis	16	150	78.40%	76.67%	NA	84.60%	77.60%

References

[1] Audet C., Hansen P., Karam A., Ng C.D., Perron S. (2007) Exact L_2-norm plane separation. Technical report, Les Cahiers du GERAD, G-2007-09, Montreal, Canada.

[2] Carrizosa E. and Plastria F. (2007) Optimal expected distance separating halfspace. *Mathematics of Operations Research*, to appear, *Online version*: `http://www.optimization-online.org/DB_HTML/2004/10/971.html`

[3] Cristianini N. and Shawe-Taylor J. (2000) *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

[4] CPLEXINT - Matlab interface for the CPLEX solver, `http://control.ee.ethz.ch/~hybrid/cplexint.php`

[5] Hansen P. and Mladenović N. (2003) Variable neighborhood search. In Glover F. Kochenberger G. (Eds.) *Handbook of Metaheuristics*. Kluwer Academic Publisher. p. 145–184.

[6] Horst R. and Tuy H. (1990) *Global optimization: deterministic approaches.* Springer.

[7] Karam A., (2005) Essays on Linear Discrimination, PhD thesis at HEC Montréal, Option Méthodes Quantitatives de Gestion.

[8] Karam A., Caporossi G., Hansen P. (2007) Arbitrary-norm hyperplane separation by variable neighbourhood search. *IMA J Management Math 2007* 18:173–189.

[9] Mangasarian, O.L. (1999) Arbitrary-Norm Separating Plane. *Operations Research Letters* 24: 15–23.

[10] Mladenović N. and Hansen P. (1997) Variable neighborhood search. *Computers and Operations Research* 24: 1097–1100.

[11] Musicant D. (1998) NDC: Normally distributed clustered datasets, `http://www.mathcs.carleton.edu/faculty/dmusican/ndc/`

[12] Newman D.J., Hettich S., Blake C.L., Merz C.J. (1998). UCI Repository of machine learning databases `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.

[13] Plastria F. and Carrizosa E. (2001) Gauge-distances and median hyperplanes. *Journal of Optimization Theory and Applications*110: 173–182, .

[14] Plastria F. and Carrizosa E. (2002) Optimal distance separating halfspace. *Working Paper:* BEIF/124, Vrije Universiteit Brussel. `http://www.optimization-online.org/DB_HTML/2004/10/970.html`

[15] Shawe-Taylor J. and Cristianini N. (2004) *Kernel Methods for Pattern Analysis*. Cambridge University Press.

[16] Witten I.H. and Frank E. (2005) *Data Mining: Practical machine learning tools and techniques, 2nd Edition* Morgan Kaufmann, San Francisco, 2005.
Weka software: `http://www.cs.waikato.ac.nz/~ml/weka/index.html`