Combinatorial Optimization and High Dimensional Billiards

Pál Ruján[1,2]

[1] SER Systems AG, Knowledge Processing Team,
Sandweg 236, D-26135 Oldenburg
[2] Fachbereich Physik and I.C.B.M.
Carl von Ossietzky Universität
Postfach 2503, D-26111 Oldenburg
Germany

Abstract. Combinatorial optimization deals with algorithms for finding extrema of functions subject to a (possibly large) number of constraints. Bayesian inference requires in addition averages over such extrema. In this talk we show how simple dynamic systems like billiards can be used to find solutions for such problems. The topics covered are linear and quadratic programming, classification, and Bayesian mixture problems.

1 Introduction

Everyone who has or is studying physics remembers with somewhat mixed feelings the d'Alambert principle and the Lagrange equations of motion of the first kind. In one side, one could argue that it is exactly the cumbersome mathematics of taking into account boundary conditions which led to the development of Hamiltonian mechanics. On the other side, like it or not, most problems of practical interest consist of finding the extrema of simple functions (mostly linear or quadratic), subject to a large set of constraints, mostly linear (in)equalities.

In this talk I will consider three different problems and touch superficially a few others. These problems are: linear programming, the Bayes Perceptron, and the mixture inference problem. The presentation is mostly geometrical: once a problem and its ramifications are understood geometrically, it is easy to write it down in algebraic form, and ultimately, devise appropriate algorithms.

One of the simplest Hamiltonian systems is a free particle moving in a closed container of given shape. Assuming that the collisions between the particle and the walls are perfectly elastic, we are led to a *billiard*. The analysis of all possible phase space trajectories is exciting physics, the quantization of such systems even more so. This talk is, however, not about new results concerning dynamic systems and ergodic theory. Our goal is to use physical knowledge in order to devise working algorithms for solving effectively difficult problems.

Consider the convex polygon shown in Fig. 1. In general terms, I will deal here with convex polyhedra living in D-dimensions. Such polyhedra can be defined in different ways:

1. Give the set of its vertices (here A,B,C,D, and E)

$$\boldsymbol{x}_A,\ \boldsymbol{x}_B, \ldots, \boldsymbol{x}_E \in \mathcal{R}^D \tag{1}$$

2. Give a set of normal vectors pointing outside the polyhedron ($\boldsymbol{n_A}$, $\boldsymbol{n_B}$, etc.) as a set of inequalities:

$$\begin{aligned} \boldsymbol{n}_A\boldsymbol{x} &\geq b_A \\ \boldsymbol{n}_B\boldsymbol{x} &\geq b_B \\ \boldsymbol{n}_C\boldsymbol{x} &\geq b_C \\ \boldsymbol{n}_D\boldsymbol{x} &\geq b_D \\ \boldsymbol{n}_E\boldsymbol{x} &\geq b_E \end{aligned} \tag{2}$$

   $\boldsymbol{ab}$ denotes here the scalar product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.
3. Give a set of linear equations as

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{x} \geq 0 \tag{3}$$

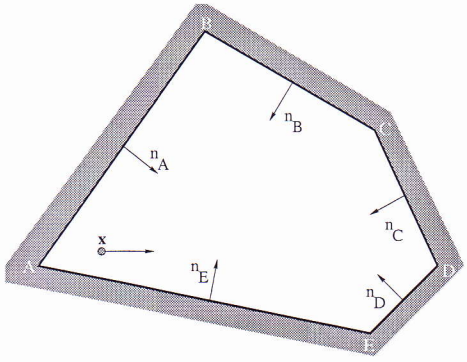

Fig. 1. A convex polygon

The form (2) is equivalent to form (3) - see below. In low dimensions, we can easily transform (1) into (2) and vice versa. Since the number of vertices, lines, facets, etc. grows exponentially with the dimensionality, this transformation becomes infeasible in high dimensions. Therefore, the difficulty of the problem we have to solve depends crucially on the available representation of the convex polyhedron.

Any point of a convex polyhedron can be expressed as a convex combination of its N vertices, $\{\boldsymbol{\xi}^{(i)}\}_1^N$:

$$\boldsymbol{x} = \sum_{i=1}^{N} a_i \boldsymbol{\xi}^{(i)} \tag{4}$$

where

$$\sum_{i=1}^{N} a_i = 1 \quad \text{and} \quad a_i \geq 0, \quad i = 1, \ldots, N \tag{5}$$

Note that a polyhedron might have many more vertices than the space dimensionality, hence the linear combination (4) is in general not unique. A vertex of the polyhedron i has by definition the form $a_i = 1$, $a_{j\neq i} = 0$, a segment-like facet is defined by $a_i + a_j = 1$, $a_{k\neq i,j} = 0$, etc.

As exposed here, it might be not clear why this type of problems are *combinatorial*. In reality, a typical problem looks more like Fig. 2, where we do not know a priori which of the constraints are stringent and which ones are shadowed by others.

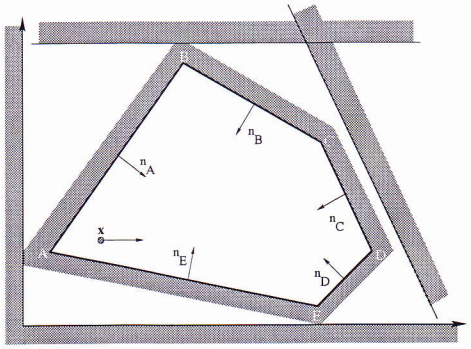

Fig. 2. A set of redundant linear inequalities

One aspect of the combinatorial problem is to determine the *stringent* inequalities, those which form in our case the faces of the internal polyhedron (polygon). Once the stringent inequalities and the corresponding polyhedron has been found, we must find at least one *feasible* solution, a vector $\boldsymbol{x}$ satisfying all constraints. The space of all such vectors is called the feasibility space in mathematical programming or *version* space in pattern recognition and Artificial Intelligence. Another aspect of the combinatorial problem is the shear number of constraints. Many difficult optimization problems, like for instance the traveling salesman problem, lead to an exponential number of constraints. An example will be shown below.

In the rest of this talk we will consider three different problems and will discuss different algorithms to solve them:

1. **The *linear* programming problem**
   Given a cost vector $\boldsymbol{c}$ with N components, find
   $$\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{6}$$
   satisfying the set of M inequalities (3) and $\boldsymbol{x} \geq 0$.
2. **The Perceptron Bayes-point problem:** find the theoretically best classifier for a linear classification machines, equivalent to:
   Given a set of inequalities (3), compute the center of mass of the polyhedron.
   As discussed later, the polyhedron is in this case either open (a polyhedral

cone) or defined on the intersection of a polyhedral cone with the unit hypersphere.
We will also discuss the main problems occurring when kernels are introduced and the space dimensionality becomes infinite.

3. **The Bayesian inference problem for mixture assignment problem**
Consider M substances defined through their (normalized) spectral decomposition. Maximally N spectral lines are considered. A large database of such substances is built, $M > N$. Next, the spectrum of an unknown substance is measured. We are asked to make a scientifically sound estimate of its origin in terms of the substances stored in the database.
Define a set of M vertices in N dimensions (the substances in the database) as $\{\boldsymbol{\xi}^{(\alpha)}\}_1^M$, an additional set of N equalities, and an point $\boldsymbol{x}_0$ (the unknown substance). Hence,

$$\boldsymbol{x}_0 = \sum_{\alpha=1}^{M} a_\alpha \boldsymbol{\xi}^{(\alpha)} \tag{7}$$

with $a_\alpha \geq 0$ and $\sum_{\alpha=1}^{M} a_\alpha = 1$. Estimate the marginal probabilities $p(a_\alpha)$, $\alpha = 1, \ldots, M$ over all allowed convex decompositions Eq. (7).

While the first problem ask for an extremal solution, like finding the ground system of system with many degrees of freedom in physics, the last two problems require finding optimal solutions based on posteriori probabilities. I call such problems *statistical* combinatorial optimization problems, since they require the computation of certain averages over the whole feasibility (version) space.

2 Billiards

A *billiard* is defined as a closed space region (compact set) $\mathcal{P} \in \mathbb{R}^N$ of the N dimensional vector space. The boundaries of a billiard are piecewise smooth functions. Within these boundaries a point mass (ball) moves freely, except for the elastic collisions with the enclosing walls. Hence, the absolute value of the momentum is preserved and the phase space $\mathcal{B} = \mathcal{P} \times \mathcal{S}^{N-1}$, is the direct product of $\mathcal{P}$ and $\mathcal{S}^{N-1}$, the surface of the N-dimensional unit velocity sphere. Such a simple Hamiltonian dynamics defines a flow and its Poincaré map an automorphism. The mathematicians have defined a finely tuned hierarchy of notions related to such dynamic systems. For instance, simple ergodicity as implied by the Birkhoff-Hincsin theorem means that the average of any integrable function defined on the phase space over a single but very long trajectory equals the spatial mean, except for a set of zero measure. Furthermore, integrable functions invariant under the dynamics must be constant. From a practical point of view this means that almost all very long trajectories will cover the phase space uniformly.

Properties like mixing (Kolmogorov-mixing) are stronger than ergodicity. They require the flow to mix uniformly different subsets of $\mathcal{B}$. In hyperbolic systems one can go even further and construct Markov partitions defined on symbolic dynamics and eventually prove related central limit theorems.

Not all convex billiards are ergodic. Notable exceptions are ellipsoidal billiards, which can be solved by a proper separation of variables [1]. Already Jacobi knew that a trajectory started close to and along the boundaries of an ellipse cannot reach a central region bounded by the so-called caustics. This simple fact is seemingly ignored by the many architects building round concert halls.

In addition to billiards which can be solved by a separation of variables, there are a few other exactly soluble polyhedral billiards [2]. Such solutions are intimately related to the reflection method - the billiard tiles perfectly the entire space. A notable example is the equilateral triangle billiard, first solved by Lamé in 1852. Other examples of such integrable billiards can be obtained by mapping exactly soluble one dimensional many particle system into a one-particle high-dimensional billiard [3].

Apart from these exceptions, small perturbations in the form of the billiard usually destroy integrability and lead to chaotic behavior. For example, the stadium billiard (two half-circles joined by two parallel lines) is ergodic in a strong sense, the so-called metric entropy is non-vanishing - for definitions and details see [4]. The dynamics induced by the billiard is hyperbolic if at any point in phase space there are both expanding (unstable) and shrinking (stable) manifolds. A famous example is Sinai's reformulation of the Lorentz-gas problem. Deep mathematical methods were needed to prove the Kolmogorov-mixing property and in constructing the Markov partitions for the symbolic dynamics of such systems [5].

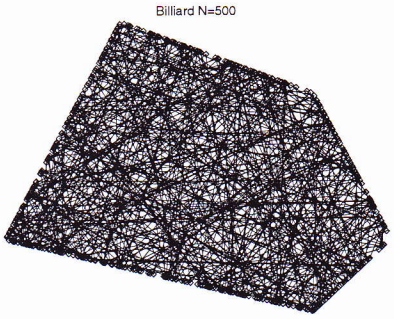

Fig. 3. A trajectory after 500 collisions

The question whether a particular billiard is ergodic or not can be decided in principle by solving the Schrödinger problem for a free particles trapped in the billiard-box. If the eigenfunctions corresponding to the high energy modes are roughly constant, then the billiard is ergodic. Only few general results are known for such quantum problems. In fact, there are not many theoretical results concerning the ergodic properties of convex polyhedral billiards in high dimensions. If all angles of the polyhedra are rational, then the billiard is weakly ergodic in the sense that the velocity direction will reach only rational angles (relative to

the initial direction). In general, as long as two neighboring trajectories collide with the same polyhedral faces, their distance will grow only linearly. Once they are far enough as to collide with different faces of the polyhedron, their distance will abruptly increase. Hence, except for very special cases with high symmetry, it seems unlikely that high dimensional convex polyhedra as generated by the training examples will fail to be ergodic.

In order to illustrate this concept, I show below two pictures: Fig (3) is the space covered by a trajectory after 500 collisions and one showing (Fig. (4)) only the collision points. Both the volume and the surface of the polygon are already well covered by the trajectory.

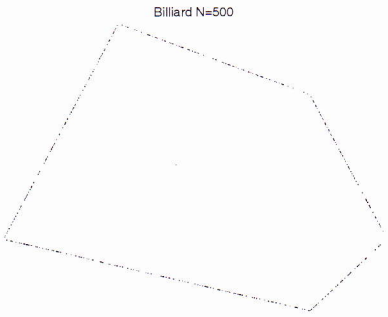

Fig. 4. The 500 collisions points

3 Linear Programming

The *general* linear programming problem is defined as following. Find

$$\max_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{8}$$

subject to m_1 inequalities of the form

$$\boldsymbol{a}^{(\alpha_+)} \boldsymbol{x} < b_{\alpha_+}; \quad \alpha_+ = 1, \ldots, m_1 \tag{9}$$

m_2 equalities

$$\boldsymbol{a}^{(\alpha)} \boldsymbol{x} = b_\alpha; \quad \alpha = m_1 + 1, \ldots, m_1 + m_2 \tag{10}$$

and m_3 inequalities

$$\boldsymbol{a}^{(\alpha_-)} \boldsymbol{x} > b_{\alpha_-}; \quad \alpha_- = m_2 + 1, \ldots, m_1 + m_2 + m_3 \tag{11}$$

Consider the following simple elementary transformations:

- If $a < b$ then $a + s = b$; $s \geq 0$. The auxiliary variable s is called a *slack* variable.

- If x is arbitrary write x as $x = x_+ - x_-$, where now $x_+, x_- \geq 0$,
- If a linear equation system has N unknowns but only $M < N$ linearly independent variables, parameterize the solution in terms of arbitrary $N-M$ variables.

Using these transformations one can rewrite the general LP format into either the *canonical* format with constraints of the form

$$\mathbf{A}\boldsymbol{x} \geq \boldsymbol{r} \tag{12}$$
$$\boldsymbol{x} \geq 0 \tag{13}$$

or into the *standard* form:

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{14}$$
$$\boldsymbol{x} \geq 0 \tag{15}$$

For the geometrical approach the appropriate form is the canonical form Eq. (12). For example, consider the constraints

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 4 \\ x_1 &\leq 2 \\ x_3 &\leq 3 \\ 3x_2 + x_3 &\leq 6 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_3 &\geq 0 \end{aligned} \tag{16}$$

which we will use to illustrate the different algorithms. The corresponding polytope is shown in Fig. (5).

The cost vector $\boldsymbol{c}$ is shown as a dotted line, it could point in either of the two directions, depending on we seek a minimum or a maximum of the linear function (8). All algorithms described below must start from a feasible point belonging to the convex hull. There are only three possibilities: either the feasibility (version) space is empty, the polytope is open and $\boldsymbol{c}$ has its extremum at infinity, or (like in our case) the solution will be one of the faces of the polytope (most probably a vertex). However, if the $\boldsymbol{c}$ would point in the $-z$ direction, then the upper triangle corresponds to the minimal cost.

3.1 The simplex algorithm

The simplex algorithm was developed by Dantzig and implements the following simple but effective idea: once we have a starting feasible solution (usually a vertex) one moves from one vertex to the next so that in each step the cost does not increase. Eventually, the optimal vertex or facet is reached. The starting vertex can be always defined by extending the dimension of the problem (the

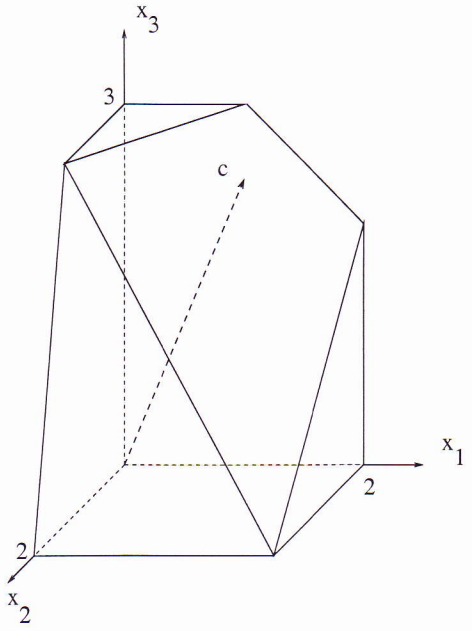

Fig. 5. The polytope defined by the constraints (17). The vector $\boldsymbol{c}$ indicates a possible cost vector.

number of variables in $\boldsymbol{x}$) so that all (hyper)planes will pass through the origin, which is then a good starting point. The first phase of the algorithm tries then to remove these auxiliary variables by maximizing an "artificial" cost function $\boldsymbol{c} = \sum_{aux} \boldsymbol{x}_{aux}$.

This type of solution is illustrate in Fig. 6.

In practice, the simplex algorithm runs quite fast (mostly linear in N or M) but one can construct examples when it has to visit a finite fraction of all vertices, which leads to exponential run times [6]. Since mathematicians are mostly interested in worst case analysis, the question was whether the LP problem is worst case solvable in polynomial time (class $\mathcal{P}$) or not. The breakthrough came with the work of L.G. Khachian, who published in 1979 the ellipsoid algorithm, showing that linear programming is in worst case polynomial on N, M, and on the number of significant digits of the solution vector. For more details, see [6].

3.2 The internal point algorithms

The ellipsoid algorithm was the first among a family of algorithms called interior point algorithms. While the simplex and dual-primal algorithms move on the "surface" of the version polytope, the ellipsoid and other algorithms move "inside" the feasibility space. The ellipsoid algorithm is rather complex but the main idea is simple: given a polytope, fit to it an ellipsoid so that by cutting the space with a plane perpendicular to the cost vector the feasibility space is

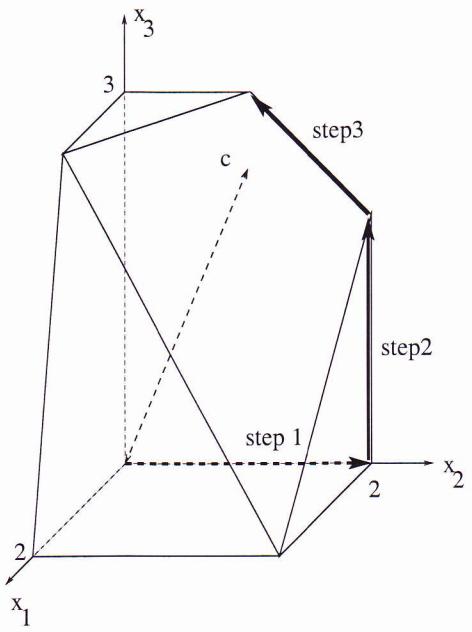

Fig. 6. The simplex algorithm moves from vertex to neighboring vertex

roughly halved. In this way, given a desired numerical precision, the optimal solution will be localized as in bisection in a polynomial number of steps.

The best worst case algorithms known today use the concept of the "analytic center of a set of inequalities" introduced by Sonnevend [7]. See for instance the article by Vaidya [17].

3.3 The billiard algorithm - I

Instead of these complex methods let us consider a simpler approach. Each inequality defines a hyperplane (halfspace), whose normal points always towards the "'outside"' of the polytope. Let us consider these hyperplanes as made from a material behaving like a half mirror. Each hyperplane would then reflect specularly a light ray incident against the normal but lets it pass through when incident from the other side. If such a ray enters the polytope, it will remain trapped inside. The most difficult part of the algorithm is the initialization (finding an interior point), because we do not know a priori if the problem has a solution at all or if that solution is bounded. However, this difficulty is common to all algorithms described here and for "well behaved" problems it is not that difficult to "shot" a ray into the feasible polytope.

Once this is achieved, we compute the trajectory's collision points and their cost. Once a collision point gives a better cost, we move a hyperplane whose normal is antiparallel to the cost vector up to the collision point and restart the billiard. We measure the projections of the trajectory on the cost vector. The

algorithm converged when its fluctuations become smaller than the required precision.

3.4 The billiard algorithm - II

Note that for solving the LP or similar optimization problems we do not really need an ergodic dynamics. In fact, we would like to ensure that all trajectories have an attractive sink at the optimum. Therefore, instead of specular reflections we can bend the reflection angle as much as allowed towards the cost vector. This strategy is illustrated in Fig. 7.

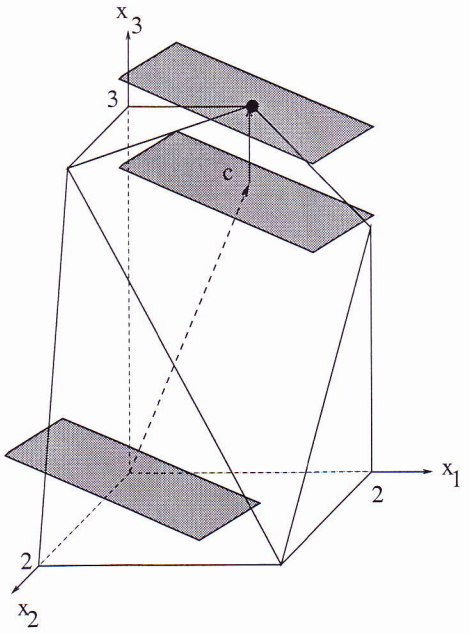

Fig. 7. The billiard algorithm moves from facet to facet. It starts from the origin and runs first parallel to the cost vector (here we seek the maximum cost, not the minimum). When a collision point has a better cost, a large part of the feasibility region will be cut by a plane whose normal is the cost vector (shown schematically).

3.5 Difficult (integer) linear programming problems

If an addition to the usual conditions one requires that the solution vector $\boldsymbol{x}$ is integer, the problem is called *integer linear programming* (ILP) and is $\mathcal{NP}$-hard. The naive expectation to solve the problem as it were a normal LP ("relaxed problem") and then to look for the nearest integer $\boldsymbol{x}$ can go very wrong, as illustrated below.

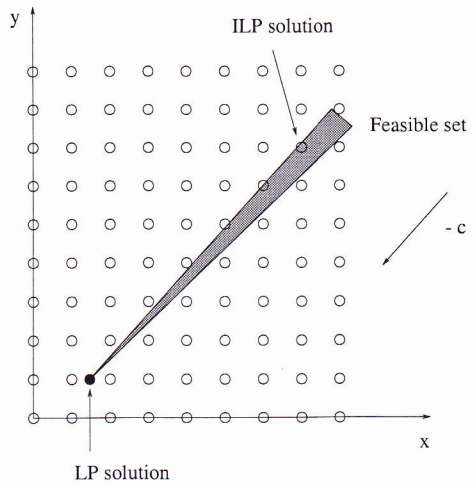

Fig. 8. The relaxed LP (condition on integer solution removed) and the ILP solutions can be very far away

One practical solution to this problem is to introduce *cuts*, additional inequalities reducing the feasibility region but leaving all feasible integer solutions intact. It is not yet clear how the billiard method could accommodate the introduction of such additional Gomory cuts. We need a clever trick forcing the trajectory to run only along paths whose normals are Miller indices.

All $\mathcal{NP}$-hard problems are related through polynomial time algorithms. For example, it is shown below how to formulate the traveling salesman problem as an ILP problem. Assume we have $N+1$ cities $0, 1, 2, \ldots, N$ and the intercity distances $[c_{ij}]$. Let the variable $x_{ij} = 1$ if that edge is in the tour and 0 if not. Then, we require

$$\min z = \sum_{i,j=0}^{N} c_{ij} x_{ij} \tag{17}$$

$$x_{ij} = \{0, 1\} \quad \forall\, i, j \tag{18}$$

$$\sum_{i=0}^{N} x_{ij} = 1 \qquad j = 0, \ldots, N \tag{19}$$

$$\sum_{j=0}^{N} x_{ij} = 1 \qquad i = 1, \ldots, N \tag{20}$$

requiring that only one arc is incident and outbounding from each node. As formulated above, however, this corresponds to the *assignment* problem, which allows for disjoint (cycles) subtours. In order to exclude this possibility, consider all possible decompositions of the tour S into two disjoint subsets, S_1 and S_2,

$S = S_1 + S_2$. For each pair of such cities subsets one must satisfy

$$\sum_{i \in S_1; j \in S_2} x_{ij} = 1 \tag{21}$$

From this formulation it becomes clear where the times is needed when trying to solve exactly the TSP. Of course, there are more economic formulations of the TSP, leading to mixed (both real and integer variables) LP's [19,20]. These are the methods used together with a bound and cut strategy to solve exactly quite large TSP problems (over 20 000 cities).

4 The Bayes Perceptron

As shown in Figure 9, a *Perceptron* [8] is a network consisting of N (binary or real) inputs x_i and a single binary output neuron. The output neuron sums first all inputs with the corresponding synaptic weights w_i, then performs a threshold operation according to

$$\sigma = sign(\sum_{i=1}^{N} x_i w_i - \theta) = sign(\boldsymbol{x}\boldsymbol{w} - \theta) = \pm 1 \tag{22}$$

In general, the synaptic weights and the threshold can be arbitrary real numbers. A network where $\boldsymbol{w}$ has only binary components is called a *binary* Perceptron. The output unit has binary values $\sigma = \pm 1$ and labels the class to which the input vector belongs. Both the weight vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_N)$ and the threshold θ should be determined from a predefined set of labeled M examples (the training set).

Our protocol assumes that both the training and test examples are drawn independently and identically from the distribution $P(\boldsymbol{\xi})$. The simplest way to generate such a rule is to define a *teacher* network with the same structure as shown in Fig. 9, providing the correct class label $\sigma = \pm 1$ for each input vector. Given a set of M examples $\{\boldsymbol{\xi}^{(\alpha)}\}_{\alpha=1}^{M}$, $\boldsymbol{\xi}^{(\alpha)} \in \mathbb{R}^N$ and their corresponding binary class label $\sigma_\alpha = \pm 1$, the task of the learning algorithm is to find a *student* network which mimics the teacher by choosing the network parameters such as to classify correctly the training examples. Eq. (22) implies that the learning task consists of finding a hyperplane $\boldsymbol{w}\boldsymbol{x} = \theta$ separating the positive from the negative labeled examples:

$$\begin{array}{lll} \boldsymbol{w}\boldsymbol{\xi}^{(\alpha)} & \geq \theta_1, & \sigma_\alpha = 1 \\ \boldsymbol{w}\boldsymbol{\xi}^{(\beta)} & \leq \theta_2, & \sigma_\beta = -1 \\ \theta_1 & \geq \theta \geq & \theta_2 \end{array} \tag{23}$$

The typical situation is that either none or infinitely many such solution exist. The vector space whose points are the vectors $(\boldsymbol{w}, \theta)$ is called the *version space* and its subspace satisfying Eq. (23) the *solution polyhedron.*

In theory, one knows that for a given training set the optimal Bayes decision [9] implies an average over all $\{\boldsymbol{w},\ \theta\}$ solutions satisfying Eq. (23). Since each solution is perfectly consistent with the training set, in the absence of any other a priori knowledge, one must consider them as equally probable. This is not the case, for instance, in the presence of noise. The best strategy is then to associate with each point in the version space a Boltzmann weight whose energy is the squared error function and whose temperature depends on the noise strength [10].

For examples independently and identically drawn from a constant distribution ($P(\boldsymbol{\xi}) = const$) Watkin has shown that in the thermodynamic limit the single Perceptron corresponding to the the center of mass of the solution polyhedron has the same generalization probability as the Bayes-decision [11]. On the practical side, known learning algorithms like Adaline [12,13] or the maximal stability (or maximal margin) Perceptron (MSP) [14–16] are not Bayes-optimal. In fact, if all input vectors have the same length, then the maximal stability Perceptron network corresponds to the center of the largest hypersphere inscribed into the solution polyhedron.

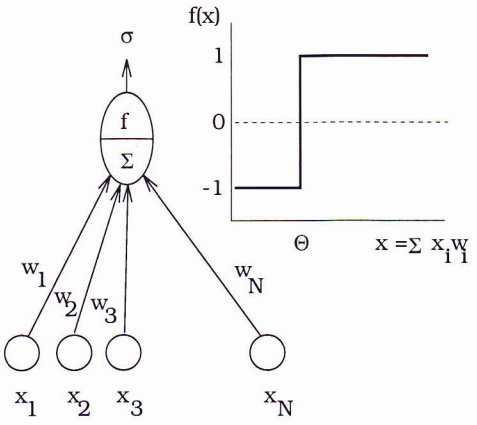

Fig. 9. The Perceptron as neural network

4.1 The geometry of Perceptrons

Consider a set of M training examples, consisting of N-dimensional vectors $\boldsymbol{\xi}^{(\nu)}$ and their class σ_ν, $\nu = 1, \ldots, M$. Now let us introduce the $N+1$-dimensional vectors $\boldsymbol{\zeta}^{(\nu)} = (\sigma_\nu \boldsymbol{\xi}^{(\nu)}, -\sigma_\nu)$ and $\boldsymbol{W} = (\boldsymbol{w}, \theta)$. In this representation Eq. (23) becomes equivalent to a standard set of linear inequalities

$$\boldsymbol{W}\boldsymbol{\zeta}^{(\nu)} \geq \Delta > 0 \tag{24}$$

The parameter $\Delta = \min_{\{\nu\}} \boldsymbol{W}\boldsymbol{\zeta}^{(\nu)}$ is called the *stability* of the linear inequality system. A bigger Δ implies a solution which is more robust against small changes in the example vectors.

The vector space whose points are the examples $\boldsymbol{\zeta}^{(\nu)}$ is the *example space*. In this space $\boldsymbol{W}$ corresponds to the normal vector of a $N+1$ dimensional hyperplane. The *version space* is the space of the vectors $\boldsymbol{W}$. Hence, a given example vector ζ corresponds here to the normal of a hyperplane. The inequalities (24) define a convex polyhedral cone whose boundary hyperplanes are determined by the training set $\left\{\boldsymbol{\zeta}^{(\nu)}\right\}_{\nu=1}^{M}$.

How can one use the information contained in the example set for making the best average prediction on the class label of a new vector drawn from the same distribution? Each new presented example $\boldsymbol{\zeta}^{(new)}$ corresponds to a hyperplane in version space. The direction of its normal is defined up to a $\sigma = \pm 1$ factor, the corresponding class.

The best possible decision for the classification of the new example follows the Bayes scheme: for each new (test) example generate the corresponding hyperplane in version space. If this hyperplane does not intersect the solution polyhedron, consider the normal to be positive when pointing to the part of the version space containing the solution polyhedron. Hence, all Perceptrons satisfying Eq. (24) will classify unanimously the new example. If the hyperplane cuts the solution polyhedron in two parts, point the normal towards the bigger half. Therefore, the decision which minimizes the average generalization error is given by evaluating the average measure of *pro* vs the average measure of *contra* votes of all Perceptrons making no errors on the training set.

The solution polyhedral cone is either open or is defined on the unit $N+1$-dimensional hypersphere (see also Fig. 10b for an illustration).

The practical question is now to find simple approximations for the center of mass (Bayes point). One possibility is to consider the Perceptron with *maximal stability*, defined as following:

$$\boldsymbol{W}_{MSP} = \arg\max_{\boldsymbol{W}} \Delta; \quad \boldsymbol{W}\boldsymbol{W} = 1 \tag{25}$$

or, equivalently, as

$$\boldsymbol{w} = \arg\max_{\boldsymbol{w}} \{\theta_1 - \theta_2\}; \quad \boldsymbol{w}\boldsymbol{w} = 1 \tag{26}$$

where $\theta = (\theta_1 + \theta_2)/2$ and $\Delta = (\theta_1 - \theta_2)/2$. The quadratic conditions $\boldsymbol{W}\boldsymbol{W} = 1$ [$\boldsymbol{w}\boldsymbol{w} = 1$] are necessary because otherwise one could multiply $\boldsymbol{w}$ and $\theta_{1,2}$ by a large number, making $\theta_1 - \theta_2$ and thus Δ arbitrary large.

Eq. (26) has a very simple geometric interpretation, shown in Fig. 10. Consider the convex hulls of the vectors $\boldsymbol{\xi}^{(\alpha)}$ belonging to the positive examples $\sigma_\alpha = 1$ and of those in the negative class $\boldsymbol{\xi}^{(\beta)}$, $\sigma_\beta = -1$, respectively. According to Eq. (26), the Perceptron with maximal stability corresponds to the slab of maximal width one can put between the two convex hulls (the 'maximal dead zone' [18]). Geometrically, this problem is equivalent (dual) to finding the

direction of the shortest line segment connecting the two convex hulls (the minimal connector problem). Since the dual problem minimizes a quadratic function subject to linear constraints, it is a *quadratic programming problem.* By choosing $\theta = \frac{\theta_1+\theta_2}{2}$ (dotted line in Fig. 10) one obtains the maximal stability Perceptron (MSP).

The direction of $\boldsymbol{w}$ is determined by at most $N+1$ vertices taken from both convex hulls, called *active constraints.* Fig. 10a shows a simple two dimensional example, the active constraints are labeled by A, B, and C, respectively. The version space is three dimensional, as shown in Fig. 10b. The three planes represent the constraints imposed by the examples A (left plane), B (right plane), and C (lower plane). The bar points from the origin to the point defined by the MSP solution. The sphere corresponds to the normalization constraint. If the example vectors $\boldsymbol{\xi}^{(\nu)}$ have all the same length $\mathcal{N}$, then Eqs. (24, 25) imply that the distance between the $\boldsymbol{W}_{MSP}$ and the hyperplanes corresponding to active constraints are all equal to $\frac{\Delta_{\max}}{\mathcal{N}}$. All other hyperplanes participating in the polyhedral cone are further away. Accordingly, the maximal stability Perceptron corresponds to the center of the largest circle inscribed into the spherical triangle defined by the intersection of the unit sphere with the solution polyhedron, the point where the bar intersects the sphere in Fig. 10b.

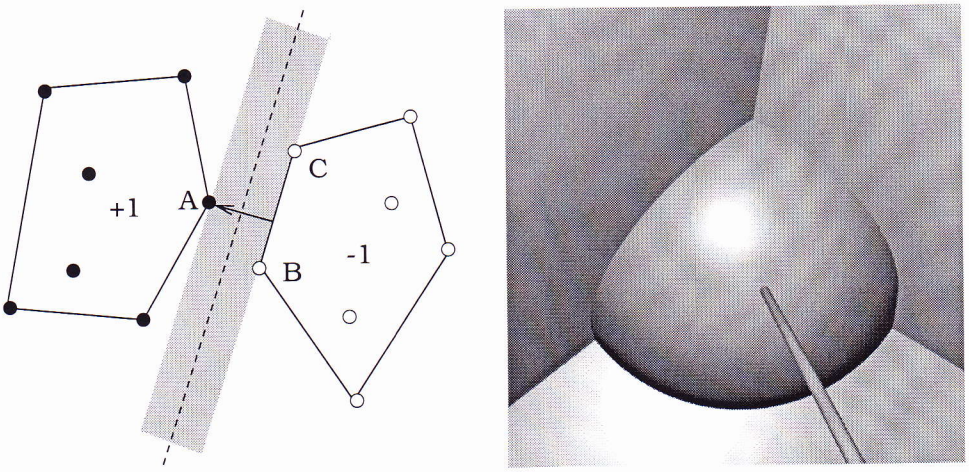

Fig. 10. *a) The Perceptron with maximal stability in example space. b) The solution polyhedron (only the bounding examples A, B, and C are shown). See text for details.*

A fast algorithm for computing the minimal connector requiring in average $O(N^2M)$ operations and $O(N^2)$ storage place can be found in [16].

4.2 How to play billiard in version space

Each billiard game starts by first placing the ball(s) on the pool. As already discussed for the LP, this is not a trivial task. In our case, the maximal stability Perceptron algorithm [16] does it or signals that a solution does not exist. Alternately, one can use the half-rejecting mirror method (see the LP-algorithms) to find an interior point. The trajectory is initiated by generating at random a unit direction vector $\boldsymbol{v}$ in version space.

The basic step consists of finding out where - on which hyperplane - the next collision will take place. The idea is to compute how much time the ball needs until it eventually hits each one of the M hyperplanes. Given a point $\boldsymbol{W} = (\boldsymbol{w}, \theta)$ in version space and a unit direction vector $\boldsymbol{v}$, let us denote the distance along the hyperplane normal $\boldsymbol{\zeta}$ by d_n and the component of $\boldsymbol{v}$ perpendicular to the hyperplane by v_n. In this notation the flight time needed to reach this plane is given by

$$\begin{aligned} d_n &= \boldsymbol{W}\boldsymbol{\zeta} \\ v_n &= \boldsymbol{v}\boldsymbol{\zeta} \\ \tau &= -\frac{d_n}{v_n} \end{aligned} \tag{27}$$

After computing all M flight times, one looks for the smallest positive $\tau_{min} = \min_{\{\nu\}} \tau > 0$. The collision will take place on the corresponding hyperplane. The new point $\boldsymbol{W}'$ and the new direction $\boldsymbol{v}'$ are calculated as

$$\boldsymbol{W}' = \boldsymbol{W} + \tau_{min}\boldsymbol{v} \tag{28}$$

$$\boldsymbol{v}' = \boldsymbol{v} - 2v_n\boldsymbol{\zeta} \tag{29}$$

This procedure is illustrated in Fig. 11a. In order to estimate the center of mass

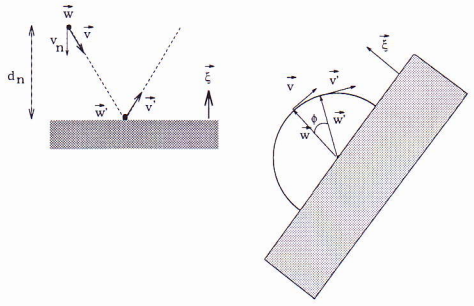

Fig. 11. *Bouncing in version space. a) Euclidean b) spherical geometry.*

of the trajectory one has first to normalize both $\boldsymbol{W}$ and $\boldsymbol{W}'$. By assuming a constant line density one assigns to the (normalized!) center of the segment $\frac{\boldsymbol{W}'+\boldsymbol{W}}{2}$ the length of the vector $\boldsymbol{W}' - \boldsymbol{W}$. This is then added to the actual

center of mass - as when adding two parallel forces of different lengths. In high dimensions ($N > 5$), however, the difference between the mass of the full $N+1$ dimensional solution polyhedron and the mass of the bounding N dimensional boundaries becomes negligible. Hence, we could as well just record the collision points, assign them the same mass density, and construct their average.

Note that by continuing the trajectory beyond the first collision plane one can sample also regions of the solution space where the $\boldsymbol{W}$ makes one, two, etc. mistakes (the number of mistakes equals the number of crossed boundary planes). This additional information can be used for taking an optimal decision when the examples are noisy [10]. This method, which we call the "onion algorithm", has been shown to be superior to all other ones. This is not surprising, since we have here some free parameters to adjust on the test set.

Since the polyhedral cone is open, the implementation of this algorithm must take into account the possibility that the trajectory might escape to infinity. The minimal flight time becomes then very large, $\tau > \tau_{max}$. When this exception is detected, a new trajectory is started from the maximal stability Perceptron point in a yet another random direction. Hence, from a practical point of view, the polyhedral solution cone is closed by a spherical shell with radius τ_{max} acting as a special 'scatterer'. This *flipper* procedure is iterated until enough data is gathered.

If we are class conscious and want to remain in the billiard club, we must do a bit more. As explained above, the solution polyhedral cone can be closed by normalizing the version space vectors. The billiard is now defined on a curved space. However, the same strategy works also here if between subsequent collisions one follows geodesics instead of straight lines. Fig. 11b illustrates the change in direction for a small time step, leading the the well known geodesic differential equation on the unit sphere:

$$\dot{\boldsymbol{W}} = \boldsymbol{v} \tag{30}$$

$$\dot{\boldsymbol{v}} = -\boldsymbol{W} \tag{31}$$

The solution of this equation costs additional resources. Actually, the solution of the differential equation is strictly necessary only when there are no bounding planes on the actual horizon [1]. Once one or more boundaries are 'visible', the choice of the shortest flight time can be evaluated directly, since in the two geometries the flight time is monotonously deformed. Even so, the flipper procedure is obviously faster.

Both variants deliver interesting additional informations, like the mean escape time of a trajectory or the number of times a given border plane (example) has been bounced upon. The collision frequency classifies the training examples according to their 'surface' area in the solution polyhedron - a good measure of their relative 'importance'.

Note that billiard algorithm has been successfully generalized to kernel classifiers [22] by us and others. Kernel classifiers perform *implicitly* a nonlinear trans-

[1] Assuming light travels along Euclidean straight lines

formation from $\boldsymbol{x} \to \boldsymbol{\Phi}(\boldsymbol{x})$, where the $\boldsymbol{\Phi}$ are highly or infinitely dimensional vectors. In order to generalize the learning method described above to kernels, one must rewrite the algorithm in terms of dot-products of the form $q_{ij} = \boldsymbol{\xi}^{(i)}\boldsymbol{\xi}^{(j)}$. If this is possible, one makes the substitution $q_{ij} \leftarrow \boldsymbol{\Phi}(\boldsymbol{\xi}^{(i)})\boldsymbol{\Phi}(\boldsymbol{\xi}_{(j)}) = k(\boldsymbol{\xi}^{(i)}, \boldsymbol{\xi}^{(j)})$, where $k(\boldsymbol{x}, \boldsymbol{y})$ is a (positive definite) Mercer-kernel. The output function of the Perceptron becomes now an operator:

$$\boldsymbol{w}\boldsymbol{x} = \sum_{\alpha} a_\alpha k(\boldsymbol{x}, \boldsymbol{\xi}^{(\alpha)}) \tag{32}$$

where the α index runs over all examples which contributed to the Bayes point $\boldsymbol{w}$.

The interesting fact about the kernel classifiers is that although the dimension where the separation might be infinite (if the kernel is a Gauss function, for example) their estimated error is bounded (in the case of errorless learning) by $O(R/\Delta)$, where R is the radius of the smallest circumscribed circle around all examples and Δ is the distance between the two convex hulls of *active* positive and negative examples. More details, results and performance tests can be find in [21,22]

5 Bayesian inference for the mixture assignment problem

This problem was originally considered by Prof. Brumsack at ICBM in Oldenburg for identifying sources of industrial air pollution. It can be generally formulated as following: assume we create a database containing M substances generated by different processes. For example, these substances are particles polluting the air, produced by different industrial processes. Each substance is analyzed with a mass spectrograph and leads to a (discrete) set of normalized spectrograms containing N lines each. Hence, each substance is a vector

$$\boldsymbol{\xi}^{(\alpha)} = (\xi_1^{(\alpha)}, \ldots, \xi_N^{(\alpha)}) \tag{33}$$

normalized as $\sum_{i=1}^{M} \xi_i^{(\alpha)} = 1$, for all $\alpha = 1, \ldots, M$.

Now we are faced with an unknown substance (particle) and its spectrogram, $\boldsymbol{x} = (x_1, \ldots, x_N)$. We are asked to estimate using the database how likely it is that this particle was produced by a particular process (or industrial plant).

One way to proceed is to compute the minimal distance between the convex hull of "'example"' substances in our catalog and $\boldsymbol{x}$. If this distance is positive (larger than, say, epsilon) we can uniquely identify a point belonging to the convex hull of the catalog examples. As explained in the case of a Perceptron, the shortest distance between this point and the database convex hull results in a unique convex combination of the form

$$\boldsymbol{x} = \sum_{\alpha} a_\alpha \boldsymbol{\xi}^{(\alpha)} \tag{34}$$

Here, we can interpret a_α as the probability that substance α contributed to the pollution. If our catalog is complete, however, the unknown substance

will lie inside the database CH, in which case the minimal distance to convex hull is zero and the convex combination Eq. (34) is not unique. Assuming this is the most frequent case, the interesting question is to rank the possible catalog substances as possible "sources" of the "mixture" $\boldsymbol{x}$. It is not hard to see why this problem is important (especially in culinary art!) but also why it is very difficult to make scientific statements about it.

First let us remark that due to mass conservation

$$\sum_{\alpha=1}^{M} a_\alpha \xi_i^{(\alpha)} = x_i \quad i = 1, \ldots, N \tag{35}$$

and that the coefficients a are convex:

$$\sum_{\alpha=1}^{M} a_\alpha = 1; \quad a_\alpha \geq 0 \tag{36}$$

We are interested in the joint distribution $P(a_1, \ldots, a_M)$. It seems very difficult to solve that task. For Bayesian classification it might be enough to estimate the marginal probabilities $p_i = P(a_i)$. This we can do by considering the version space of $\boldsymbol{a}$ points. If $M > N + 1$, we eliminate the equality constraints and play billiard in the remaining $M - N - 1$ dimensional space, accumulating statistics from the collision points. That allows us to build a histogram approach to p_i and estimate the average, the most probable value, and the variance of a_i. I mention this problem here because it is a very interesting one from both a theoretical and a practical point of view. Also, a lot of questions are open, for example, how to take into account correctly a priori probabilities of finding cataloged substances.

6 Conclusions

Although the billiard method is in practice slower than other methods, like simplex or the primal-dual implementation I developed for finding the maximal margin Perceptron [16], it has some evident advantages

- It is fully parallelizable, since we can gather data from two or more billiard trajectories running in parallel,
- Is conceptually simple to understand and to program, hence it can be easily adapted to special situations,
- It provides a powerful method of sampling (posterior) probability distributions for many optimization problems. This is illustrated by the Bayesian inference problem for mixture assignment, a kind of inverse diet-problem.

With computing power becoming an Aldi commodity, it is not hard to see that statistical (or Bayesian, if you wish) combinatorial optimization problems will be solved routinely on PC's or even handheld devices. Although the above considerations might seem to you rather academic, such algorithms can solve surprisingly well real life problems. I have no doubt that they will become the core of a new era in information technology.

References

1. J. Moser: *Geometry of quadrics and spectral theory* In: The Chern Symposium, ed. by Y. Hsiang et al (Springer , Berlin, Heidelberg 1980) p. 147
2. J. R. Kuttler and V. G. Sigillito: *Eigenvalues of the Laplacian in two dimensions*: SIAM Review **26**, 163 (1984)
3. H.R. Krishnamurthy, H.S. Mani, and H.C. Verma: *Exact solution of Schrödinger equation for a particle in a tetrahedral box*: Journal of Physics A **15**, 2131 (1982)
4. L. A. Bunimovich: *On the ergodic properties of nowhere dispersing billiards*: Communications in Mathematical Physics **65**, 295 (1979)
5. L. A. Bunimovich and Y. G. Sinai: *Markov partitions for dispersed billiards*: Communications in Mathematical Physics **78**, 247 (1980)
6. C. H. Papadimitriou and K. Steiglitz: *Combinatorial Optimization*, (Prentice-Hall, Englewood Cliffs, 1982)
7. Gy. Sonnevend: *New algorithms based on a notion of 'centre' (for systems of analytic inequalities) and on rational extrapolation.* In Trends in Mathematical Optimization - Proceedings of the 4th French-German Conference on Optimization, Irsee 1986 Eds: K.H. Hoffmann, J.-B. Hiriart-Urruty, C. Lemarechal and J. Zowe. ISNM, Vol. **84** (Birkhauser, Basel, 1988)
8. F. Rosenblatt: *The perceptron: a probabilistic model for information storage and organization in the brain* Psychological Review **65** 386 (1961)
9. M. Opper and D. Haussler: *Generalization performance of Bayes optimal classification algorithm for learning a Perceptron*, Physical Review Letters **66** 2677(1991)
10. M. Opper and D. Haussler: *Calculation of the learning curve of Bayes optimal classification algorithm for learning a Perceptron with noise.* In: IVth Annual Workshop on Computational Learning Theory (COLT91), Santa Cruz 1991, (Morgan Kaufmann, San Mateo, 1992), pp 61-87
11. T. Watkin: *Optimal learning with a neural network*, Europhysics Letters **21** (1993) 871
12. B. Widrow and M. E. Hoff: *Adaptive switching circuits*, 1960 IRE WESCON Convention Record,(New York, IRE), 96
13. S. Diederich and M. Opper: *Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules*, Physical Review Letters **58** 949 (1987)
14. D. Vapnik: *Estimation of Dependencies from Empirical Data*, (Springer Verlag, 1982) - see Addendum I
15. J. Anlauf and M. Biehl: *The AdaTron: an adaptive Perceptron algorithm*, Europhysics Letters **10** (1989) 687
16. P. Rujan: *A fast method for calculating the Perceptron with maximal stability*, Journal de Physique (Paris) I **3** 277 (1993)
17. P. M. Vaidya: *A new algorithm for minimizing a convex function over convex sets*, In: Proceedings of the 30th Annual FOCS Symposium, Research Triangle Park, NC, 1989 (IEEE Computer Society Press, Los Alamitos, CA, 1990) 338-343
18. P. F. Lampert: Designing pattern categories with extremal paradigm information, in *Methodologies of Pattern Recognition*, M. S. Watanabe (Ed.), (Academic Press, NY, 1969), p 359
19. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys (Eds): *The Traveling Salesman Problem*, (John Wiley, NY, 1984)
20. M. Padberg and G. Rinaldi: *Optimization of a 532-city symmetric traveling salesman by branch and cut*, Oper. Res. Lett. **6** (1987) 1 and *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, IASI preprint R. 247, 1988

21. P. Rujan: *Playing billiard in version space* Neural Computation **9** 197 (1997)
22. P. Rujan and M. Marchand: *Computing the Bayes Kernel Classifier* In: *Advances in Large Margin Classifiers*, ed. by A. Smola et al (MIT Press , Cambridge, London 2000) pp.329-348