Flexible Exploitation of Space Coherence to Detect Collisions of Convex Polyhedra

Claudio Mirolo
Dip. di Matematica e Informatica - Univ. di Udine
Via delle Scienze, 206 - 33100 Udine - Italy
claudio@dimi.uniud.it

Enrico Pagello
Ist. Ladseb - C.N.R. di Padova
Corso Stati Uniti, 4 - 35127 Padova - Italy
& Dip. di Elettronica e Informatica - Univ. di Padova
epv@dei.unipd.it

Abstract
The subject of this paper is a fast algorithm to compute collision translations for pairs of convex polyhedra with some interesting features. From a theoretical viewpoint, besides the novelty of the approach, the polylog asymptotic trend in the average case is as good as that of the best algorithms proposed to solve similar problems. On the other hand, the measured performances to detect possible collisions from scratch are satisfactory, and this is especially true in the cases where the bodies do not collide. But the most peculiar feature is a simple and flexible mechanism to exploit spatial coherence in a continuous range, which distinguishes this algorithm from all the other proposals we know. Furthermore, the nature of the approach is such that the self-tuning capability is attained at negligible additional costs even for unrelated collision tests. After a brief outline of the main ideas characterizing the approach, a set of numerical results are summarized. The proposed algorithm may be appropriate to plan collision-free paths, both on-line and off-line, on the basis of fine-grain descriptions of the objects in the workspace.

1. Introduction
In this paper we discuss the performances of a fast algorithm, with additional potential for incremental computations, solving the following problem in three-space: Given two convex polyhedra $P$ and $Q$, compute the collision translation for $P$ moving in direction $d$. The key idea characterizing our approach is that collision detection for two convex bodies can be reduced to collision detection for pairs of planar sections and minimization of a bivariate convex function. As pointed out in [25], where one of the author introduces this novel approach and discusses its expected benefits on a precise theoretical ground, this idea leads to the design of an algorithm which runs in $O(\log n)$ time in the average, where $n$ is the total number of vertices.

The above asymptotic polylog-trend and the experimental performances reported in [27] refer to computations of collision configurations from scratch, i.e., when no previous proximity information is available. In practice, this is exactly what we have to do if we are using collision detection as a tool to build or update a characterization of the configuration space starting from the workspace model. In this respect, the numerical experiments in [24] show that, in order to plan free paths in cluttered environments, we need only few collision probes for each obstacle, relative to unrelated motions. A different situation arises if several collision tests are carried out after repeated short movements of the objects in the workspace, which is the typical scenario of on-line motion planning, simulation, animation, computer aided design. It makes then sense to try to exploit the spatial and temporal coherence, which suggests that the outcomes of the collision tests relative to close workspace states are likely to give rise to close collision configurations. Suitable algorithms designed to solve proximity problems of this kind are referred to as incremental algorithms. Their efficiency rests on the capability of exploiting the knowledge of a previous result (initialization) in order to speed up the computation of the updated proximity measure or property. However, most of the known algorithms show poor performances if no previous information is available for initialization.

A remarkable feature of the algorithm we propose is its simple and flexible mechanism to exploit spatial coherence in a continuous range. To the best of our knowledge, indeed, we are not aware of any other related algorithm having this property. The very nature of our approach allows us to endow the algorithm with a self-tuning capability — and at negligible additional costs even for unrelated collision tests — by simply refining the way we compute the splitting points during the minimization process. This has two favorable consequences: first, the algorithm is still well-performing to compute collision translations from scratch, i.e., without initialization; second, the closer two consecutive workspace states are, the faster the proximity measure can be updated (in the average).

Here the main focus is on the results of a large set of numerical experiments on pairs of polyhedra, which substantiate some interesting properties of the algorithm, namely:

(i) fast computation from scratch in complex cases;
(ii) faster detection from scratch (about one fourth of the time) that unbounded translations are collision-free;
(iii) very fast incremental computation (about one tenth of the time) after short movements, i.e., after movements of about the size of the polyhedron edges;
(iv) as-fast-as-possible incremental computation after motions of arbitrary size.

Motivations
The proposed algorithm may be appropriate to plan collision-free paths both on-line and off-line, on the basis of fine-grain descriptions of the workspace. A complex representation of the objects is a usual situation for the geometric modelers based on CAD systems, as pointed out in [1]. Having this in mind, our recent work has tried to achieve two main goals: designing fast algorithms to detect collisions for simple motions and approaching the characterization of the configuration space via systematic collision detection. Further support to this research also comes from some previous work of the authors:

• Collision detection as a primitive operation is suitable for incremental and dynamic characterization of the configuration space, [26].

• A planning strategy exploiting collision detection to incrementally build and refine the representation of the configuration space has been successfully experimented in two dimensions, [24].

Related work
There is a rich literature on proximity measures and properties relative to convex polytopes, since an efficient solution to these problems is thought to be critical to the development of useful tools not only for robot motion planning, as recognized in [13], but also in other fields such as simulation, animation and virtual reality, see, e.g., [5]. The complexity of the settings arising in such applications has fostered research on incremental
algorithms. [18, 19, 20, 2, 31, 32, 16]. A recent survey on collision detection can also be found in [21]. If we consider the trend in the worst-case, the best solutions of various proximity problems (intersection detection, collision detection, distance, depth of collision) exploit the properties of the hierarchical representations and require \( O(\log^2 n) \) time, [7, 8, 42]. However, we use a different kind of solid representation deriving from the drum-based structure introduced in [4, 6].

Much work on proximity problems has been done in the field of robot path planning and a variety of proposals witness the persisting interest in the proximity problems and the growing attention to the computational costs. Several algorithms focus on convex objects, e.g., [9, 10, 35, 11, 36, 40]. Of particular significance for the analysis of the role of the proximity measures as tools for motion planning and for the generality of the approach is [3]. Recently, considerable efforts have also addressed the case of general bodies: to define different measures of penetration, [30]; to speed up the elementary interference tests for polyhedron vertices, edge and simple faces (triangles), [38, 14, 34], or for more general bounding curves and surfaces, [39]; to design hierarchical representations and related algorithms, [33, 5, 20, 37, 41, 15, 22, 17].

Some basic work on convex minimization is also relevant to our approach; however, all the necessary tools can easily be found, for instance, in [29]. Finally, as a quite different example of application of the properties of multivariate convex functions to proximity problems, we can mention the polylog intersection detection technique proposed in [12]. The algorithm designed there may work in more than three dimensions, but it uses preprocessed representations which are only invariant by translation, and must be recomputed whenever the orientation of a polyhedron changes.

Organization of the paper

The organization of the paper is as follows. In section 2 we introduce the approach with a brief summary of the main ideas and of the results discussed in [25, 27]. The incremental version of the algorithm, which is the main contribution presented in this paper, is the subject of section 3. Finally, the experimental results demonstrating the peculiar properties of the algorithm are summarized and discussed in section 4.

2. Summary of the Previous Work

The technique discussed in this paper develops from some more theoretical works on a convex-minimization approach to collision detection, [25], and follows the relevant refinements outlined in [27]. So, before going into the core topics presented here, it may be helpful to summarize what is important to know about our previous results.

2.1. Collision detection and convex minimization

Relative to [25], two main points are worth mentioning. First point, contact translations for two convex bodies \( P \) and \( Q \) can be reduced to minimization of a bivariate convex function which represents contact translations relative to pairs of planar sections of \( P \) and \( Q \). More formally, we can prove the following proposition:

**Proposition 2.1** - Let \( P, Q \) be two closed convex regions, \( d \) a direction in the space, \( \{ \rho_P(x) \mid x \in \mathbb{R}^d \} \) and \( \{|\sigma(y)| \mid y \in \mathbb{R}^d \} \) two independent families of parallel planes. Then

\[
\theta(x,y) = \text{coll}(d, P \cap \rho_P(x), (Q \cap \sigma(y)))
\]

is a convex function with a bounded domain in \( \mathbb{R}^2 \).

In the above statement, \( \text{coll}(d,X,Y) \) denotes the extent of the collision translation in direction \( d \) for \( X \) and \( Y \), i.e., the least \( t \in \mathbb{R} \) such that \( \text{dist}(p + td, \{p \in X \mid p \in Y \}) = 0 \); \( \rho_P(x) \) and \( \sigma(y) \) identify the planes by their distances \( x, y \in \mathbb{R} \) from two independent reference planes. The meaning of the proposition is also illustrated in Figure 2.1 for two convex polyhedra. It tells us that if we are able to compute collision translations for pairs of polygons in the space, then we can converge to the collision configuration relative to two convex polyhedra by standard convex minimization. This is not yet a satisfactory result, since it does not guarantee that we can find an exact solution in a finite number of steps.

2.2. Minimization on a discrete domain

Thus, we come to the second point: how to transform a search problem on a continuous domain into a search problem on a discrete domain. In [25] the chosen discrete domain is the set of hyperrectangles of an isotonic grid. The interesting result is that, under reasonable assumptions, the hyperrectangle containing the point of minimum can be found in \( O(\log n) \) minimization steps in the average, where \( n \) is the size of the grid. For our purposes, the grid is planar, an upper bound to \( n \) is the number of vertices of the polyhedra \( P \) and \( Q \) and the cost of a minimization step is the (log-time, [23]) cost of collision detection for a pair of polygons in the space. On the basis of these ideas and after solving a few related (sub)problems and putting all the pieces together, we end up with an algorithm that computes collision translations for pairs of polyhedra with \( O(n) \) vertices in \( O(\log^2 n) \) time in the average and \( O(\log^2 n) \) in the worst case.

2.3. Mark of the convex function

Further contributions to exploit the nice properties of the convex function \( \varphi \) are sketched in [27]. Being \( P \) and \( Q \) polyhedra, it is not surprising to see that \( \varphi \)'s graph is faceted and projects into a partition of \( \varphi \)'s domain. Such a partition can be extended to the rectangle \( \{ (x,y) \mid P \cap \rho_P(x,\sigma(y)) \neq \emptyset \} \) of all the pairs of planar sections on the basis of a suitable invariant property applying to each cell of the partition. We refer to such a partition as the mark of \( \varphi \). To understand the invariant property, consider a standard minimization technique, such as the method of centers of gravity, [29]. Basically, a convex region containing the point of minimum is repeatedly shrunk by cutting off a slice at each step with a cut line through the centroid. For points in \( \varphi \)'s domain, the orientation of the corresponding cut line depends on the gradient of \( \varphi \), hence it does not change if the cut point shifts within the same cell of the mark. So, also the cells of the mark outside \( \varphi \)'s domain are defined in such a way that the orientation of the cut lines is invariant. As a result, we can get rid of the grid mentioned in section 2.2 and use the mark of the convex function instead, since it is also a suitable discrete structure to be searched for the point of minimum.

2.4. Improved convex minimization

The local properties of the convex function \( \varphi \) in a neighborhood of the point \((x,y)\) can be determined from the output of the algorithm used to detect collisions of the pair of planar sections \( P \cap \rho_P(x,\sigma(y)) \) and \( Q \cap \sigma(y) \) in the space. Given the information on either the contact or the separation of the planar sections, we can build the cut line through \((x,y)\) in constant time. We can do even more: we can find the most favorable point of the corresponding cell of the mark, i.e., the point such
that the cut line gets closest to the point of minimum; by doing this, we speed up the search and we discretize the minimization process by discarding the whole cell from further consideration. From a technical viewpoint, it is convenient to give a slightly more subtle definition of the mark in order to find the "most favorable" point of a cell in log-time, but we will not discuss such details here.

However, we can notice that a convenient characterization of the mark distinguishes three types of cells, according to the possible results of collision detection for pair of sections corresponding to points \((x,y)\) in a cell:

a) **Edge contact** – a pair of edges of the planar sections \(P \cap p(x)\) and \(Q \cap \sigma(y)\) get into contact in a collision configuration;

b) **Vertex contact** – a vertex of \(P \cap p(x)\) hits the interior of \(Q \cap \sigma(y)\) in a collision configuration;

c) **Separation configuration** – the planar sections \(P \cap p(x)\) and \(Q \cap \sigma(y)\) do not collide.

The qualitative properties listed above are indeed invariant within each cell of the mark.

### 2.5. Collision detection for pairs of polyhedra

Figure 2.2 outlines informally algorithm **PhPhColl**, which is designed to solve the collision problem for two polyhedra \(P, Q\) and a translation direction \(d\). The region \(M\) is initialized to represent the rectangular set of all pairs of planar sections. Then, at each iteration of the minimization loop: (1) the centroid \(c\) of \(M\) is computed; (2) the cut line \(s\) through \(c\) is computed and (3) it is shifted to the most favorable point \(q\) in the mark cell containing \(c\); finally, (4) the region \(M\) is updated by cutting off the slice on one of the sides of \(s\). Eventually, \(q\) is recognized to be either the point of minimum or a point "proving" that \(\phi\)'s domain is empty. In both cases it represents the solution.

```plaintext
Algorithm PhPhColl

input: two convex polyhedra \(P, Q\), and a direction \(d\)

begin

\(M := \) rectangle of all pairs of planar sections of \(P, Q\);

repeat

1. \(c := \) centroid of \(M\);
2. \(s := \) cut line through \(c\);
3. cell-shift \(s\) to point \(q\);
4. update \(M\) w.r.t. \(s\)

until \(q\) solves the problem

output either \(\phi(q) = \text{coll}(P,Q)\) or "no collision"

end

Figure 2.2 - Algorithm structure.
```

From the complexity viewpoint, we can reasonably estimate that the average number of minimization steps grows as the logarithm of the total number of cells \(C\), i.e., \(k = O(\log C)\), since the number of cells which overlap with the region \(M\) tends to be proportional to the area of \(M\) and about half of the area of \(M\) is discarded at each step. In [27] we argued that \(C = O(n^2)\) for polyhedra with \(O(n)\) vertices, and then \(k = O(\log n)\) from which it follows that algorithm **PhPhColl** runs in \(O(\log^2 n)\) time in the average.

### 3. Exploiting Spatial Coherence

In a variety of applications, including on-line motion planning, a proximity measure (or property) is recomputed after small intervals of time. Therefore, also the movements of the objects between two subsequent time steps should not be too large and we can expect that the relationships between their configurations do not change much. This observation applies, in particular, to the location of the closest points, as well as to the location of the contact points relative to subsequent collision tests. In similar situations we can gain a considerable speed-up by processing the information relative to the last computation of the same proximity measure, i.e., by exploiting the spatial coherence. Algorithms designed to this purpose are referred to as **incremental** algorithms in the literature.

#### 3.1. Incremental minimization

The nature of our approach allows us to endow the algorithm with a flexible mechanism to exploit spatial coherence at negligible additional costs. Roughly speaking, this mechanism rests on a very simple idea: during the minimization process, we can try to focus the search for the point of minimum in a suitable neighborhood of a previous solution. In order to implement this idea, we have to address two problems: (i) how to choose a suitable neighborhood and (ii) how to recover if the solution lies outside of it.

**Choice of the neighborhood**

Suppose that we are measuring collision translations relative to the bodies \(P, Q\) and the direction \(d\), while the configurations of \(P\) and \(Q\), and possibly the orientation of \(d\), are slightly changing in any arbitrary way from one computation to the next. (Notice that we are not assuming that \(P\) is actually moving in direction \(d\), although it might be, since \(d\) is only a parameter of the tests.) Moreover, to measure in some way the relative motion of \(P\) and \(Q\), which is all that matters here, look in direction \(d\) and suppose that you see the points initially overlapping to move less than a distance \(\delta\) apart with respect to each other.

Under these assumptions, also the contact points are likely to move less than \(\delta\) on the surfaces of \(P\) and \(Q\), and similarly would move the corresponding planar sections (although we can conceive situations where the contact points jump rather far away). This heuristic reasoning suggests that a suitable neighborhood should contain all the points closer than \(\delta\) to the last computed solution. Our specific choice is an isotropic square \(N_p(p)\) of size \(2\delta\), centered at the "previous" solution \(p\); this is equivalent to measuring the distances of the points representing pairs of sections by taking the \(|||\cdot|||_\infty\) norm. Incidentally, for a translational displacement of \(P\) the suitable \(\delta\) is simply the component of the translation vector perpendicular to \(d\); for more complex movements we can use an approximation of \(\delta\) based on a "maximal radius" of each polyhedron.

**Focused search vs. standard search**

It is clear, however, that we cannot guarantee that the new solution actually falls in the chosen neighborhood. So, we have to provide a mechanism for switching to the standard (non incremental) strategy whenever necessary. In that case, it is also desirable to save the work already done. Here is a possible technique to address both problems. The search starts at the point \(p\) representing the previous solution, but the minimization region \(M\) is initialized as usual (Figure 2.2). At each step, if the centroid \(c\) of the minimization region does not fall inside the neighborhood \(N^p(p)\) centered at \(p\), we consider the intersection point \(c'\) between the boundary of \(N^p(p)\) and the straight line segment \(pc\), as illustrated in figure 3.1. If \(c^o\) lies in \(M\), like \(c^o\) in the figure, then the next cut line is computed at \(c^o\) otherwise, as for \(c^o\), the next cut point is the centroid \(c\) and we proceed in the standard way. It is worth observing that the search focus can be tuned in a continuous range by means of the parameter \(\delta\), depending on the actual change of the relative configuration of \(P, Q\) and \(d\). As a consequence, the closer two consecutive configurations are, the faster the proximity measure can be updated (in the average).
3.2. Incremental algorithm

The incremental version of algorithm PhPhColl is outlined in figure 3.2, which should be self explanatory (see also fig. 2.2).

Incremental Version of PhPhColl
input: two convex polyhedra \( P, Q \), and a direction \( d \); a previous solution \( p \) and a neighborhood size \( \delta \)
begin
\( M := \) rectangle of all pairs of planar sections of \( P, Q \);
\( c := p \);
repeat
1. \( s := \) cut line through \( c \);
2. cell-shift \( s \) to point \( q \);
3. update \( M \) w.r.t. \( s \);
4. \( c := \) centroid of \( M \);
5. possibly update \( c \) w.r.t. \( N^\delta(p) \)
until \( q \) solves the problem;
output either \( \varphi(q) = \text{coll}_d(P, Q) \) or “no collision”
end

A rough estimate of the computational costs can be obtained as follows. Call \( \theta \) the ratio between the length \( \delta \), measuring the configuration change, and the height of the polyhedra (for simplicity, suppose that the polyhedra are about the same size). For the expected number \( C^\theta \) of cells intersecting the neighborhood \( N^\theta(p) \) over the total number \( C \) of cells we have
\[
\frac{C^\theta}{C} \approx \frac{\text{Area}(N^\theta(p))}{\text{Area}(M)} = 4\theta^2
\]
and for the number \( k \) of iteration steps
\[
k \approx \log C^\theta = \log C - 2 \log(1/\theta)
\]
So, if the updated solution lies inside \( N^\theta(p) \), the gain with respect to the standard strategy is of about \( O(\log(1/\theta)) \) iterations in the average. Since this rough estimate appears to be in good accordance with the experimental trends shown in figure 4.6, we can also see that the updated solution falls inside \( N^\theta(p) \) with high probability. This witnesses the important role which may play space coherence with the approach outlined in this paper.

4. Experimental Results

The algorithm outlined in the previous sections has been implemented in Pascal and tested on a platform based on the processor UltraSparc / 300 MHz. In what follows we summarize a detailed analysis of the results of about 20,000 collision detections planned in order to test the properties of the algorithm, more specifically:

- Trend of the measured time costs relative to collision detected from scratch.
- Cost reduction when the tested polyhedra do not collide (computations from scratch).
- Incremental behavior contrasted with standard behavior.
- Trend of the measured costs relative to the extent of the configuration change.

4.1. Computation of collision translations from scratch

A first set of experiments is aimed at testing the trend of the computational costs while increasing the complexity of the representation of the polyhedra, measured by their number of faces. The test polyhedra look like those depicted in figure 2.1. with all the vertices on the intersection between the surface of an ellipsoid and a set of equally spaced parallel planes. The numbers of faces of each polyhedron vary from \( n = 200 \) to \( n = 204,800 \). The configurations of the polyhedra are generated in a pseudo-random way. As we can see in figure 4.1, the average number of minimization steps grows as the logarithm of the number of faces; moreover, the number of steps keeps small also in complex cases (less than 15 steps in the average for two polyhedra of about 200,000 faces each). The average computation times reported in figure 4.2 are compared with the theoretical polylog trend, i.e., \( \log^2 n \). Notice that all the averages reported in figures 4.1 and 4.2 are relative to sets of independent and successful collision tests. In other words, each single test is carried out without exploiting the spatial coherence and eventually reports a collision configuration.

It may be interesting to notice that the algorithm behaves reasonably well also for simple bodies, [27]. For instance, successful collisions detections for tetrahedrons and cubes require about 0.2 msec in the average, which is less than the time required by a brute-force algorithm based on collision detection of all pairs of faces.

4.2. Detection that the bodies do not collide

The previous section has considered only successful collision tests. In figure 4.3, on the other hand, we summarize the performances of the algorithm when it detects from scratch that the polyhedra do not collide, since it turns out that the computation times are considerably reduced in that case. We
spend just a few words to explain the plots in that figure. The average times on the leftmost vertical line are exactly the same as in figure 4.2 and are reported for the sake of comparison. The other piles of data refer to unsuccessful collision tests for different separations of the two bodies when they get closest to each other. The order of such separation is measured by the distance between the region swept by the translating body (unbounded translation) and the other polyhedron, relative to an approximate radius \( R \) of the bodies, more specifically: about \( R/100 \) (the bodies get very close to each other), about \( R/10 \), about \( R \) and about \( 3R \) (when we can say that the bodies move very far from each other).

The results reported in figure 4.3 show that the algorithm runs faster to detect that there is no collision. The computation times are more than halved already for bodies getting very close and lowers down to about \( 1/4 \) of the reference time (successful collision detection) if they move far apart.

### 4.3. Incremental computation of collision translations

It is interesting to compare the incremental behavior of the algorithm, exploiting spatial coherence, with the corresponding behavior when each collision translations is computed from scratch. The two plots in figure 4.4 (“from scratch” above, “incremental” below) are relative to the same sequence of configurations, which results from a continuous translation of one of the two polyhedra in a different direction than that of the (successful) collision tests. Both polyhedra have about 12,800 faces and the results of one hundred collision tests are reported. Both polyhedra have about 12,800 faces and the results of one hundred collision tests are sampled at regular intervals, each having a length to about 2\% of the approximate radius \( R \) of the polyhedra. A similar comparison is done in figure 4.5, where the length of the elementary motions are shorter. Again, this provides strong evidence for the usefulness of exploiting spatial coherence.

A comparison of the figures 4.4 and 4.5 gives an intuitive idea of what we mean by self-tuning mechanism. Roughly speaking, the degree of gain in performances is related in a

![Figure 4.3 - Collision vs. no collision](image)

![Figure 4.4 - Standard strategy vs. incremental strategy](image)

![Figure 4.5 - Comparison for a sequence of closer configurations](image)

![Figure 4.6 - Standard strategy vs. incremental strategy: trends](image)

![Figure 4.7 - Standard strategy vs. incremental strategy: trends](image)

---

**4.4. Self-tuning mechanism**

The incremental behavior for different polyhedron complexities and for different sizes of the elementary configuration change, i.e., of the parameter \( \delta \), are summarized in figures 4.6 and 4.7. The meaning of \( R \) is the same as that introduced in section 4.2; this time the fractions of \( R \) are used to measure \( \delta \). As we can see, the average numbers of steps reported in figure 4.6 follow a logarithmically decreasing trend which is in accordance with the trend estimated in section 3.2. (In the cases of a smaller numbers of faces, notice that with our implementation we cannot solve the problem in less than one minimization step.)
reading the results of the experiments; we can try to relate the average number of steps to the size δ expressed as a number or fraction ν of the average size of the polyhedron faces. The idea behind this is that the size of the cells of the mark is likely to be related to the size of the polyhedron faces. As we may expect, in the case of high spatial coherence the average number of minimization steps does not seem to depend much on the polyhedron complexity, but there is a strong correlation with the parameter ν. For instance, for ν = 1 the average number of steps fluctuates between 3 and 4.5; for ν = 1/4 it is confined between 1 and 2.

5. Conclusions

The main focus of this paper has been on the experimental analysis relative to an algorithm for computing collision translations of convex polyhedra. More specifically, we have compared the performances of the algorithm in a variety of situations and we have also been able to appreciate the benefits which can be attained by exploiting spatial coherence. The results of the experiments seem to be encouraging and reveal a few interesting features of the approach: fast computation from scratch (without initialization), faster detection that there are no collisions, flexible exploitation of space coherence in a continuous range. These properties can be especially appreciated in the field of robot path planning, where we have found the original motivations of our work, [28].

It is worth mentioning that we have also investigated on the distribution of computation times relative to the different kinds of geometric operations carried out by the algorithm. We have seen that about half of the time is consumed for computations on matrices and vectors in two or three dimensions which could be speeded up by specialized hardware; on the other hand, only a negligible fraction of the time is spent for the control of the minimization process. Further benefits may be gained by improving the technique to focus the search for the point of minimum in the vicinity of a previous solution. However, our analysis has not been limited to the permanent viewpoint, but has also addressed other important issues and possible shortcomings of the approach discussed in this paper, some of which are above all, those concerning the robustness of the approach — are briefly discussed in [27].

References