Université catholique de Louvain
Louvain School of Engineering
Institute of Information and Communication Technologies, Electronics and Applied Mathematics

Center for Systems Engineering and Applied Mechanics

Heuristic optimization methods for three matrix problems

Chia-Tche Chang

Thesis submitted in partial fulfillment of the requirements for the degree of *Docteur en Sciences de l'Ingénieur*

Dissertation committee

Prof. Vincent Blondel	Université catholique de Louvain
Prof. Paul Van Dooren	Université catholique de Louvain
Prof. François Glineur	Université catholique de Louvain
Prof. Raphaël Jungers	Université catholique de Louvain
Prof. Moritz Diehl	Katholieke Universiteit Leuven
Prof. Ilse Ipsen	North Carolina State University, USA

Louvain-la-Neuve, December 2012

Acknowledgements

First of all, I would like to thank my advisor, Vincent Blondel, for accepting me as a Ph.D. student and guiding me during these four years of research. I am grateful for his valuable support and working under his supervision was a real pleasure.

I am very thankful to Paul Van Dooren for accepting to participate in my supervising committee, but also for his insightful comments about my work. I also would like to thank the other members of my Jury: François Glineur, Raphaël Jungers, Moritz Diehl and Ilse Ipsen. Their remarks and questions were very valuable for the final writing of this thesis.

Of course, research also allows one to interact with many other people. In particular, I would like to thank Quentin Rentmeesters, my officemate at CESAME for more than four years, but also Samuel Melchior, Bastien Gorissen, Romain Hollanders, Mathieu Renauld, Nicolas Boumal and _____[1] for the numerous discussions about research — and non-research — topics.

Research is not limited by international borders and I gratefully acknowledge Denise Maurice, Ismaël Bouya, Jill-Jênn Vie and Olivier Blazy for all our interactions. Special thanks go to Randall M. and Jorge C. for their amazing work that helps graduate students (and others...) cope with this unique experience.

Finally, I would like to thank my friends and my parents for their continuous support.

[1] Fill in your name if you think you should be here!

This thesis has been supported by a F.R.S–FNRS fellowship. It also benefited from the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, and the "Action de Recherche Concertée", funded by the French Community of Belgium.

Contents

List of Figures

List of Tables

List of Algorithms

List of Publications

Journal articles

[CGM11] Chia-Tche Chang, Bastien Gorissen, and Samuel Melchior. Fast oriented bounding box optimization on the rotation group $SO(3, \mathbb{R})$. *ACM Transactions on Graphics*, 30(5):122:1–122:16, Oct. 2011.

[CB12a] Chia-Tche Chang and Vincent D. Blondel. An experimental study of approximation algorithms for the joint spectral radius. *Numerical Algorithms*, Accepted for publication, 2012.

[CB12b] Chia-Tche Chang and Vincent D. Blondel. A genetic based algorithm for fast approximations of the joint spectral radius. Submitted to *Systems & Control Letters*, 2012.

[CIBD12] Chia-Tche Chang, Ilse Ipsen, Vincent D. Blondel, and Paul Van Dooren. Polynomial-time subset selection via updating. In preparation, 2012.

Conference papers

[CB11a] Chia-Tche Chang and Vincent D. Blondel. Approximating the joint spectral radius using a genetic algorithm framework. In *Proceedings of the 18th IFAC World Congress (IFAC WC2011)*, pages 8681–8686, Milano, Italy, Aug. 2011.

Conference abstracts

[CJB09] Chia-Tche Chang, Raphaël M. Jungers, and Vincent D. Blondel. On the growth rate of matrices with row uncertainties. In *Book of Abstracts of the 14th Belgian-French-German Conference on Optimization (BFG09)*, page 86, Leuven, Belgium, Sept. 2009.

[CB10] Chia-Tche Chang and Vincent D. Blondel. A comparison of approximation algorithms for the joint spectral radius. In *Book of Abstracts of the 29th Benelux Meeting on Systems and Control (BMSC10)*, page 85, Heeze, The Netherlands, Mar. 2010.

[CB11b] Chia-Tche Chang and Vincent D. Blondel. A genetic algorithm approach for the approximation of the joint spectral radius. In *Book of Abstracts of the 30th Benelux Meeting on Systems and Control (BMSC11)*, page 105, Lommel, Belgium, Mar. 2011.

The journal article [CGM11] has also been presented at the 5^{th} ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia (SIGGRAPH Asia 2012) in Singapore, Nov. 2012.

CHAPTER 1
Introduction

Optimization is present in a wide variety of situations: many decision problems can be summarized in a single sentence: "*What is the best choice?*" Of course, the meaning of "best" is highly dependent on the situation. There may be several possible "best choices" or sometimes no "best" solution to the problem. These solutions may be easy to find, or very hard to obtain, even by using computers or supercomputers.

Let us consider several problems.

- *Isoperimetric problem:* given a rope of fixed length, we want to enclose a planar region whose area is as large as possible. What shape should we use?
- *Shortest path problem:* given a map of a country containing our current location and a destination city, what path should we take in order to reach the destination as fast as possible?
- *Bin packing problem:* given a bunch of identical containers of fixed capacity and a list of objects with given sizes, how many containers do we need to pack all objects?

The solution of the isoperimetric problem was already known in Ancient Greece: if L denotes the length of the rope, the area A of any enclosed area satisfies $A \leqslant \frac{L^2}{4\pi}$, with equality if and only if the rope forms a circle. Independently of the length of the rope, the best choice is to enclose a disk.

The shortest path problem is more difficult: the best solution may not be obvious. However, the problem may be solved quickly enough using a procedure conceived by Dijkstra in 1956 and published in [Dij59]. The idea is to examine all cities, starting from the closest ones and slowly moving to further ones, while establishing the fastest way to reach all

these cities and reusing this information. The cities are thus explored in increasing order of distance from the starting point. Even though this may take a while to compute, each city is only analyzed at most once, and there is no backtracking to be done.

The bin packing problem is significantly harder in the general case. One may have to try a large number of combinations before finding a partition with the smallest number of bins *and knowing that it is indeed impossible to do better*. More precisely in computational complexity theory, this problem is said to be NP-hard [GJ79].

The class of NP problems is defined to be the set of decision problems for which if the answer is positive, then there are certificates that can be verified in a time that is polynomial in the size of the input. For example, an instance of the *Hamiltonian path problem* asks whether it is possible to visit each vertex of a given graph *exactly once*, knowing the list of edges. This problem is indeed in NP since if the answer to the question is "yes", then this can be easily verified if we are given such a path.

The class of NP-hard problems corresponds to problems that are "at least as hard" as any problem in NP. In this context, "at least as hard" means that any problem in NP can be reduced to the NP-hard problem in polynomial time. In other words, an algorithm for the NP-hard problem can be used to produce algorithms to solve any problem in NP.

This implies that there is no deterministic polynomial algorithm known to solve the bin packing problem, and it is suspected that such algorithm does not exist. With such NP-hard problems, it is sometimes the case that we are already satisfied with a solution suspected to be optimal but without having a proof of optimality, or even a *good enough* solution instead of an optimal one.

In practice, one may want to obtain a solution in a short amount of time. When several items have to be successively examined, the solution may be reached faster if the items are ordered following some rule. Let us consider the shortest path problem. If we know that the destination is located north of our position, it may be wise to look at the northern part first. This corresponds to the A* algorithm [HNR68]. In this case, the A* algorithm reaches the solution faster than Dijkstra's algorithm due to a *heuristic*, viz. the rule biasing the search towards the location of the destination.

Of course, such heuristics do not always work: if we consider a shortest path problem in a maze, aiming towards the location of the exit may not necessarily be a good idea as mazes are designed so that this simple heuristic leads to dead ends. In this case, the A* algorithm will not necessarily be able to find the shortest path faster than Dijkstra's algorithm, even though it will still reach the solution (provided that several properties are satisfied [DP85]).

Similarly, a possible heuristic for the bin packing problem is to order the objects by decreasing sizes, and then put them successively into the first bin that can contain the object [Dós07]. This may clearly lead to a suboptimal solution. For example, if we have two bins with remaining capacities 6 and 5, and objects with sizes 4, 3, 3 and 1, putting the item with size 4 in the bin with capacity 6 will lead to the creation of a supplementary bin, even though it was possible to pack the four objects in the two original bins. Of course, selecting the smallest bin that can contain the current object instead of the first one is also suboptimal: this strategy fails if we have two bins with capacities 7 and 5, and objects with sizes 4, 3, 3 and 2.

When considering difficult problems, or even easier problems but in situations where the computation time has to be as low as possible, one often uses heuristics, i.e., strategies that are expected to speed up the procedure in *most* cases. However, this usually comes with a drawback: the solution obtained by a heuristic approach may be suboptimal. Even though in some cases, it is possible to prove some guarantees on the solution returned by the heuristic method, heuristic approaches do not guarantee that an optimal solution will be found in general.

In practical applications, this may not necessarily be a problem: being able to quickly find a good suboptimal solution is usually better than having a very expensive algorithm able to find the optimal solution. Many difficult or NP-hard problems are tackled in practice using heuristic methods due to time constraints, e.g., scheduling problems, covering problems, or network design problems.

In this thesis, we are studying general heuristic methods to tackle hard optimization problems. These frameworks, also known as *metaheuristics*, are designed to be applicable to a wide range of problems as they are using general rules that are independent of the problem to solve. At the same time, there is usually a lot of freedom when implementing these general methods, and it is often possible to obtain very good re-

sults if these choices are carefully done, for example depending on the type of the problem.

In our case, we will be using combinations of several heuristics to help the convergence to a good — and hopefully optimal — solution. The main goal is to design methods with a very low computational cost, although there will be no a priori guarantee on the quality of the solution found by the proposed methods. Indeed, for the problems we consider, classical methods able to find the optimum solution are often much too slow to be of practical use. A method with a low computational cost has thus obvious advantages if the computation resources are limited or if the application is time-critical. Furthermore, this should allow us to handle problems with larger sizes, where classical methods are too expensive.

Outline

This thesis is organized as follows. We begin by presenting several well-known methods for general optimization problems in Chapter 2. Starting with classical approaches, we move into different general heuristic methods such as simulated annealing [KGV83] or genetic algorithms [Hol75]. This chapter is not intended to list all existing approaches, but rather to present different tools required for the construction of our own algorithms.

In Chapter 3, we look at the *minimum-volume arbitrarily oriented bounding box* problem, which has applications mainly in computer graphics and related fields. The problem consists in finding the smallest rectangular parallelepiped enclosing a given set of points. Even though an exact algorithm solving the problem in polynomial time exists since 1985 (O'Rourke's algorithm, see [O'R85]), its performance is nowhere near the practical requirements: the computation of the optimal bounding box may take several hours, days or even weeks for problems with more than 10000 vertices on its convex hull. In practice, heuristic approaches are preferred, e.g., techniques based on principal component analysis [Jol02]. These methods are very fast but usually return suboptimal solutions. We propose a new algorithm, HYBBRID, that has been able to find the optimal bounding box in most cases, very good solutions being returned when optimality is not reached. These results correspond to a benchmark containing about 300 test cases taken from [Gro08]. Hence, HYBBRID offers an alternative to methods that are accurate but very slow, and very fast algorithms with poor accuracy. The exact trade-off

between accuracy and computation time can also be set depending on the considered applications.

Next, we consider another application of heuristics in Chapter 4: the *column subset selection* problem. Here, we are interested in selecting a fixed number of columns from a given matrix so that the volume of the parallelepiped spanned by these columns is as large as possible. This can be interpreted as selecting columns that are far from being redundant and has applications in fields such as data mining. This problem has been proved to be NP-hard [cMI09]. We present several algorithms for the subset selection problem, with different accuracy/computation time trade-offs. Numerical experiments show that when compared to a reference algorithm (Gu and Eisenstat's strong rank revealing QR algorithm, see [GE96]), the methods we propose are able to find a subset of columns with similar quality in a shorter amount of time. In several cases, our set of columns has even a significantly larger volume than Gu and Eisenstat's solution.

Last but not least, we present our contributions to the study of the *joint spectral radius* in Chapter 5. This quantity characterizes the growth rate of products of matrices, generalizing the notion of spectral radius of a single matrix to sets of matrices [RS60]. It is known since 1997 that the problem of approximating the joint spectral radius is NP-hard [TB97]. Despite this negative theoretical result, many approaches have been proposed for the approximation of the joint spectral radius. We present a brief survey of these algorithms, which we have implemented in MATLAB® as part of the JSR Toolbox [VHJ+11]. This is not without importance as many algorithms have never been studied from a practical point of view, and it is clear that even though most algorithms have a way to converge to the exact value of the joint spectral radius (in exponential time), their practical behaviors may present many differences. In addition to this comparison of methods based on practical performances, we also propose a new algorithm based on a heuristic approach, viz. the genetic algorithm. Again, similarly to the previous chapters, our algorithm has been able to find very good bounds on the exact solution in a short amount of time, these bounds matching the value of the joint spectral radius in many cases. It is also able to tackle larger sets of matrices, while most classical methods become too expensive to use.

We present our final comments and conclude this thesis in Chapter 6.

CHAPTER 2

From classical optimization to heuristic approaches

In this chapter, we present several well-known techniques and heuristic methods for function optimization (without constraints). Of course, this is not intended as an exhaustive compilation of algorithms, given the huge number of different approaches. We will rather focus on the main methods, and in particular those related to our contributions in the next chapters.

2.1 Classical optimization methods as starting point

Let f be a function so that our optimization problem can be expressed as

$$\max_{x \in \Omega} f(x),$$

where $\Omega \subset \mathbb{R}^n$ is the domain of the optimization variable x. This domain may be continuous or discrete. The function f will be referred to as the *objective function*, the *utility function*, or the *fitness function* depending on the context. When the problem is written as a minimization problem, we also use the term *cost function*. Here, the objective is to find the global maximum of the function, $f(x_{\max})$, and/or an associated solution $x_{\max}$ without using a brute-force method testing all solutions $x \in \Omega$, or sampling the search space and taking the best result. However, obtaining the global maximum is very difficult to guarantee. In most cases, it is only reasonable to try to find a maximum said to be *local*. In the continuous case, this means that the value of objective function becomes worse when the solution $x_{\max}$ is slightly perturbed. In the discrete case,

local optimality corresponds to the fact that all other points in a neighborhood of $x_{\max}$ are not better than $x_{\max}$ with respect to the objective function. Thus, this property depends on the chosen neighborhood.

2.1.1 Hill climbing and steepest ascent methods

Given a starting point $x_0 \in \Omega$, one wants to build a sequence of solutions $(x_i)_{i\in\mathbb{N}}$, hopefully converging to $x_{\max}$. One of the most basic methods is the *hill climbing* technique, also known as *gradient method* for continuous spaces [NW06], or *steepest ascent — steepest descent* for minimization problems. The idea is very simple: among all points $y_{k,i}$ that are "close" to the current point x_k, the next point x_{k+1} is taken as the $y_{k,i}$ maximizing the objective function. This is thus a *greedy* method as it is supposed that taking the best solution at each step will result in a best global solution. Of course, this hypothesis does not hold in general, and the method usually finds a local maximum which is not global. In fact, this method is quite sensitive to the starting point. One may want to restart the algorithm from different initial points to try to find a better local maximum. This is known as the *restart strategy*, which can be used in any algorithm requiring to choose a starting point.

Algorithm 2.1 Generic gradient method

Input: f function to maximize
tolerance $\tau > 0$
initial solution x_0
Output: solution x expected to maximize $f(x)$

1: $x_{\text{cur}} := x_0$
2: **repeat**
3: $\quad d := \nabla f(x_{\text{cur}})$
4: $\quad$ Determine a step size α associated to the point x_{cur} and the direction d
5: $\quad x_{\text{prev}} := x_{\text{cur}}$
6: $\quad x_{\text{cur}} := x_{\text{cur}} + \alpha d$
7: **until** $\|x_{\text{cur}} - x_{\text{prev}}\| < \tau$
8: $x := x_{\text{cur}}$

For continuous spaces, the best direction to go from the current point x_k can be obtained by computing the gradient $\nabla f(x_k)$, hence the name of the method. Algorithm 2.1 shows the steepest ascent method in the

continuous case. Note that this gradient may be difficult — or impossible — to obtain depending on the objective function. In this case, one possibility would be to produce an approximation or an empirical gradient. Given the value of the gradient, the point x_{k+1} is then taken as:

$$x_{k+1} = x_k + \alpha_k \nabla f(x_k),$$

where the size α_k of the step can for example be fixed depending on k or taken such that the increase of the objective function is maximized, either exactly or approximately. This approach is referred to as a *line-search method*. In order to avoid the optimization of this one-dimensional subproblem in the direction $d_k = \nabla f(x_k)$, one can also use the Wolfe conditions [Wol69], which correspond to the two conditions

$$f(x_k + \alpha_k d_k) \geqslant f(x_k) + c_1 \alpha_k d_k^T \nabla f(x_k), \tag{2.1}$$

and

$$d^T \nabla f(x_k + \alpha_k d_k) \leqslant c_2 d_k^T \nabla f(x_k). \tag{2.2}$$

The condition (2.1), which is called the Armijo condition, ensures that the value of the objective function decreases "sufficiently", and the condition (2.2), known as the curvature condition, ensures that the slope is "sufficiently" reduced. In particular, this condition rules out extremely small step sizes α_k that satisfy the Armijo condition. The two constants satisfy $0 < c_1 < c_2 < 1$. The constant c_1 is usually chosen to be close to 0, while c_2 is chosen to be close to 1 [NW06]. Other line-search conditions such as the strong Wolfe conditions or the Goldstein conditions are also described in [NW06].

Several variants are possible: instead of using a greedy strategy, *stochastic hill climbing* methods (Algorithm 2.2) are sometimes used (see [FM93] for an example in the discrete case). Rather than testing all neighbors or directions and taking the best one, the stochastic version chooses one random neighbor or direction and uses it to obtain x_{k+1}, provided that the value of the objective function improves when moving to the new candidate. When combined with the restart strategy, this variant may result in a better exploration of the search space.

Nevertheless, basic hill climbing algorithms are not expected to perform well in general problems with several local maxima: the starting point has to be sufficiently close to a global maximum in order to be able to reach it, and the method may require a large number of iterations before converging to the corresponding solution.

Algorithm 2.2 Generic stochastic gradient method

Input: f function to maximize
tolerance $\tau > 0$
initial solution x_0
Output: solution x expected to maximize $f(x)$

```
1: x_cur := x_0
2: repeat
3:    if ∇f(x_cur) = 0 then
4:       x := x_cur
5:       exit
6:    end if
7:    d := random direction such that d^T ∇f(x_cur) > 0
8:    Determine a step size α associated to the point x_cur and the direction d
9:    x_prev := x_cur
10:   x_cur := x_cur + αd
11: until ||x_cur − x_prev|| < τ
12: x := x_cur
```

Let us first look at some well-known alternative methods in the continuous case. For discrete problems, local search-based methods are more suitable. They are presented in Section 2.2.

2.1.2 Optimization methods in the continuous case

For differentiable functions, the steepest ascent method uses the gradient information at the current solution in order to determine the next candidate. Higher-order differentials can be used if they are available. For example, *Newton's method* uses the gradient ∇ and the Hessian ∇^2, with the iteration

$$x_{k+1} = x_k - \alpha_k \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k).$$

Newton's method and its variants appear in many applications, including interior-point methods used to solve convex optimization problems [NN94]. When applicable, this method converges much faster than gradient methods. The main issue is that the computation of the Hessian is required for the application of Newton's method. Of course, one

can use approximations of the Hessian in order to reduce the computational cost; this gives us the class of *Quasi-Newton methods*. Common variants are the SR1 (symmetric rank one) formula [MS70], or the BFGS (Broyden-Fletcher-Goldfarb-Shanno) method [Bro70, Fle70, Gol70, Sha70].

Other methods such as the *conjugate gradient method* take into account the previous update direction in order to compute the current direction:

$$x_{k+1} = x_k + \alpha_k d_k, \qquad d_k = \nabla f(x_k) + \beta_k d_{k-1},$$

for some value of β_k. Several popular methods used to determine the value of β_k are the Fletcher-Reeves [FR64] and the Polak-Ribière [PR69] formulas.

It is clear that there are a bunch of possibilities when one has to choose an optimization method. However, both gradient-based methods and Newton-based methods share a major common feature: the value of the gradient at the current point is required. What if the gradient does not exist, is unavailable or is too expensive to compute?

There are several approaches for the optimization of functions without using their derivatives. Classical derivative-free optimization (DFO) methods have the same goal as the techniques presented above: starting from an initial point x_0, try to guarantee convergence to a local optimum. Several classes of DFOs can be distinguished.

Trust-region methods consist in approximating the objective function with a simpler model function in a neighborhood of the current point x_k. This approach can be used for both differentiable and non-differentiable objective functions. At each iteration, the optimization is done on the model function, but only within the neighborhood or *trust region*. The result is then compared to the true function. If the behavior of the original function matches the expected improvement based on the model function, then the size of the trust region is increased. The trust region is reduced if the model does not reflect well the behavior of the objective function. A comprehensive reference on trust-region methods can be found in [CGT00].

Another main approach is the class of direct search algorithms (see [KLT03] for a survey). The spirit of the approach is to examine several "trial" solutions and use some strategy to determine what the next candidate will be, based on the results obtained so far. Line-search methods

(see [Kel99]) are also eligible in a derivative-free framework as one does not need to follow the steepest ascent direction to improve the objective function.

In the next section, we will concentrate on the *Nelder-Mead simplex algorithm*, one of the best known methods of optimization by direct search which has led to many variants. A more detailed review on DFO methods can be found in [CSV09].

2.1.3 Nelder-Mead simplex algorithm

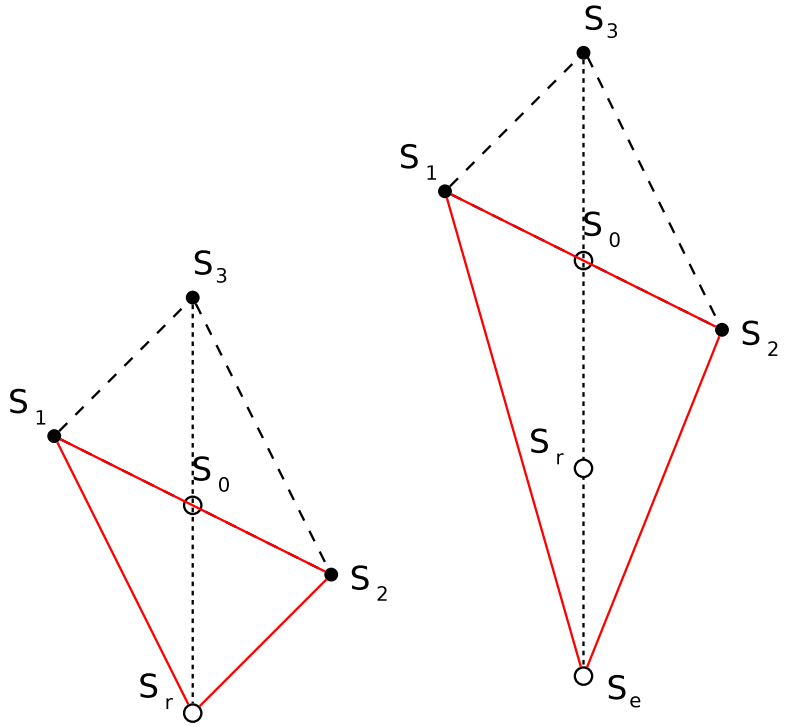
The *Nelder-Mead algorithm* was first proposed by John Nelder and Roger Mead in 1965 [NM65]. The main idea is that the method always keeps a simplex, i.e., a polytope of $d + 1$ vertices for a d-dimensional space, instead of a single candidate. This simplex is then moved and deformed depending on the values of the objective function at its vertices.

More precisely, the algorithm starts with an initial simplex composed of $d + 1$ points. At each iteration, the worst point is replaced by its reflection through the centroid of the d remaining points, if this reflected point is good enough with respect to the fitness function. In particular, if the reflected point is even better than the best current point, a good search direction may have been found, so the algorithm tries to explore further in that direction.

If the new point is not good enough, then we have two points located at opposite directions from the centroid that are both considered "bad". In this case, an optimal point may lie inside the simplex, so we shrink it. The detailed steps of an iteration of the algorithm are shown below (see also Figure 2.1), and a pseudocode version of the method is given in Algorithm 2.3.

Initialization

Let $\mathcal{S} \subset \mathbb{R}^d$ be a simplex with vertices $S_1, S_2, \ldots, S_{d+1}$. Without loss of generality, we can assume that $f(S_1) \geqslant f(S_2) \geqslant \ldots \geqslant f(S_{d+1})$, where f is the function to maximize. The worst point is thus S_{d+1} here. Let the centroid of $\{S_1, S_2, \ldots, S_d\}$ be denoted by S_0.



(a) Reflection step (b) Expansion step

(c) Contraction step (d) Reduction step

Figure 2.1: *Illustration of the Nelder-Mead algorithm in* $\mathbb{R}^2$. *The simplex* $\mathcal{S} \subset \mathbb{R}^2$ *has vertices* S_1, S_2, S_3, *with* S_3 *being the worst point with respect to the objective function.*

Reflection step

Let S_r be the reflection of S_{d+1} through the centroid S_0, that is,

$$S_r = S_0 + (S_0 - S_{d+1}) = 2S_0 - S_{d+1}.$$

If $f(S_r) > f(S_d)$, then the new simplex is defined by replacing S_{d+1} with S_r.

Expansion step

If the reflected point is the current best candidate, i.e., $f(S_r) > f(S_1)$, then an expansion step is attempted. The expanded point S_e is defined as

$$S_e = S_r + (S_r - S_0) = 2S_r - S_0 = 3S_0 - 2S_{d+1}.$$

If the expansion is successful, i.e., $f(S_e) > f(S_r)$, then instead of replacing S_{d+1} with S_r when constructing the new simplex, we replace it with S_e.

Contraction step

If the reflection was unsuccessful ($f(S_r) \leqslant f(S_d)$), we compute the contracted point S_c using the expression

$$S_c = S_0 + \frac{1}{2}(S_{d+1} - S_0) = \frac{1}{2}(S_0 + S_{d+1}).$$

If $f(S_c) > f(S_{d+1})$, the fitness is improving when moving inwardly of the simplex from the worst vertex S_{d+1}, so the new simplex is taken as $\{S_1, S_2, \ldots, S_d, S_c\}$.

Reduction step

If both the reflection and the contraction steps fail, the whole simplex $\mathcal{S}$ is reduced around its best vertex S_1: a new simplex $\{S_1, S_{2'}, S_{3'}, \ldots, S_{d'}, S_{(d+1)'}\}$ is constructed using the transformation

$$S_{i'} = \frac{1}{2}(S_1 + S_i), \qquad i = 2, \ldots, d+1.$$

Comments

The Nelder-Mead algorithm is not guaranteed to converge to a stationary point (for example see [McK99]), and largely depends on the initial simplex. Several runs with different initial simplices can be performed, in order to increase the probability of reaching a global optimum. Another possibility is to use more elaborate variants of the algorithm with improved performances.

Algorithm 2.3 Nelder-mead simplex method

Input: f function to maximize
initial simplex $\mathcal{S} = (S_1, S_2, \ldots, S_{d+1})$ with vertices ordered such that $f(S_1) \geqslant \ldots \geqslant f(S_{d+1})$
Output: solution x expected to maximize $f(x)$

while stopping criterion is not met **do**
  $S_0 := \frac{1}{d}(S_1 + \cdots + S_d)$
  $S_r := 2S_0 - S_{d+1}$
  if $f(S_1) \geqslant f(r) > f(S_d)$ **then**
    $S_{d+1} := S_r$
  else if $f(r) > f(S_1)$ **then**
    $S_e := 3S_0 - 2S_{d+1}$
    $S_{d+1} := \text{argmax}_{s \in \{S_r, S_e\}} f(s)$
  else
    $S_c := \frac{1}{2}(S_0 + S_{d+1})$
    if $f(S_c) > f(S_{d+1})$ **then**
      $S_{d+1} := S_c$
    else
      $S_i := \frac{1}{2}(S_1 + S_i)$ for all $i = 2, \ldots, d+1$
    end if
  end if
  Reorder the vertices of $\mathcal{S}$ such that $f(S_1) \geqslant \ldots \geqslant f(S_{d+1})$
end while
$x := S_1$

The Nelder-Mead simplex method will be used as a component of the algorithm presented in Section 3.4.2. The role of the Nelder-Mead iterations will be to locally improve solutions associated to the optimization problem. The exploration of the search space will be left to another component, viz., a genetic algorithm (see Section 2.3.1).

2.2 Local search heuristic methods

The methods presented in the previous section are designed for an optimization problem with continuous variables. If the search space is discrete, other approaches are generally more suitable for local optimization. Indeed, it is not always possible to compute approximate gradients. Applying an iteration and taking the "closest" admissible point may not be a good idea in the general case. Depending on the problem, these approximations are sometimes simply meaningless.

Let us go back to the hill climbing strategy and let us consider a discrete search space. The general idea is to take a point in the neighborhood of the current solution, and move to it if the value of the fitness function increases. Here, a neighborhood corresponds to a set of solutions that are "close" to the current solution, in the sense that they can be obtained by applying small local changes to the current candidate. The exact nature of these changes is heavily dependent on the problem. Such local search techniques are widely used when facing hard combinatorial problems. In the following sections, we will present several well-known strategies for discrete optimization problems, starting with the simulated annealing and the tabu search.

In order to illustrate the different methods and concepts, let us use the classical *n-queens problem* as running example.

Example 2.1. *The n-queens problem is defined as follows: given an $n \times n$ chessboard, we want to place n queens in distinct squares such that no two queens attack each other. In other words, we have to place n elements in an $n \times n$ grid so that no two elements share the same row, column or diagonal. It is known that this problem admits solutions for all $n > 3$ (see [BS09] for a survey of known results on the subject). For instance, a possible solution for the usual 8×8 chessboard is shown in Figure 2.2.*

In order to manipulate candidate solutions in the different algorithms, we will use a simple encoding*, or representation of the solutions. The positions of the n queens will be encoded as a list of n integers $\mathcal{Q} = (q_1, \ldots, q_n) \in \{1, \ldots, n\}^n$, where $q_i = j$ means that there is a queen at column i, row j of the chessboard. The position in Figure 2.2 is thus encoded by the list $(4, 6, 8, 2, 7, 1, 3, 5)$ if the rows are numbered from the top to the bottom. As an additional example, the configuration in Figure 2.3 is encoded by the list $(1, 1, 2, 2, 3, 3, 4, 4)$.*

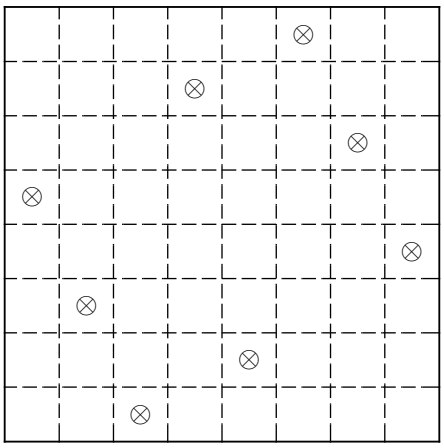


Figure 2.2: *One possible solution to the* 8*-queens problem.*

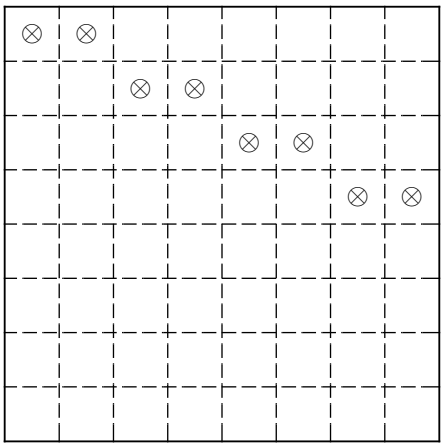


Figure 2.3: *The* 8*-queens configuration encoded by the list* $(1, 1, 2, 2, 3, 3, 4, 4)$.

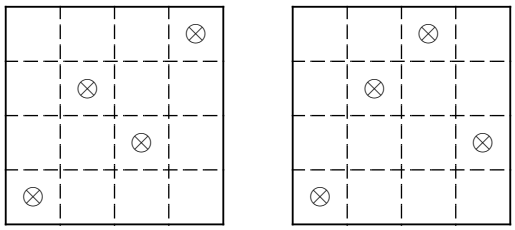


Figure 2.4: *Two neighbors for the "swap" neighborhood, in the* 4*-queens problem.*

Choosing this representation ensures that two queens will never share the same column. We also have to define a fitness function *that measures the quality of a solution. Here, a possible choice would be to use the number of pairs of queens that do* not *attack each other. With this fitness function, the goal is to find a candidate solution with a fitness of* $\binom{n}{2} = \frac{1}{2}n(n-1)$.

Finally, several neighborhoods may be defined for the n*-queens problem. For example, given a candidate solution* $\mathcal{Q} = (q_1, \ldots, q_n)$*, one can construct a neighborhood containing all* n*-tuples that differ from* $\mathcal{Q}$ *by exactly one component. This can be interpreted as the set of positions reachable by moving one queen in its column. This "move" neighborhood has thus a size of* $n(n-1)$*. Another possibility would be to consider a neighborhood formed by all positions obtainable from* $\mathcal{Q} = (q_1, \ldots, q_n)$ *by swapping two elements in the list. This "swap" neighborhood has a size of* $\frac{1}{2}n(n-1)$*. Note that if we use the second neighborhood, then starting with a permutation of* $(1, \ldots, n)$ *as initial position ensures that there will never be two queens in the same column, nor in the same row.*

As an illustration, the two positions shown in Figure 2.4 are neighbors with respect to the "swap" neighborhood. The fitness of the first position is 4 while the fitness of the second position is 5.

2.2.1 Simulated annealing

The main idea of *simulated annealing* is based on the following observation: sometimes, one has to accept an unfavorable move in order to reach a better global solution. The method was described by Kirkpatrick et al. in 1983 [KGV83] and is inspired by a physical process. Indeed, the

concept of *annealing* in metallurgy corresponds to a procedure consisting in cycles of adequate heating and cooling, so that the crystal structure of the object can be modified. A high temperature allows the atoms to move from their initial positions, which can be viewed as a local minimum for the internal energy, to configurations of higher energy. When the temperature decreases, the atoms stabilize in another configuration of low energy, which can be better than the initial structure.

The optimization method follows the same ideas: a parameter T (the "temperature") is set to a high value in the initial conditions and is slowly decreased during the process. At each iteration, a candidate that is *worse* than the current solution may be accepted, depending on the current temperature and the quality of the candidate. Hence, unfavorable moves are allowed at the beginning of the algorithm so that we do not remain stuck in a local optimum, but the process has to stabilize in the long-term.

The details of the algorithm are presented in Algorithm 2.4 for a minimization problem, in order to keep consistency with the physical process. An overview of the simulated annealing technique can be found in [vLA87].

Algorithm 2.4 Generic simulated annealing

Input: f function to minimize
initial temperature $T > 0$ and cooling schedule
initial solution x_0
Output: solution x expected to minimize $f(x)$

$x_{\text{cur}} := x_0$
$x := x_{\text{cur}}$
while stopping criterion is not met **do**
  $x_{\text{neigh}} :=$ random neighbor of x_{cur}
  if `random` $< \exp\left(\frac{f(x_{\text{cur}}) - f(x_{\text{neigh}})}{T}\right)$ **then**
    $x_{\text{cur}} := x_{\text{neigh}}$
  end if
  if $f(x_{\text{cur}}) < f(x)$ **then**
    $x := x_{\text{cur}}$
  end if
  decrease temperature T
end while

The initial solution x_0 may be randomly chosen in the search space.

The `random` expression corresponds to a random number between 0 and 1 using a uniform distribution. Note that if the neighbor is better than the current point, i.e., $f(x_{cur}) - f(x_{\text{neigh}}) > 0$, the update is accepted independently of the value of `random` since the right-hand side of the inequality would be larger than 1.

Several stopping criteria can be chosen, e.g., if the temperature T is too low, or if there was no improvement with respect to the cost function during a large number of iterations. The main parameter is the cooling schedule: how to decrease the temperature? This is mainly a design choice influenced by the problem. Common choices are a sequence of thresholds or a continuously decreasing temperature, for instance using an exponential decay. In any case, the temperature for the first iterations has to be sufficiently high in order to allow escaping from local minima. Of course, it is also possible to run this procedure several times, as with the physical process.

Example 2.2. *For the n-queens problem, a simulated annealing approach can be easily implemented. At each iteration, we choose a random candidate in the current neighborhood. For instance, let us use the "swap" neighborhood; this is interpreted as choosing a random pair of queens on the chessboard. These two queens are located on different rows and different columns. Let us look at the alternate position where the two queens are exchanging their row values, while keeping their column values. This alternate position is kept if it is better than the current one. If the row exchange produces a worse solution then we randomly choose which position is kept, depending on the difference in fitness and the current temperature.*

A possible simple choice for the temperature management (which is slightly different from the formula presented in Algorithm 2.4 in order to simplify this example) is to start with a value of n^2*, and have it decrease at each iteration so that the temperature is zero after a fixed number of iterations. Then, a worse position may be accepted if the difference in fitness, i.e., the number of additional conflicts between queens, is less than the current temperature. When the temperature becomes strictly less than* 1*, only exchanges improving the quality of the solution are accepted.*

2.2.2 Tabu search

Tabu search is a heuristic strategy created by Fred Glover in 1986-1989 ([Glo89, Glo90], see also [GL98]) which is usually used in combination with another optimization method. The principle is to keep in memory recently visited solutions and to prevent visiting them again at a short-term horizon. For example, a tabu search would forbid a cycling behavior between a small number of very good solutions. Indeed, these solutions will be marked as taboo one after the other, when each point is visited. Thus, this tabu strategy prevents the optimization method from remaining stuck in the same region and enforces a better exploration of the search space.

The set of forbidden solutions, called the *tabu list*, can contain either explicit solutions, or more general features preventing all the corresponding solutions from being examined. Usually, the tabu list has a limited size and old rules are progessively forgotten in favor of new rules. Of course, it is perfectly possible to use rules that definitely remove some regions of the search space, or to bias the search instead of completely forbidding the corresponding solutions.

Tabu search is related to our pruning heuristic presented in Section 4.3.3.

Example 2.3. *Let us consider the n-queens problem. One possibility to implement a tabu search would be to keep a* tabu list *corresponding to the list of the last k queens that have been moved, for some small value of $k < n$. These queens are thus frozen at their current positions. Then, instead of considering the whole neighborhood at a given iteration, we only look at the subset of candidates that do not move the frozen queens from their current positions. As k is constant, the queens associated to the selected neighbor will enter the tabu list and unfreeze the "oldest" frozen queens. Hence, a given queen will not be able to move several times without remaining at the same position during some number of iterations, between two moves. In particular, a move will not be able to directly undo the recent modifications done to the candidate solution. Note that if $k = n - 1$, this corresponds to a strategy that can be interpreted as a discrete version of* coordinate descent*: each variable is optimized separately, one after each other, in a cyclic pattern.*

2.3 Population-based heuristic methods

This section is devoted to population-based techniques and in particular the genetic algorithm which is one of the most famous representatives of the family. Population-based methods correspond to strategies where we keep track of a large set of candidates at each iteration, instead of moving a single candidate in the search space. This is different from a simple restart strategy done in parallel as information from different candidates is combined in population-based strategies.

These methods are mainly designed for exploration of the unknown search space, unlike local search methods which concentrate on the exploitation of the current knowledge. Like simulated annealing, most population-based methods are inspired by processes in the real world such as the behavior of populations of animals or other biological systems.

2.3.1 Genetic algorithm

Genetic algorithms have been proposed by Holland in 1975 ([Hol75], see also [Gol89, Dav91]) and are inspired by the process of natural evolution. Indeed, they produce solutions by letting a population evolve, using mechanisms such as inheritance, crossover, or mutation. The approach is used in many applications, partially due to the fact that the framework of genetic algorithms is very general and can be customized for many optimization problems, giving birth to a bunch of variants.

The search variable of a genetic algorithm is a *population* $\mathcal{P}$. Each *individual* or member of the population $\mathcal{P}$ represents a potential solution to the optimization problem. The individual can be the solution itself, or some encoding of the candidate, i.e., an alternative representation of the solution (typically a string or a list). The initial population is usually randomly generated.

The size S of the population, i.e., the number of individuals kept in memory at each step of the algorithm, is considered a parameter. Increasing the value of S tends to ensure a better exploration, but of course at the price of a higher computation time. Usually, the size of the population remains unchanged during the process, but adaptive population sizing schemes can also be used. The main part of the algorithm consists in a sequence of *generations*. At each generation, a whole process of selection and breeding is done. Indeed, we have to find individuals in the

population that perform well, then produce new candidates by combining these good elements. The breeding is usually done with two components called *crossover* and *mutation*. The detailed steps are presented below.

Evaluation and selection

In order to be able to select population members, we have to somehow quantify their "quality" or *fitness*. At the evaluation step, the performance of each member of $\mathcal{P}$ is evaluated, according to a fitness function. Some modifiers may be applied in order to bias the evaluation procedure. For example, penalties or tie-breakers can be used if solutions satisfying some additional property are preferred.

It has to be noted that this implies that the fitness of an individual should be easily computable. If the evaluation of the objective function is expensive, one may want to consider using an approximation of the fitness function as the population size in genetic algorithms is usually of the order of tens, hundreds or even thousands of individuals.

Afterwards, a subset of $\mathcal{P}$ is selected in order to produce the new population. Several schemes are possible, such as keeping the k elements with the best fitness (elitist strategy), or selecting k individuals with a probability density proportional to the fitness of the elements. Another possibility is to allow an individual to be selected only if its fitness is good enough. One can also allow the best solutions or even all solutions to be selected several times during the procedure.

Even though the fittest elements are logically favored, the other individuals should have decent chances to be selected, in order to preserve genetic diversity and thus a good exploration of the search space. A possible solution is to introduce a small number of randomly generated "strangers" in the selection to counterbalance an elitist strategy.

Crossover

A new population of size S is created by combining elements selected at the selection step. Each new individual is obtained from two selected "parent" solutions. There are a large number of breeding methods and the choice usually depends on the problem and/or the representation of a candidate solution as a population member. Some examples are as follows.

- *The crossover cut:* if the two parents A, B are represented as $a_1a_2\dots a_k$ and $b_1b_2\dots b_k$, respectively, then a new candidate C can be formed by choosing a cut point p satisfying $1 \leqslant p < k$, and taking $C = a_1a_2\dots a_pb_{p+1}\dots p_k$ or $C = a_1a_2\dots a_pb_{p+1}\dots p_k$. It would also be possible to have several cut points.

- *The linear combination:* given two parents A, B, one can produce a candidate C with the formula $C = \alpha A + \beta B$, for some values of α, β. For example, the arithmetic mean of A and B is a possibility that is often used.

- *The fitness-weighted convex combination:* in this case, the coefficients α, β of the linear combination are nonnegative and proportional to the fitness of the two solutions. Furthermore, the coefficients are normalized so that $\alpha + \beta = 1$ (convex combination).

Of course, one may use several crossover rules when creating the new population.

Mutation

With some probability, small random mutations are applied on one or several elements of $\mathcal{P}$. This step ensures that the algorithm will keep analyzing new candidates and avoid staying forever in a local optimum. Indeed, we want to explore the neighborhood of promising products but, at the same time, we need to avoid situations where the population $\mathcal{P}$ is composed of very similar elements as crossing over such populations would not result in new solutions.

It is also possible to apply controlled mutations, i.e., mutations that are not random. This is usually done when we are looking for solutions satisfying some additional property, or if we are using some strategy to improve the quality of the candidates.

Comments

The algorithm usually stops after a fixed number of generations, or if there is no significant improvement to the fitness of the best solution during some number of generations. Genetic algorithms are usually able to reach good (but not necessarily optimal) solutions in a short amount of time. However, convergence to a global optimum or even locally optimal

solutions is not necessarily obtained, depending on the problem and the strategies used in the different steps.

The genetic algorithm approach is used in our methods presented in Sections 3.4.2 and 5.4. In order to help the convergence to solutions that are at least locally optimal, an additional strategy is combined to the genetic algorithm, depending on the optimization problem.

Example 2.4. *Let us design a simple genetic algorithm for the n-queens problem. The population corresponds thus to a set of lists of n integers, each list encoding a solution as explained in the previous sections. The fitness function is again taken to be the number of pairs of non-conflicting queens.*

A crossover cut between two positions can easily be interpreted on the chessboard: we produce a new position by selecting a random column as cut point, then keeping the left part of the first position and the right part of the second position. The hypothesis used here is that we expect good positions to be constituted of good sub-positions (when we only look at a subset of contiguous columns).

The usual random mutation consists in moving (with a low probability) a random queen to a random position in the same column. A less random strategy would be to randomly select a queen, but then moving it to the square (in the same column) that minimizes the number of conflicting pairs of queens. A controlled mutation can be the application of this local improvement strategy at each generation instead of doing it randomly. Another possible controlled mutation is to perform a rotation on the list of integers. For example, $(q_1, q_2, \ldots, q_8)$ *could mutate into* $(q_6, q_7, q_8, q_1, \ldots, q_5)$*. The amplitude of the rotation can be determined by keeping the best position among all rotations of the initial solution.*

2.3.2 Other population-based approaches

Besides the genetic algorithm and its variants, many other population-based methods have been developed based on natural processes, especially over the last two decades. In 1992, *ant colony optimization* algorithms are introduced by Dorigo in his Ph.D. thesis [Dor92]. The approach is based on the behavior of a colony of ants searching for food and using pheromones to mark promising paths, in order to attract other ants on the same path.

These pheromones have a limited-time effect so that an exploration of the search space is still possible. A path will be kept only if it is good enough with respect to other solutions. Indeed, a good path will be frequently taken by the population of ants, reinforcing its density of pheromones. Hence, the method is mainly used when solving optimization problems that can be expressed as path problems.

Another popular method is the *particle swarm optimization* algorithm, proposed by Kennedy and Eberhart in 1995 [KE95], based on social behavior of a population of agents such as a bird flock or a fish school. In this framework, a population of particles is moving in the search space, starting with random initial positions and velocities. At each iteration, the velocity of each particle is computed depending on its current velocity (inertia component), the location of the best solution obtained by the particle in its past, relatively to its current position (personal component), and the location of the best solution among all particles (social component). The positions are then updated according to these velocities.

The precise weighting between the three components is an important parameter of the method since it controls the weighting between exploration and exploitation. This has led to many variants and much research, similarly to the design of the selection, crossover and mutation steps in the genetic algorithm.

Many other optimization methods have been proposed, with varying performance, depending on the design parameters and the optimization problems. Several recent examples are the artificial bee colony method [KB07], the firefly algorithm [Yan09], or the cuckoo search algorithm [YD09].

2.3.3 Hybridizations and memetic algorithms

Even though the previous sections only present a small fraction of existing optimization techniques, one can already wonder which ones are more adapted to a given optimization problem. Moreover, most heuristic methods have a very large freedom in the design of the internal steps, or the tuning of the parameters. This is not surprising as they are mainly very general rules or frameworks that can be adapted to particular problems.

An important observation is that there are two main opposing objectives: the exploration of unknown parts of the search space, and the exploitation of the obtained knowledge in order to produce a better solution. Several methods are more suitable for one of these two goals. Hence, combinations of different heuristic strategies are often used when tackling a particular problem. Combining several approaches to produce another algorithm is sometimes referred to as *hybridizing*. This notion is not specific to genetic algorithms.

Genetic algorithms are usually more adequate as an exploration component. Hence, it may be wise to consider a hybridization, i.e., combination, with another method which aims at producing local improvements with the individuals. This approach is sometimes known as *memetic algorithms* [NCM12]: at a higher level, a population-based strategy is used for exploration. But at a lower level, each individual is separately subject to a local improvement procedure. Note that these subproblems do not have to be completely solved, i.e., until convergence to a local optimum. A small number of iterations may be sufficient and would be much less computationally intensive.

In the following chapters, we present heuristic approaches for three different problems. Chapter 3 deals with the minimum-volume oriented bounding box problem. Our strategy for this problem involves an hybridization of the genetic algorithm with Nelder-Mead simplex algorithm as local improvement procedure, or controlled mutation.

Chapter 4 presents the column subset selection problem, which is a combinatorial NP-hard problem. In this case, the genetic algorithm is avoided. Indeed, the evaluation of the fitness function for an arbitrary solution is generally expensive. The optimization problem is handled using a window-based approach, combined with variants of stochastic hill climbing and tabu search.

Finally, Chapter 5 focuses on the problem of approximating the joint spectral radius, which is also NP-hard. As with the bounding box problem, we use a genetic algorithm with local improvement rules to tackle the problem and derive bounds on the joint spectral radius.

CHAPTER 3

The minimum-volume oriented bounding box problem

In this chapter, we study a well-known problem in computational geometry and computer graphics, the approximation of the *minimum arbitrarily oriented bounding box* of a set of points in $\mathbb{R}^3$. More precisely, the problem consists in finding the minimum-volume rectangular parallelepiped enclosing the given points. We present the main currently used methods and introduce a new approach, where the problem of obtaining the minimum-volume oriented bounding box is formulated as an unconstrained optimization problem on the rotation group $SO(3, \mathbb{R})$. It is solved by a hybrid method combining the genetic and Nelder-Mead algorithms. This method is shown to be either faster or more reliable than other presented methods, for any accuracy. The chapter presents original research published in [CGM11].

3.1 Introduction

The arbitrarily oriented bounding box (OBB) fitting problem is encountered in many applications, such as fast rendering of 3D scenes [AM00], collision detection [JTT00, GASF94], visibility tests [IZK98] or mesh reparameterization [JFGCM10]. In these applications, one has to perform several tests involving geometric objects. For example, collision detection consists in determining whether two geometric objects are intersecting, an important operation done in physics engines. As objects are mainly defined as sets of points, possibly with descriptions of edges and/or faces, testing whether two objects are intersecting or disjoint

is not obvious, especially since it has to be done quickly. Hence, we usually use a simplified hierarchical representation of 3D objects (see [GLM96, PML97]).

The various 3D models can be approximated by a tree of successively smaller volumes ([GLM96] uses such a method), requiring three characteristics:

1. An intersection test between two volumes must have a low computational cost. Indeed, the main purpose of these representations is to be able to perform such operations much faster than with the original objects. This becomes particularly important when we are dealing with a large number of moving objects, since this implies a large number of collision detection tests.

2. The volumes themselves must be quickly computable as the trees have to be built before the intersection tests can be performed. Moreover, the set of considered objects may vary since new elements may appear in the scene. Volumes approximating animated or deformable objects may also have to be modified in real-time.

3. Those volumes need to be as close as possible to the geometry defined by the set of points in order to minimize the number of superfluous tests and thus improve the total running time of the algorithms.

Using volumes such as OBBs, i.e., rectangular parallelepipeds, is thus a compromise between these two objectives. Indeed, testing whether two OBBs are intersecting can be easily done. In particular, we can use the *axis-aligned bounding box* (AABB, see [dB98]), i.e., an OBB whose orientation is parallel to the three axes of the reference frame. Other bounding volumes can of course be used, such as bounding spheres, ellipsoids, cylinders, capsules (cylinders capped by two half-spheres) or k-DOPs (*discrete oriented polytopes* that correspond to AABBs with beveled edges and/or corners), all of which form different compromises between computing ease and accuracy. A review of various bounding volumes can be found in [Eri04].

On the one hand, the AABB is the easiest bounding volume to compute, as we only have to evaluate the range of the coordinates of the points. However, it is obviously not very good at fitting most geometries. Moreover, rotating the object usually results in a modification of

its minimum AABB. On the other hand, the convex hull of the model provides, by definition, the closest convex approximation. However, collisions between convex polyhedra can be just as hard to detect as those involving the original models. This is why OBBs are often a good compromise.

Computing the minimum-volume OBB is far from trivial, although it can be solved in polynomial time using O'Rourke's exact algorithm [O'R85]. Approximations can also be obtained using heuristics such as the ones based on *principal component analysis* (PCA) [Jol02]. Here, we propose a hybrid global optimization algorithm for the computation of the orientation of an optimal OBB that searches in the space of rotation matrices. Our experiments show that optimal OBBs of about 300 test cases from [Gro08] can for example be estimated with relative accuracy of 1% or better in 98% of the runs on average. Running the algorithm on all test cases takes about 4 minutes. In comparison, PCA-based methods only reach such accuracy in less than 60% of the cases. For another choice of parameters of our method, we were able to compute all optimal volumes with a relative error of at most 10^{-12} in more than 95% of the runs on average, in about 20 minutes. In comparison, O'Rourke's method requires nearly one *week* of computation time to compute one optimal OBB for each example. All these computation times include the computation of the convex hulls.

The main idea behind our HYbrid Bounding Box Rotation IDentification (HYBBRID) algorithm is that the problem of finding an optimal OBB can be written as an unconstrained optimization problem on $SO(3, \mathbb{R})$, the special orthogonal group of degree 3. Hence, the objective function is continuous but non-differentiable as it takes into account the geometric constraints. Therefore, it cannot be written in closed form although it is easy to evaluate. This is why solving such a formulation of the problem requires the use of derivative-free optimization methods, such as those used in HYBBRID. It consists in a hybridization of the genetic and Nelder-Mead algorithms, based on the method described in [DA99]. Such a hybrid scheme combines the strength of the genetic algorithm in terms of exploration of the search space, and the capacity of the Nelder-Mead method to quickly converge to locally good solutions.

The remainder of this chapter is organized as follows. We start by presenting the main notation and basic results associated to the problem. This section is followed by an extensive review of the literature on the subject. Next, the formulation of the OBB fitting problem as

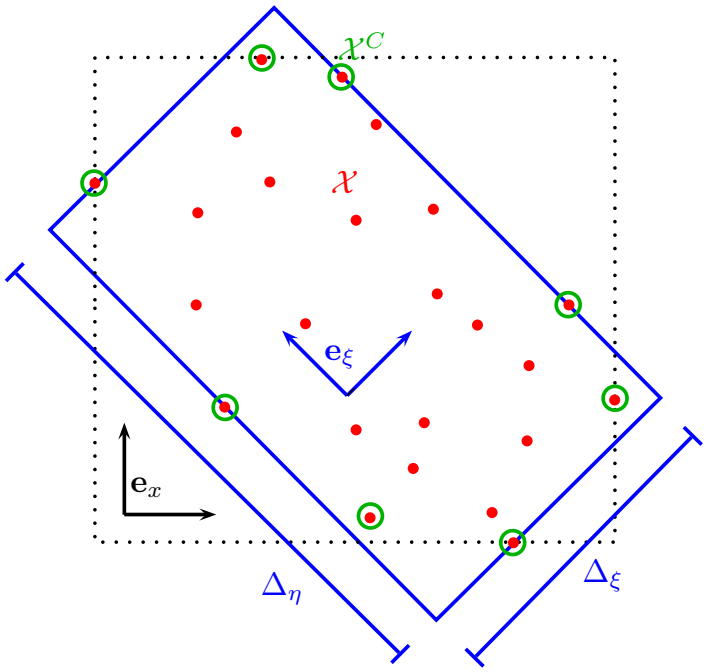


Figure 3.1: *Illustration of the notation associated to OBBs used throughout this chapter, in the two-dimensional case. The axis-aligned bounding box and the optimal oriented bounding box are drawn as dotted black and solid blue lines, respectively.*

an optimization problem is detailed. Our HYBBRID algorithm is then described, analyzed and compared to the other methods.

3.2 Notation and basic properties

Let $\mathcal{X} \subset \mathbb{R}^3$ be a given *finite* set of N points. The problem consists in finding an oriented bounding box, i.e., rectangular parallelepiped, of minimum volume enclosing $\mathcal{X}$. This is illustrated for the 2D case in Figure 3.1. Each OBB is defined by its center $\mathbf{X} \in \mathbb{R}^3$, its dimension $\mathbf{\Delta} \in \mathbb{R}^3$ and its orientation $R \in SO(3, \mathbb{R})$.

Here, the rotation group $SO(3, \mathbb{R})$ is the special orthogonal group of

degree 3 over $\mathbb{R}$:

$$SO(3,\mathbb{R}) = \left\{ R \in \mathbb{R}^{3\times 3} \mid R^T R = I = R R^T, \det(R) = 1 \right\},$$

where $GL(3,\mathbb{R})$ is the general linear group of degree 3, i.e., the set of 3-by-3 invertible real matrices. The matrix R rotates the reference frame $\mathbf{e}_x$ onto $\mathbf{e}_\xi$ as shown in Figure 3.1. Hence, an AABB corresponds to an OBB where the matrix R is chosen as the identity matrix I.

The convex hull of $\mathcal{X}$ and the set of its vertices are denoted by $\text{conv}(\mathcal{X})$ and $\mathcal{X}^C \subset \mathcal{X}$, respectively. Just as $N = |\mathcal{X}|$, let $N_V = |\mathcal{X}^C|$ be the number of vertices of $\text{conv}(\mathcal{X})$. The computation of this convex hull can be used as a preprocessing step before computing an OBB, as only the points in $\mathcal{X}^C$ have an influence on the OBB. In $\mathbb{R}^2$ and $\mathbb{R}^3$, the convex hull of a set of N points can be obtained with a cost of $\mathcal{O}(N \log N)$ [Ski98], for example by using Qhull [BDH96]. Note that the volume of the convex hull gives a lower bound on the volume of the minimal OBB.

The problem of computing an optimal OBB for a given set of points is not trivial. In 2D, an optimal bounding rectangle can be computed in linear time using the so-called *rotating calipers* method as proposed in [Tou83]. This technique is based on the idea developed by Michael Ian Shamos in his Ph.D. thesis [Sha78] (see also [PS85]) to compute the diameter of a convex polygon. The fact that the 2D problem can be easily solved is due to the following property.

Theorem 3.1 ([FS75])**.** *Given a set of points $\mathcal{X}$ in $\mathbb{R}^2$, the minimum-volume rectangle enclosing all points in $\mathcal{X}$ has at least one side aligned with an edge of the polygon* $\text{conv}(\mathcal{X})$.

In other words, at least one side of the optimal bounding rectangle contains at least two points of $\mathcal{X}^C$ (The upper right side for the example in Figure 3.1). Thus, one does not have to test all orientations in order to find the smallest bounding rectangle.

The rotating calipers algorithm for the minimum-area bounding rectangle in $\mathbb{R}^2$ is given in Algorithm 3.1.

In this algorithm, the four indices i_{down}, i_{right}, i_{up}, i_{left} correspond to the indices of four points located on the four sides of the bounding rectangle. The vector $d_{\text{down}} = (d_x, d_y)$ corresponds to the direction of one side of the current bounding rectangle, and thus completely defines the orientation of this rectangle. The two parallel vectors d_{down} and

Algorithm 3.1 Rotating calipers for the minimum-area bounding rectangle problem

Input: $(z_0, z_1, \dots z_{N_V-1})$ with $z_i = (x_i, y_i)$, ordered list of the coordinates of the vertices of the convex hull $\mathrm{conv}(\mathcal{X})$. In this algorithm, all indices are supposed to be modulo N_V. The edges of the convex hull are thus $[z_i, z_{i+1}]$, for $i = 0, \dots, N_V - 1$.

Output: f area of the minimum-area bounding rectangle enclosing X^C

$f := +\infty$
$i_{\text{down}} := \text{argmin}_i\, y_i$
$i_{\text{right}} := \text{argmax}_i\, x_i$
$i_{\text{up}} := \text{argmax}_i\, y_i$
$i_{\text{left}} := \text{argmin}_i\, x_i$
$(d_x, d_y) := (1, 0)$
while $d_x > 0$ **do**
  $\delta_j := z_{i_j+1} - z_{i_j}$ for $j \in \{\text{down}, \text{right}, \text{up}, \text{left}\}$
  Let $d_{\text{down}} = (d_x, d_y)$, $d_{\text{right}} = (-d_y, d_x)$, $d_{\text{up}} = (-d_x, -d_y)$ and $d_{\text{left}} = (d_y, -d_x)$
  $j_{\max} := \text{argmax}_j \left(\delta_j \cdot d_j / \|\delta_j\|\right)$
  $d_{\text{new}} := z_{i_{j_{\max}}+1} - z_{i_{j_{\max}}}$
  Update d_x, d_y such that $d_{j_{\max}} = d_{\text{new}} / \|d_{\text{new}}\|$
  $i_{j_{\max}} := i_{j_{\max}} + 1$
  Compute the area f_{cur} of the bounding rectangle with directions (d_x, d_y) and $(-d_y, d_x)$
  if $f_{\text{cur}} < f$ **then**
    $f := f_{\text{cur}}$
  end if
end while

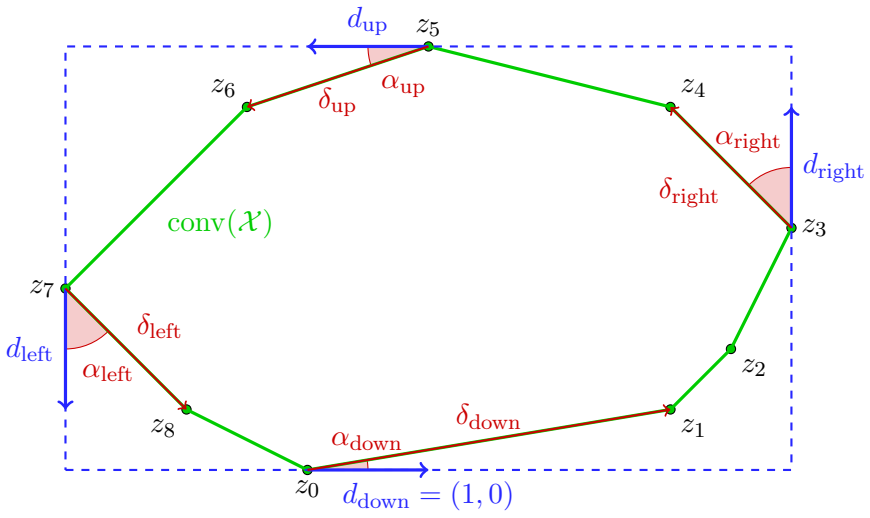


Figure 3.2: *The rotating calipers algorithm for the minimum-area bounding rectangle problem. In this situation, we have* $i_{\text{down}} = 0$, $i_{\text{right}} = 3$, $i_{\text{up}} = 5$ *and* $i_{\text{left}} = 7$.

d_{up} define a first pair of *calipers*, and a second pair is defined by d_{left} and d_{right}. These two pairs of calipers are then rotated around $\text{conv}(\mathcal{X})$. Whenever a caliper is aligned with an edge of the convex hull, the area of the corresponding bounding rectangle is computed. The rotation angle between two such events can be obtained by comparing the α_j angles between the δ_j's and the d_j's (see Figure 3.2). Since $0 \leqslant \alpha_j \leqslant \pi$, the smallest angle (which determines the next rotation to apply) is associated to the largest value of the cosines, and thus to the largest value of the normalized dot products. After a total rotation of at most $\frac{\pi}{2}$, all edges have been tested.

A naive generalization to the three-dimensional case would be the following conjecture.

Conjecture 3.2 (FALSE)**.** *Given a set of points* $\mathcal{X}$ *in* $\mathbb{R}^3$*, the minimum-volume OBB enclosing all points in* $\mathcal{X}$ *has at least one face aligned with a face of the polyhedron* $\text{conv}(\mathcal{X})$.

Unfortunately, this assertion does not hold, as shown by the simple counterexample in Figure 3.3. Indeed, it is clear that none of the six faces of the cube is parallel to a face of the inscribed regular tetrahedron.

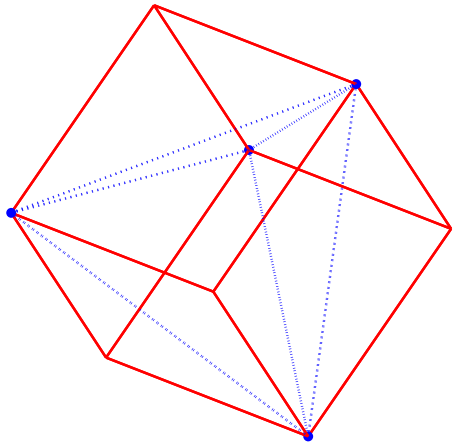


Figure 3.3: *Optimal OBB of a regular tetrahedron (in blue dotted lines). Here, the optimal OBB is a cube (in red solid lines).*

However, it should be noted that the *edges* of the tetrahedron are all embedded into the faces of the cube.

In fact, in the 3D case, the following theorem holds.

Theorem 3.3 ([O'R85])**.** *A rectangular parallelepiped of minimum volume circumscribing a convex polyhedron must have at least two adjacent faces that contain edges of the polyhedron.*

In Theorem 3.3, the two faces are said to be *flush* with the corresponding edges. This theorem shows that it is possible to find the optimal OBB by examining a finite number of orientations, similarly to the 2D case.

3.3 State of the art

The current best exact algorithm for the 3D problem, published by O'Rourke in 1985 [O'R85] and based on Theorem 3.3, has a time complexity of $\mathcal{O}(N_V^3)$. O'Rourke's algorithm is too slow to be of practical use and is known to be extremely hard to implement [BHP01, Eri04].

Most of the time, heuristic approaches are used instead. The most popular ones are based either on principal component analysis [Jol02] or on brute-force search. Note that given one axis $\mathbf{p}$ of an optimal OBB, the two remaining axes can easily be obtained by using the rotating calipers technique to compute the minimum-area rectangle enclosing the set of points projected on a plane orthogonal to $\mathbf{p}$. Several methods use this idea of finding the orientation of the OBB, aligned with a given direction $\mathbf{p}$, that has the minimum volume by solving the associated 2D problem.

In the following subsections, these algorithms are presented, but a more detailed discussion can be found in [Eri04]. We have implemented them all in MATLAB® in order to compare these currently widespread methods for OBB fitting. Several studies comparing the performances of bounding box algorithms can also be found in [DHKK09, LKM+00].

3.3.1 O'Rourke's algorithm

In [O'R85], Joseph O'Rourke presented an algorithm that can be used to compute an optimal OBB of a set of points in 3D. Although exact, this method has the main drawbacks of being both extremely complicated and very slow. It can be seen as a generalization of the rotating calipers for the 3D case. Indeed, it is a smart exhaustive search across all potential optimal orientations of the bounding box.

Based on Theorem 3.3, O'Rourke devised an algorithm that examines every pair (e_1, e_2) of edges of $\text{conv}(\mathcal{X})$. The idea is to perform a rotation of the OBB such that a face and an adjacent one are continuously flush with e_1 and e_2, respectively. Such a rotation is shown in Figure 3.4. The volume of the OBB is a continuous but non-smooth function of the rotation matrix R. Indeed, the derivative is not continuous each time a third edge is flush with one face of the OBB. Between two such particular rotations, the volume is a rational function whose local minima can be obtained from the roots of a polynomial of degree 6. If one of these volumes or the volume with three flush edges is smaller than the current best volume found, the incumbent optimal solution is updated.

This algorithm runs in cubic time since the computational cost for each pair of edges is linear in N_V, as in the 2D rotating calipers technique, and there are $\mathcal{O}(N_V^2)$ pairs of edges. We have implemented O'Rourke's algorithm in order to compute the optimal volume and thus verify the accuracy of the other methods.

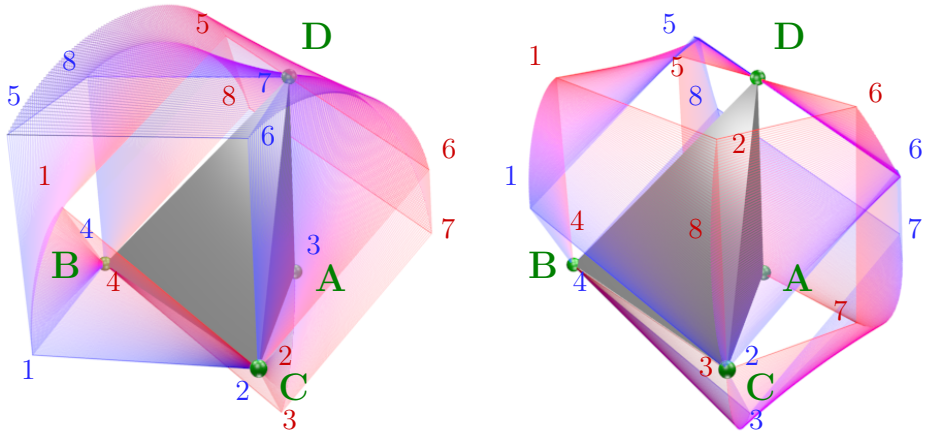


Figure 3.4: *Illustration of two successive steps of the rotation involved in O'Rourke's algorithm for the tetrahedron $ABCD$ with vertices $\{(0,0,0),(1,0,0),(0,1,0),\ (0,0,1)\}$. During the whole rotation, the adjacent faces* 3784 *and* 1234 *are flush with the edges AB and BC, respectively. Hence, both faces evolve in a plane rotating around their corresponding edge. In general, the rotation of the other faces are not around a fixed axis as adjacent faces of the OBB have to stay orthogonal while being flush with at least one vertex of the tetrahedron. The first OBB is the cube in blue of which the faces* 1234, 3784 *and* 2376 *are flush with the faces ABC, ADB and ADC of the tetrahedron, respectively. In fact, the latter is a corner of this cube; hence, all edges are flush with faces of this particular OBB. In the intermediate bounding box, which is in red (resp. blue) on the left (resp. right), the face* 1256 *is flush with the edge CD; since this particular OBB has three edges flush, it corresponds to a salient point of the volume function during the rotation. As far as the last bounding box, in red on the right, is concerned, the faces* 3784 *and* 5678 *are flush with the face ABC and the edge AD, respectively. Note that its vertex* 8 *is hidden behind the tetrahedron.*

3.3.2 PCA-based methods

A very popular class of heuristic methods is the one based on principal component analysis. The idea behind it is to first perform a PCA on $\mathcal{X}$, that is, computing the eigenvectors of its covariance matrix and choosing them as axes of the orthonormal frame $\mathbf{e}_\xi$. The first (resp. last) axis is the direction of largest (resp. smallest) variance. Either some or all axes resulting from the PCA can then be used, the choice leading to three different variants of the methods: All-PCA, Max-PCA and Min-PCA [BCG+96]. In general, PCA-based methods do not require the computation of the convex hull and run in linear time with respect to the number of points. In fact, using $\mathcal{X}^C$ instead of $\mathcal{X}$ may change the bounding box returned by these three variants as the distribution of points is modified. Depending on the set of points, using X^C may yield better *or* worse results than using $\mathcal{X}$.

All-PCA

This is the most basic method. It consists in directly using the three directions given by the PCA as the axes of the OBB. In practice, $\mathcal{X}$ is simply rotated to be placed in the PCA frame, and the AABB in this frame is obtained by computing the minimum and maximum values for each component.

Though this method is particularly fast and easy to implement, it can be shown that the ratio between the volumes of the OBB obtained using PCA and the optimal volume is unbounded [DKKR09]. In fact, the PCA is very sensitive to the way points are distributed, and can fail to give good results even in simple cases. For example, the PCA yields a very badly fitted bounding box if the points form two crosses roughly on two parallel rectangles, as shown in Figure 3.5. This is why one of the two following variants are often used to improve the quality of the solution.

Max-PCA and Min-PCA

The idea here is to use only one of the three axes determined by the PCA. The orientation of the OBB, aligned with this axis, that has the minimum volume can then be obtained by solving the associated 2D problem obtained by projecting the points on the plane formed by the

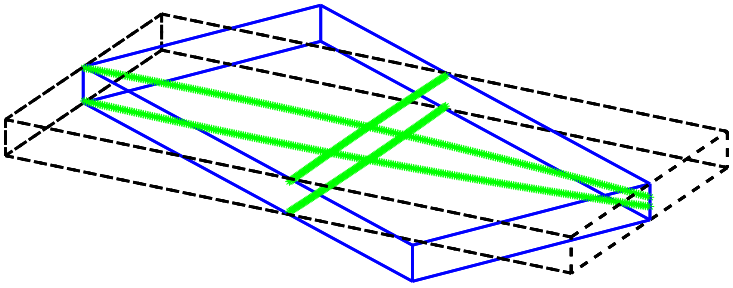


Figure 3.5: *Simple case showing the OBBs given by different PCA-based methods on a set of points (in green). The two bounding boxes correspond to the result obtained by All-PCA or Max-PCA (black dashed lines), and Min-PCA or Continuous PCA (blue solid lines).*

two unused axes. In the Max-PCA variant, the direction of projection is given by the first axis which is the eigenvector corresponding to the largest eigenvalue of the covariance matrix. On the other hand, one can choose to keep the last axis instead and thus obtain the Min-PCA variant.

These methods are a little slower, but are always more accurate than All-PCA. Indeed, the orientation given by All-PCA corresponds to a feasible solution of both 2D optimization problems solved by Min-PCA and Max-PCA with rotating calipers. Of course, the two corresponding *optimal* orientations are at least as good as the one obtained by All-PCA. Using Min-PCA or Max-PCA may avoid some of the pitfalls of All-PCA. For example, using Min-PCA (resp. Max-PCA) will give an almost optimal bounding box if $\mathcal{X}$ has a predominantly two-dimensional (resp. one-dimensional) shape. Figure 3.5 illustrates the case of an object that has one dimension with a significantly much smaller range than the others. On sets of points with a real 3D extension, the Min-PCA variant tends to yield better results in practice.

Continuous PCA

As regular PCA-based algorithms are too sensitive to the distribution of the points in the set, Stefan Gottschalk, Ming Lin and Dinesh Manocha have proposed the following two improvements in [GLM96]:

1. Only the vertices of $\text{conv}(\mathcal{X})$ determine the optimal bounding box. Hence, computing the PCA simply on $\mathcal{X}^C$ should provide better results.

2. This reduced set of points can still give poor results; it typically happens when points are very concentrated in one region of the convex hull. A resampling of $\text{conv}(\mathcal{X})$ should be performed in order to obtain a uniformly spread out set.

Those two improvements yield a slightly more complex method because, to be completely general, the convex hull needs to be resampled "infinitely densely"; this leads to a reformulation of the covariance matrix as a continuous form. However, it is shown in [DKKR09] that, for an octahedron, the volume of the bounding box given by this method is still four times bigger than the optimal volume.

3.3.3 Brute-force methods

Given the complexity of the problem, a common simple strategy is to examine a large number of possible orientations and choose the one yielding the best OBB. These methods are used quite regularly given the ease of implementation. Some of them are independent of the distribution of points in the geometry, unlike PCA-based heuristics.

There are two main approaches. First, one can decide to sample the search space and try all these possible orientations. For example, a uniform distribution can be used to sample $SO(3, \mathbb{R})$, and then try all these rotation matrices. Another possibility is to consider a large set of points on the unit sphere, each point defining a direction $\mathbf{p}$. The orientation of the OBB, aligned with p, that has the minimum volume can then be obtained by solving the associated 2D problem using the rotating calipers technique. In either case, the best OBB obtained can be relatively close to the real optimum provided that the number of considered orientations is sufficiently large. Obviously, the major drawback of this approach is the large computation time required to check all these orientations. More details can be found in [Eri04].

A second approach is to select a large set of candidates based on some properties of the geometry. For instance, for every direction that connects any two points of the set, the associated 2D problem can be solved to obtain the orientation of the OBB. The one that yields the

smallest volume is then selected. This heuristic is described in [BHP01] as the *all-pairs* method and runs in cubic time as the number of possible pairs grows quadratically with the size of $\mathcal{X}$.

Many other strategies can be used. In [Kor08], three variants are described. The fastest one was already proposed in the discussion section of [O'R85] and corresponds to a search among all bounding boxes of which one side coincides with one face of the convex hull. It is thus a generalization of the 2D naive algorithm based on Theorem 3.1 and runs in $\mathcal{O}(N_V^2)$ time. Both other strategies also take edges of the convex hull into account but are slower ($\mathcal{O}(N_V^3)$ running time). In fact, Korsawe's slowest variant is even slower than O'Rourke's algorithm (see Section 3.5.3).

3.3.4 (1+ε)-approximation

In [BHP01], Gill Barequet and Sariel Har-Peled have proposed algorithms to compute, for any value of $\varepsilon \in]0, 1]$, an approximating OBB whose volume is at most $(1+\varepsilon)$ times the optimal volume. For that purpose, a grid in $\mathbb{R}^3$ whose discretization step d is inversely proportional to ε is built. Its orientation is given by an OBB whose volume is a constant factor approximation of the optimal one. This OBB is chosen aligned with an approximation of the diameter of $\mathcal{X}$ that is computed using the AABB.

Based on this grid, two very different methods can be designed. On the one hand, $\mathcal{X}$ can be projected on the grid with a computational cost that is linear in N. O'Rourke's algorithm can then be performed on the projected set. In addition to requiring an implementation of O'Rourke's method, a main drawback of this method is that it can be very slow in practice if d is too large. This first method is called ApproxMinVolBbx and has a complexity of $\mathcal{O}(N + \frac{1}{\varepsilon^{4.5}})$.

On the other hand, each vector pointing from the center of the grid to one of its nodes can be taken as a direction $\mathbf{p}$. The orientation of the OBB aligned with $\mathbf{p}$, that has the minimum volume, can then be obtained by solving the associated 2D problem. For this variant, it is necessary to choose how many cells of the grid will be considered to build a candidate OBB. This is done by considering only the cells that have a Chebyshev (or infinity-norm) distance of d or less from the center of the grid. The way to compute the value of d to reach an accuracy of

ε is exposed in [BHP01]. This second method is referred to as GridSearchMinVolBbx in the article and has a complexity of $\mathcal{O}(\frac{N_V}{\varepsilon^3})$, with an additional cost of $\mathcal{O}(N \log N)$ for the computation of the convex hull of $\mathcal{X}$ as preprocessing.

Note that the original technique described in [BHP01] can be improved. Indeed, it is possible to reduce the size of the projected set of points to a smaller set as described in [AHPV04, Cha06].

3.4 A new method based on optimization

The approach we propose in this thesis consists in formulating the search of the minimum-volume OBB as an optimization problem defined on a manifold. To the best of our knowledge, this idea has only been used in [LKM+00]. In that article, the authors combine Powell's quadratic convergent method [PTVF92] and a multi-scale grid search to minimize the volume of the OBB. A comparison of this scheme with our algorithm can be found in Section 3.4.4. The method we propose is based on a formulation of the minimum OBB problem as an unconstrained optimization problem on $SO(3, \mathbb{R})$, which is presented below.

3.4.1 Formulation of the optimization problem

A first direct optimization formulation can be written as follows:

$$\begin{aligned} \min_{\substack{\boldsymbol{\Delta},\boldsymbol{\Xi}\in\mathbb{R}^3 \\ R\in SO(3,\mathbb{R})}} \quad & \Delta_\xi \Delta_\eta \Delta_\zeta \\ \text{s.t.} \quad & -\frac{\boldsymbol{\Delta}}{2} \leqslant R\mathbf{X}_i - \boldsymbol{\Xi} \leqslant \frac{\boldsymbol{\Delta}}{2} \qquad \forall i \in \{1, \ldots, N\}, \end{aligned} \tag{3.1}$$

where $\mathbf{X}_i \in \mathcal{X}$, $\boldsymbol{\Delta} = (\Delta_\xi, \Delta_\eta, \Delta_\zeta)$ denotes the dimensions of the OBB and $\boldsymbol{\Xi}$ its center after rotation by R. Note that the $\leqslant$ operator is applied componentwise.

The objective function is trilinear and the constraints are linear. One of the major difficulties in this problem is the feasible set of R. The minimization over the two vectors $\boldsymbol{\Delta}, \boldsymbol{\Xi}$ in $\mathbb{R}^3$ and the rotation matrix R can be carried out as two successive minimizations. One can single out the minimization with respect to R, and the other "internal" one

is simply given by the computation of an AABB after a rotation of $\mathcal{X}$ defined by a given fixed matrix R. Thus, the problem (3.1) can also be written in the following form:

$$\min_{R \in SO(3,\mathbb{R})} f(R), \tag{3.2}$$

where the objective function $f(R)$ is simply the volume of the AABB of $\mathcal{X}$ rotated by R:

$$f(R) = \left(\begin{array}{cc} \min\limits_{\boldsymbol{\Delta},\boldsymbol{\Xi} \in \mathbb{R}^3} & \Delta_\xi \Delta_\eta \Delta_\zeta \\ \text{s.t.} & -\frac{\boldsymbol{\Delta}}{2} \leqslant R\mathbf{X}_i - \boldsymbol{\Xi} \leqslant \frac{\boldsymbol{\Delta}}{2} \quad \forall i \in \{1, \ldots, N\} \end{array} \right).$$

Δ_ξ (resp. Δ_η, Δ_ζ) can indeed be computed as the difference between the maximum and minimum values of the first (resp. second, third) components of the rotated set of points.

This function $f(R)$ is only $\mathcal{C}^0$ since it is not differentiable at every rotation matrix R that brings at least one face of the OBB to be flush with one edge of the convex hull. These particular rotations are the equivalent in $SO(3,\mathbb{R})$ of salient points (points where two branches of a curve meet with different tangents) and potentially yield local minima of this objective function. A 2D example illustrating the function $f(R)$ is presented in Figure 3.6. It appears that this function is formed by the upper envelope of concave functions. Note that the problem (3.2) is also interesting in the 2D case, even though the rotating calipers technique already provides an efficient — and asymptotically optimal — linear algorithm.

Because of the non-differentiability of the function, especially at local minima, line-search methods such as steepest descent or Newton can encounter convergence issues. Therefore, derivative-free methods have been preferred. A short presentation of different methods can be found in Chapter 2.

Note that this non-smoothness comes from the elimination of the constraints in problem (3.1). Even though problem (3.2) is unconstrained on $SO(3,\mathbb{R})$ with an easy-to-evaluate objective function, it would be possible to try to solve problem (3.1) as a constrained smooth optimization problem. This alternative approach will also be included in the results presented in Section 3.5.3.

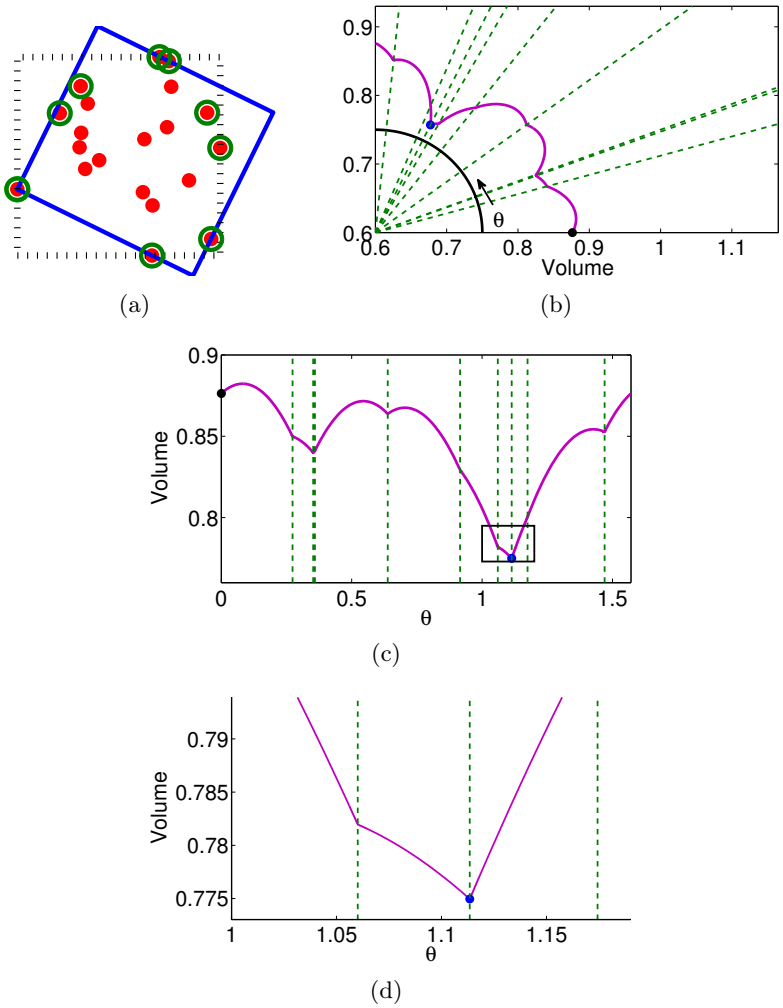


Figure 3.6: *Representation of the area of an oriented bounding rectangle depending on its orientation. (a) A 2D example is shown with the same conventions as those used in Figure 3.1. (b) The area of the minimum oriented bounding rectangle is shown for all directions around a quarter of the circle. This is sufficient given the symmetries of a rectangle. Salient points corresponding to angles aligning the bounding rectangle with a face of* $\mathrm{conv}(\mathcal{X})$ *are shown by dashed lines. (c) The same curve is also drawn as a function of the angle* θ *ranging from* 0 *to* $\frac{\pi}{2}$*. It appears that only four of the salient points are local minima. (d) A close-up view on the global minimum is shown. Note that the salient point at* $\theta \approx 1.05$ *is not a local minimum as increasing the value of* θ *reduces the area of the bounding rectangle.*

3.4.2 The HYbrid Bounding Box Rotation IDentification (HYBBRID) algorithm

In order to find a global minimum of the volume function $f(R)$, a local search component to ensure the convergence to a minimum and a global search component to explore the search space can be combined. Several schemes are possible but in this thesis, we have chosen to focus on a hybrid method combining the Nelder-Mead algorithm (see Section 2.1.3) and the genetic algorithm (see Section 2.3.1), as preliminary tests were showing very promising results. A genetic algorithm without some local improvement rule is not adequate when the search space is continuous, whereas a pure Nelder-Mead would have to be restarted a large number of times in order to find a good local optimum (see Section 3.5.2). Another example of an applicable global-local scheme for this problem is the particle swarm optimization method (see [BA10]).

By combining Nelder-Mead simplex algorithm and genetic algorithm one could hope to obtain very good solutions in a short time. This idea has been studied by several authors: in [CS03] the authors first use a continuous genetic algorithm with elitist strategy to locate a "promising area" in the search space, where the Nelder-Mead algorithm is then used to try to find the best solution situated in that region. Another example is given by Durand and Alliot in [DA99], where the authors apply a genetic algorithm with p-tuples of points as population elements, with several Nelder-Mead iterations applied at each generation.

The combination used in HYBBRID is close to the variant of Durand and Alliot and is detailed hereafter, step by step. As the dimension of the rotation group $SO(3,\mathbb{R})$ is 3, a simplex is a set of four rotation matrices $\mathcal{R} = \{R_1, R_2, R_3, R_4\} \subset SO(3,\mathbb{R})$. Hence, it forms a tetrahedron on this manifold with R_j's at its vertices. An element A_k of the population $\mathcal{A}$ is thus a simplex $\mathcal{R}$ and its *fitness* is defined as $\min_{j\in\{1,\dots,4\}} f(R_j)$. The Nelder-Mead algorithm will also have to be adapted as we are optimizing on $SO(3,\mathbb{R})$ instead of $\mathbb{R}^3$. Generalizations of the Nelder-Mead simplex algorithm to Riemannian manifolds have already been studied in [Dre06].

We will first describe the different steps of the HYBBRID algorithm in terms of the usual operations in a linear space, and then explain in Section 3.4.3 how these operations are implemented on the manifold $SO(3,\mathbb{R})$.

Initialization

Let M be the size of the total population. It is initialized with random simplices, i.e., the four vertices R_j of each simplex are obtained by using a QR factorization of random 3-by-3 real matrices.

Evaluation and selection

The fitness of all the simplices is evaluated. The best $\frac{M}{2}$ simplices are selected, the others are discarded. From this reduced population, four groups $\mathcal{A}_1^I, \mathcal{A}_2^I, \mathcal{A}_1^{II}, \mathcal{A}_2^{II}$ are created at random using a uniform distribution. Each group has $\frac{M}{2}$ elements, and each population member can be selected any number of times, in any number of groups.

Crossover

A standard *mixing* crossover is applied between the two groups $\mathcal{A}_1^I$ and $\mathcal{A}_2^I$. A pair of parents is constituted by choosing the k^{th} element of both subpopulations: $A_1 \in \mathcal{A}_1^I$ and $A_2 \in \mathcal{A}_2^I$. They produce an offspring A_{new}. Each vertex of the simplex A_{new} is either the corresponding vertex of A_1 or of A_2, the selection being random, but the parent with the best fitness having a higher probability of being chosen. In our algorithm, the probability to select one of the two parents is 0.6 (resp. 0.5 or 0.4) if it is better (resp. equivalent, worse) than the other parent. This first crossover gives us $\frac{M}{2}$ new simplices.

The other $\frac{M}{2}$ new simplices are given by an affine combination crossover between $\mathcal{A}_1^{II}$ and $\mathcal{A}_2^{II}$. Let $A_1 \in \mathcal{A}_1^{II}$, $A_2 \in \mathcal{A}_2^{II}$ be the k^{th} pair of parents as before. The four vertices $A_{\text{new},j}$ of the corresponding offspring A_{new} are defined by

$$A_{\text{new},j} = \lambda A_{1,j} + (1 - \lambda) A_{2,j},$$

where the value of λ depends on whether A_1 is better or worse than A_2. For example, λ can take the value 0.4 (resp. 0.5, 0.6) if the fitness value of A_1 is smaller (resp. equal, larger) than that of A_2.

Nelder-Mead mutation

K Nelder-Mead iterations (as described in Section 2.1.3) are applied on *all* the M new simplices in order to obtain the new generation of the population. This is thus expected to improve the general fitness of the whole population.

Stopping criterion

This process (Selection $\rightarrow$ Crossover $\rightarrow$ Mutation) is repeated until a stopping criterion is met, usually if the fitness of the best simplex stalls for several iterations with respect to the desired tolerance, or if a maximal number of iterations is reached. In our case, the algorithm stops after k consecutive generations where the objective value does not improve by at least $x\%$ compared to the current best value, with $k = 5$ and $x = 1$ as default values for these parameters.

Comments

The goal of the genetic component of HYBBRID is to somehow compute the initial condition so that the Nelder-Mead algorithm converges to a global minimum. These steps, inspired by evolutionary biology, bring correlations between the initial conditions, which is better than starting with random simplices (see Section 3.5.2).

Using Nelder-Mead simplices seems to be a better choice than directly considering rotation matrices with a mutation consisting in a line-search method. Indeed, the use of Nelder-Mead simplices induces a layer of local cooperation between groups of candidate solutions. This meshes well with the global cooperation introduced by the genetic algorithm that randomly "resets" the initial conditions of the Nelder-Mead algorithm.

Experiments showed that using only one of the two crossover steps yields poorer performances. Combining both of them ensures that the simplices move enough to explore the search space sufficiently (first crossover) while still tending to gather around promising areas (second crossover). Of course, other crossovers can also be considered.

Finally, a post-processing step can be applied to the OBB obtained from any algorithm: as the 2D problem is easy to solve with the rotating calipers technique, the set $\mathcal{X}$ can be projected along one of the axes of

the candidate OBB and the associated 2D problem can be solved. This amounts to a rotation of the box around the normal of one of the faces and ensures *local* optimality in that direction. A post-processed OBB is thus guaranteed to be locally optimal with respect to its elementary rotations.

3.4.3 Taking into account the structure of $SO(3,\mathbb{R})$

The second crossover in the genetic component of HYBBRID and the update of the simplices in the Nelder-Mead algorithm consist of affine combinations of rotation matrices. However, the affine combination of two or more rotation matrices is not a rotation matrix, in the general case. The geodesics can be used in order to take into account the geometry of $SO(3,\mathbb{R})$ in these computations. On this particular manifold, the exponential map and the log map [KHM07] can be used.

Let us look at the reflection and expansion steps in the Nelder-Mead algorithm. A simplex is defined by four vertices which are rotation matrices here: (R_1, R_2, R_3, R_4). The rotation matrix R_0 is somehow defined as the "centroid" of these four points. The mathematical operation that brings R_4 on R_0 is the left multiplication by the rotation matrix $R_0 R_4^T$ since $R_0 = (R_0 R_4^T) R_4$. In the reflection step, by definition, R_r is obtained by performing the same displacement on the manifold but starting from R_0 instead of R_4. Hence, the reflection point can simply be computed as $R_r = R_0 R_4^T R_0$. Similarly, the expansion point can be expressed as $R_e = (R_0 R_4^T)^2 R_0$ as the displacement has to be done twice. The new vertices in the contraction and reduction steps can also be written in a closed form, e.g., $R_c = (R_0 R_4^T)^{1/2} R_4$.

The computation of a square root of a matrix is significantly more expensive than simple multiplications. However, a high accuracy is not required in our Nelder-Mead method as only a few number of iterations are applied to let the population get closer to the minima at each generation. In this case, the method can take advantage of the fact that $SO(3,\mathbb{R})$ is embedded in $\mathbb{R}^{3\times 3}$. After computing the usual affine combination in $\mathbb{R}^{3\times 3}$, the obtained matrix can be projected on $SO(3,\mathbb{R})$ with a QR factorization to obtain an approximation of the affine combination on this manifold [SS09]. Note that this projection is less adequate for the reflection and expansion steps as these are non-convex combinations. Nevertheless, such extrapolations are simply avoided as for the reflection

Computation method	Computation time	Error measure
Definition $C = (AB^T)^{1/2}B$ using MATLAB® `sqrtm`	376.3 s	0
Polar factorization $C = UV^T$ with $\frac{1}{2}(A+B) = U\Sigma V^T$	17.0 s	0
QR factorization $C = Q$ with $\frac{1}{2}(A+B) = QR$	8.2 s	0.063 rad $\approx 3.6°$

Table 3.1: *Performance of different methods to compute the Nelder-Mead contracted point C on the manifold $SO(3,\mathbb{R})$. The operations have been done 1000000 times with random matrices $A, B \in \mathbb{R}^{3\times 3}$. The error measure corresponds to the average value of the angle of the rotation between the approximated contracted point and the exact one.*

and expansion steps, the geodesic on $SO(3,\mathbb{R})$ can be followed exactly with a low computational cost as we have shown before.

The proper projection on $SO(3,\mathbb{R})$ is actually the polar factorization which is orthogonal for the inner product defining the Frobenius norm [GvL96]. However, it requires one to perform a singular value decomposition which is more expensive. In practice, it is observed that using the QR factorization is faster than polar decomposition without losing significant accuracy (see Table 3.1). The error is small enough so that in practice, it does not affect the bounding boxes returned by HYBBRID.

A remaining point is the computation of the centroid R_0. Indeed, averaging on manifolds is in general nontrivial (see [KHM07] for the Karcher mean on $SO(n,\mathbb{R})$ or [Moa02] for averages on $SO(3,\mathbb{R})$). In our case, we use the same strategy as with the contraction or reduction steps: the usual arithmetic mean is computed in $\mathbb{R}^3$ and the resulting matrix is projected on $SO(3,\mathbb{R})$ with a QR factorization.

3.4.4 Comparison with the algorithm of Lahanas et al.

Unfortunately, it is difficult to empirically compare the efficiency of HYBBRID with the algorithm of [LKM+00] because the way Powell's method is applied and the choice of the parameters of the multi-scale grid search

method are not detailed in the article. Nevertheless, let us emphasize the main differences from a theoretical point of view between these two hybrid approaches.

First of all, the formulation of the problem is not defined on the same search space. On the one hand, the principle of HYBBRID is to minimize an objective function on the rotation group $SO(3,\mathbb{R})$. Each evaluation of this cost function requires an AABB computation which is a trivial optimization subproblem. One important property of this method is that the search space is viewed as a manifold without a global parameterization. On the other hand, the search space on which the optimization problem is formulated in [LKM+00] is parameterized by the triplet $(\phi, \cos\theta, \alpha)$, i.e., the azimuth angle, the cosine of the zenith angle and the angle of rotation around the axis defined by ϕ and θ, respectively. A drawback of this choice is the singularity induced at the poles. One main interest of formulating the optimization problem on a manifold is to avoid such issues induced by the parameterization.

Both algorithms consist in a hybridization using an exploration and an exploitation component. The latter is the Nelder-Mead (resp. Powell's) method for HYBBRID (resp. the algorithm of Lahanas et al.). On the one hand, the Nelder-Mead simplex search is an intuitive heuristic that is very popular due to its simplicity and empirical efficiency, at least for problems of dimension less than 5. This is the case here as the dimension of the $SO(3,\mathbb{R})$ rotation group is 3. At each iteration of this method, the vertices of the simplex induce an implicit model that is used to determine the next simplex with just a few evaluations of the cost function. On the other hand, Powell's method is a line-search method that requires solving a succession of one-dimensional minimization subproblems. As far as the exploration component is concerned, HYBBRID is based on the genetic algorithm which is stochastic while the multi-scale grid search method is a priori deterministic.

3.5 Experimental analysis

All the methods presented in the previous sections (except the method of Lahanas et al.) have been implemented using MATLAB® and tested on about 300 sets of points from [Gro08]. These examples include a wide selection of different geometries, ranging from simple shapes to anatomical objects defined by millions of points. As a bounding box only

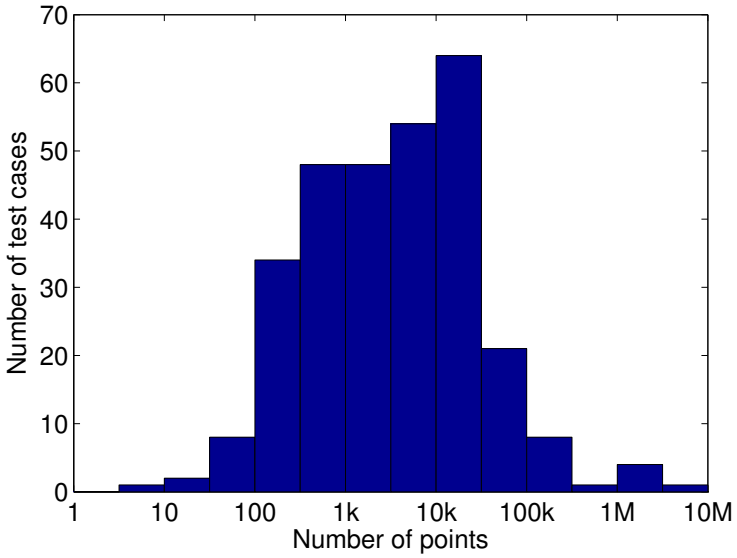


Figure 3.7: *Distribution of the number of points of the different objects used as test instances.*

depends on the convex hull of the object, computing it as a preprocessing step is a good way to reduce the number of points in the subsequent computations.

The distributions of the number of points of the objects and of vertices on their convex hull, shown in Figures 3.7 and 3.8, highlight the interest of such preprocessing. The characteristics of four of those examples are given in Table 3.2, while a graphical representation is shown in Figure 3.9. This figure is rendered in GMSH [GJF09], and the green meshes represent the convex hull of the objects.

Note that for about 15% of these objects, the AABB coincides with an optimal OBB. Furthermore, for about 40% of the test cases, the AABB has at least one face parallel to a face of an optimal OBB.

In the remaining of this section, a study of the properties and behavior of HYBBRID is first presented in Section 3.5.1. This method is then compared to the different techniques introduced in Section 3.3 based on experimental results.

Name	N	Computation time of conv($\mathcal{X}$)	vol(OBB)/vol(conv($\mathcal{X}$))	
			Optimal OBB	Random OBB (min – median – max)
`heart482`	88608	0.1227 s	1.9719	1.9858 – 2.5857 – 3.0301
`hand770`	47590	0.0630 s	2.1087	2.1481 – 5.0217 – 6.5993
`balljoint4074`	137062	0.2337 s	1.8926	1.8989 – 2.9993 – 3.6982
`globe9306`	19568	0.1108 s	1.8057	1.8170 – 2.1155 – 2.3150

Table 3.2: *Characteristics of four examples of tested sets of points. The number in the name corresponds to N_V, the number of vertices of* conv($\mathcal{X}$)*, and the datasets are ordered in increasing values of N_V. The second and third columns give the size N of the original set of points and the time required to compute the convex hull, respectively. For the convex hull computation, the algorithm used is Qhull [BDH96], which is written in C. The fourth column corresponds to the volume ratio for the minimum OBB. The fifth column shows the minimal, median and maximal volume ratio respectively, obtained by considering 100000 randomly oriented bounding boxes. All computations have been carried out using MATLAB® 7.6.0.324 (R2008a) on an Intel® Core™ 2 Duo 2.80 GHz with 3 GiB RAM, running Ubuntu Linux 10.04.*

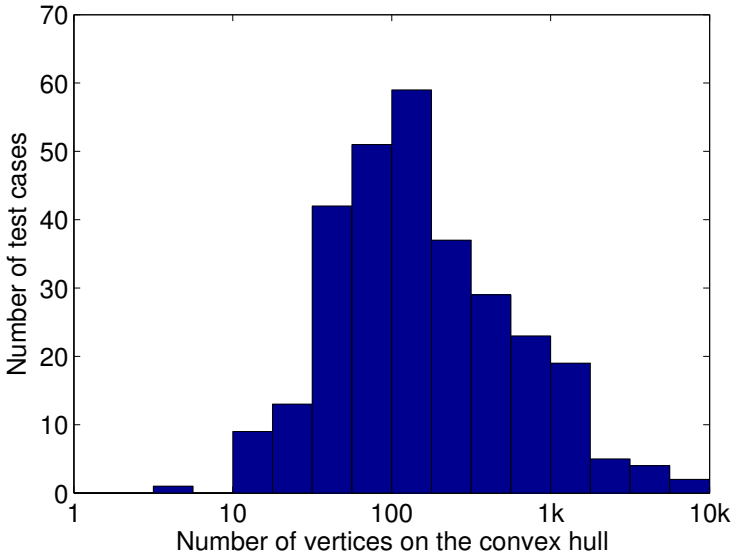
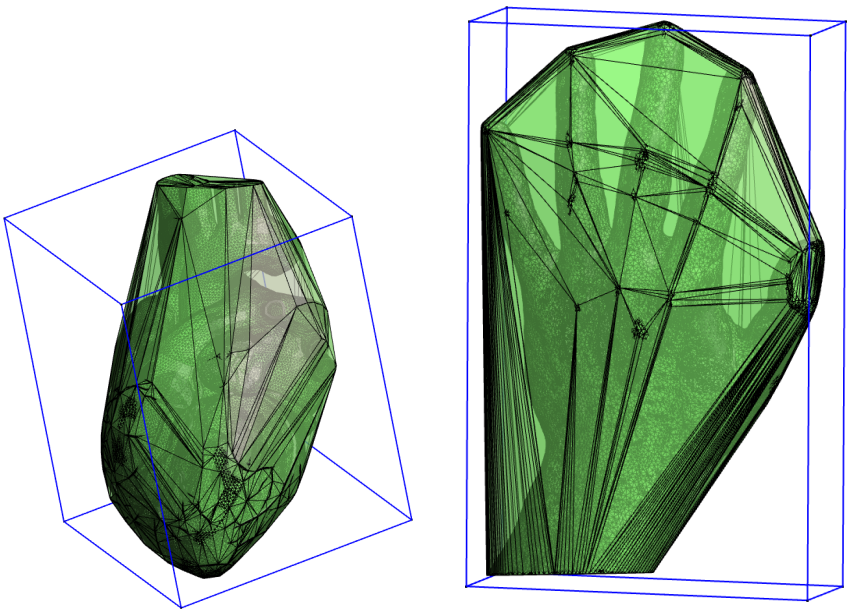


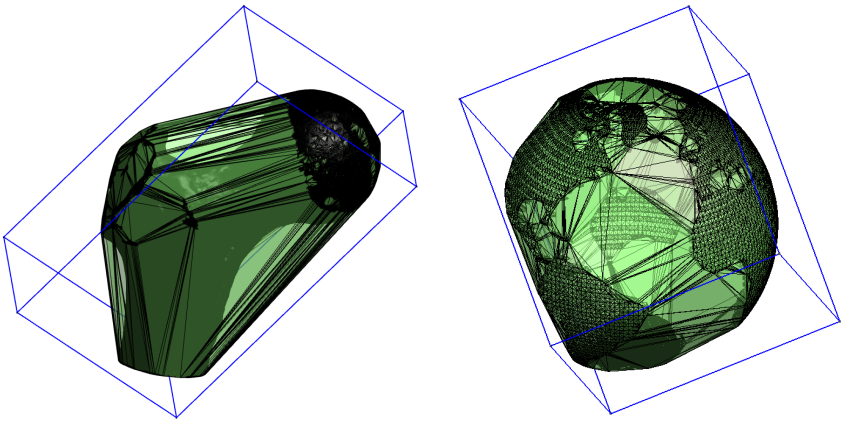
Figure 3.8: *Distribution of the number of vertices on the convex hulls of the test instances.*

3.5.1 Performance of the HYBBRID method

The HYBBRID method was tested 200 times on each object of the test set presented in the previous section. HYBBRID was able to find an optimal OBB for each dataset. Nevertheless, this solution is not reached at each run because of the random component of the genetic algorithm. The actual success rate depends on parameters such as M, the size of the population, and K, the number of Nelder-Mead iterations. With carefully chosen values this success rate may be brought close to 100%. However, changing the values of the parameters also influences the computation time required by the algorithm. Hence, we will first study the asymptotic time complexity of the algorithm in the following subsection. Then, we will analyze the effect of the two parameters M and K in terms of performance and computation time. Finally, the reasons why a suboptimal solution may be returned are explained as well as how it is possible to modify the algorithm in order to take these facts into account.



(a) `heart482` (b) `hand770`



(c) `balljoint4074` (d) `globe9306`

Figure 3.9: *Graphical representation of the four examples described in Table 3.2, along with their convex hull and the optimal OBB.*

Complexity of the algorithm

One major drawback of O'Rourke's algorithm is its cubic time complexity, whereas the 2D problem can be solved with a linear complexity assuming the convex hull is known. An interesting point to investigate is thus the time complexity of HYBBRID. In Figure 3.10, the computation time needed by HYBBRID when run on each test case with $M = 30$ and $K = 10$ is represented. The stopping criterion used in these experiments is the following: "If there is no volume improvement of at least 1% during at least 5 iterations, then the search is aborted". This corresponds to the criterion mentioned previously.

These experimental results tend to show that the asymptotic time complexity of this method would be linear with respect to the number of vertices on the convex hull N_V. Of course, this linear complexity does not include the convex hull computation done as a preprocessing step. The asymptotic linear complexity is illustrated by the red dotted line that is obtained using a weighted linear regression; indeed, data corresponding to test cases of large size have been given more weight, as we are looking for an asymptotic behavior.

As each iteration of HYBBRID takes $\mathcal{O}(N_V)$ time, which is the complexity of evaluating the volume of an AABB, the observation that the asymptotic time complexity of this optimization method seems linear would imply that the number of generations required to produce a solution is independent of the set of points. Note also that taking another pair of parameters would change the computation times, but the asymptotic complexity is expected to be the same.

Theoretically, the asymptotic time complexity of an iteration could be further lowered to $\mathcal{O}(\log N_V)$ by using the method described in [EM85] to locate extreme points. Unfortunately, the data structure described therein is difficult to implement efficiently in MATLAB® due to limitations in the usage of pointers in the language. Moreover, given the size of the test cases used, it seems that using this extreme points locating technique would not improve the total computation time in practice, as the overhead introduced by the construction of the hierarchical structure would be too high. Indeed, this additional preprocessing step requires $\mathcal{O}(N_V \log N_V)$ time.

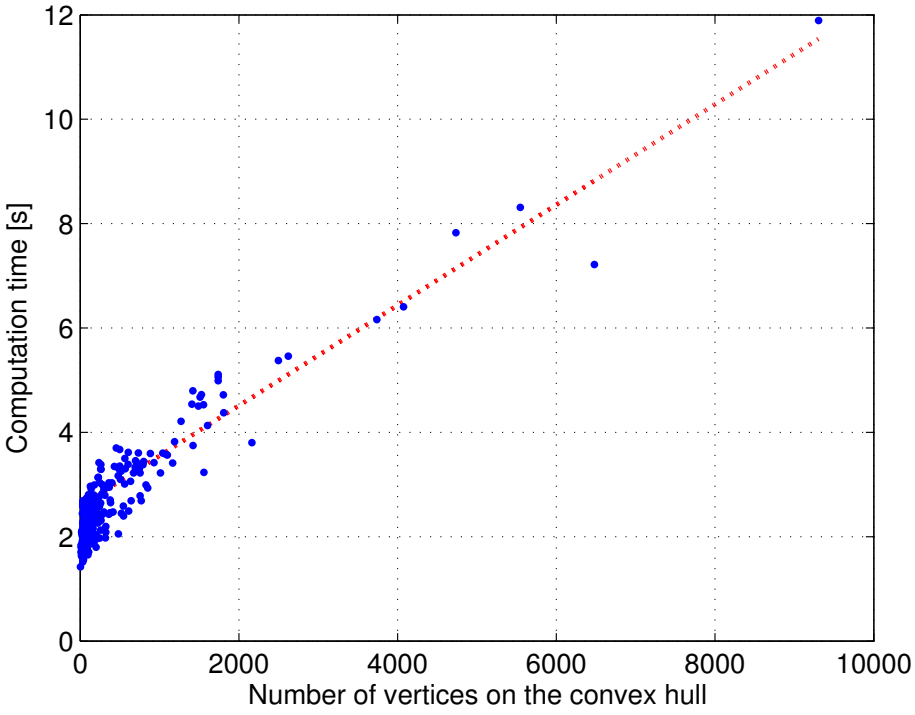


Figure 3.10: *Evolution of the computation time required by HYBBRID with datasets of increasing sizes, excluding the time required to compute the convex hull. Computer specifications are identical to what is described for Table 3.2. The results were obtained by running the method 200 times on each test case and averaging the computation times.*

Influence of the values of the parameters

In Figure 3.11, the reliability of our algorithm is studied through the performance and the computation time of HYBBRID for different sets of parameters. The performance is measured by the proportion of runs where the optimal volume has been obtained within an accuracy of 10^{-12}. It is possible to check this as the exact optimal volume is known, thanks to O'Rourke's algorithm.

Each example has been tested 200 times in order to take into account the variability due to the randomness inherent to HYBBRID. The stopping criterion is the same as in the previous section: "If there is no volume improvement of at least 1% during at least 5 iterations, then the search is aborted".

It appears that increasing the number of Nelder-Mead iterations K yields the same general trend as increasing the population size M: this increases both the reliability and the computation time. Because of this natural trade-off, the optimal choice of parameters depends on the requirements for each particular application. For a given population size, the performance increases significantly with the number of Nelder-Mead iterations until about $K = 20$. After this threshold value of 20, the performance gain seems much less interesting, especially for large population sizes. Hence, one interpretation of this figure is that using 20 Nelder-Mead iterations is somehow optimal; the population size can then be chosen depending on the needs.

Note that for very small values of K such as 0 or 1, the performance is very weak. Indeed, this corresponds to a nearly pure genetic algorithm, whereas the main idea of HYBBRID is to keep locally improving the candidates found by exploring the search space, and repeatedly combining these improved solutions. As expected, the performance gain obtained by increasing the population size is more significant for small values of M: increasing $M = 10$ by 10 units is equivalent to an increase of 100%, whereas going from $M = 40$ to 50 is only an increase of 25%.

To summarize, based on these experimental results, we suggest taking $K = 20$ and then choosing M depending on the available time and the desired reliability. Note that if parallelization is considered for HYBBRID, one should take into account the fact that the genetic component is easily parallelizable whereas at each generation and for each population member, the Nelder-Mead iterations have to be applied sequentially. Hence in this case, it may be less expensive to increase the size M of the

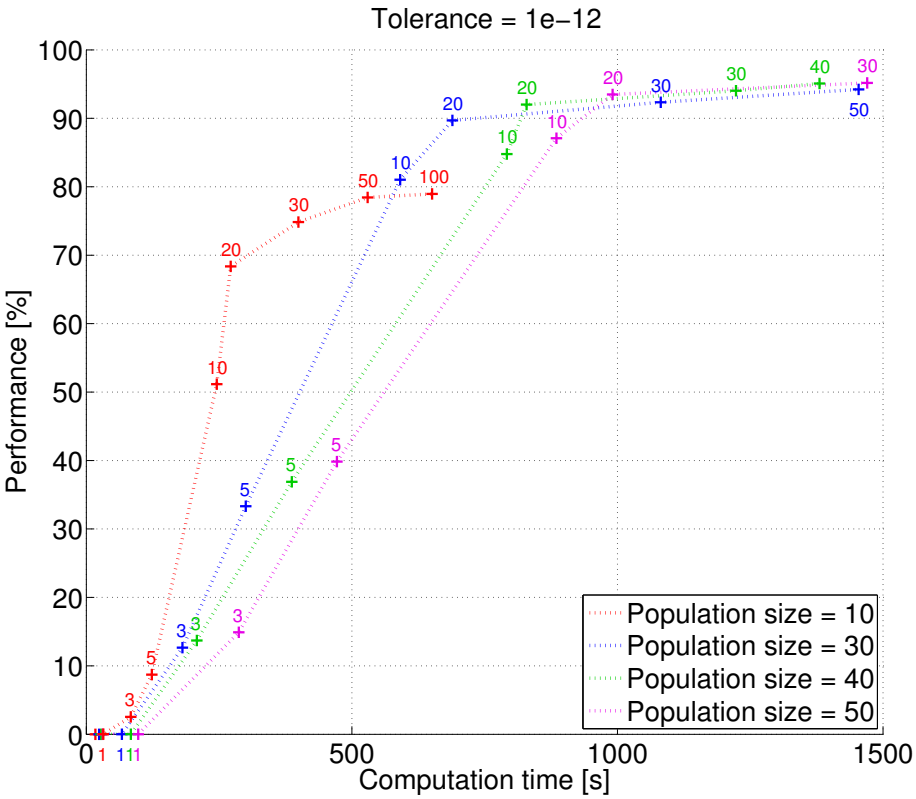


Figure 3.11: *Computation time and performance of the HYBBRID algorithm for different sets of parameters* (M, K)*. The color of the points corresponds to the population size, whereas the numerical labels correspond to the number of Nelder-Mead iterations at each generation. The computation time corresponds to the total time required to approximate the OBBs of all 300 objects, including the computation of the convex hulls. For each test case, the results were obtained by running the method 200 times and averaging the results.*

population than the number K of Nelder-Mead iterations and another value of K may be more appropriate.

Causes of suboptimal performance

Two main reasons may account for the fact that HYBBRID may sometimes miss the optimal volume. For instance, with our set of test cases, a suboptimal result was returned in about 4% of the runs on average with the pair of parameters $(M, K) = (50, 30)$.

On the one hand, the algorithm consists in exploring the search space thanks to the evolution of a population of simplices. If the search is interrupted too early, a good exploration cannot be ensured; hence, a suboptimal solution may be returned. As there is no simple way to verify whether a given candidate bounding box is optimal, a time-accuracy trade-off has to be made with the stopping criterion. We have chosen a stopping criterion of the form: "If there is no improvement of at least x% during at least k iterations, then the search is stopped". Typical values used in the experiments are $x = 1$ and $k = 5$. Increasing the value of k for example would increase the expected number of iterations and thus the exploration of the search space. Many different schemes are possible, and the choice between a faster algorithm and more exploration should be made depending on the application.

On the other hand, it is possible for all simplices to get stuck in a local minimum after some iterations. This behavior is desired if the minimum is global, but this may sometimes happen with suboptimal solutions. One way to avoid such a situation could be to introduce random mutations, e.g., at each iteration, one population member is replaced by a random simplex. Another possibility is to apply random perturbations on some or all simplices with a given small probability. The choice of such a random mutation strategy is again a trade-off problem as using too many random mutations may reduce the effect of the improvements given by the Nelder-Mead algorithm. Conversely, with a small random mutation factor, the simplices may remain at a suboptimal solution for too long, possibly activating the stopping criterion. In this work, we have chosen not to use such random mutations for the sake of simplicity, but this strategy can be easily included in the algorithm. Moreover, the results show that the correct bounding box is found in nearly all cases, and at least a good solution is found in all cases, as it can be observed in the figures in the previous subsection.

3.5.2 Comparison of HYBBRID to other simple iterative strategies

Before comparing HYBBRID to the other methods presented in Section 3.3, let us first look at alternative iterative approaches to the minimum-volume bounding box problem. Since HYBBRID uses a genetic algorithm in order to produce good starting points for the Nelder-Mead simplex algorithm, one possibility is to observe the behavior of this algorithm with random initial conditions. A first interesting alternative is thus a Nelder-Mead algorithm with random restarts. Note that given the same computational resources, the number of restarts for the simple Nelder-Mead algorithm is not equal to the population size in HYBBRID since HYBBRID only applies a small number of Nelder-Mead iterations at each generation.

Another approach is to try to solve problem (3.1) as a constrained optimization problem. Let us recall the problem hereafter:

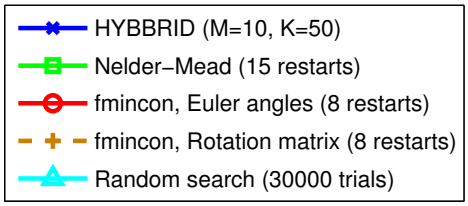
$$\begin{aligned} \min_{\substack{\boldsymbol{\Delta},\boldsymbol{\Xi}\in\mathbb{R}^3 \\ R\in SO(3,\mathbb{R})}} \quad & \Delta_\xi \Delta_\eta \Delta_\zeta \\ \text{s.t.} \quad & -\tfrac{\boldsymbol{\Delta}}{2} \leqslant R\mathbf{X}_i - \boldsymbol{\Xi} \leqslant \tfrac{\boldsymbol{\Delta}}{2} \qquad \forall i \in \{1,\ldots,N\}, \end{aligned}$$

This smooth optimization problem has 15 real variables: 9 for the rotation matrix R, 3 for the size $\boldsymbol{\Delta}$ of the bounding box and 3 for the center $\boldsymbol{\Xi}$ of the bounding box. The constraint $R \in SO(3,\mathbb{R})$ can be expressed as a set of 6 nonlinear equality constraints that correspond to the matrix relation $R^T R = I$. Note that there are only 6 independent constraints instead of 9 because the matrix $R^T R$ is symmetric. In our experiments, this constrained optimization problem is solved using the `fmincon` function in MATLAB® Optimization Toolbox version 4.0.

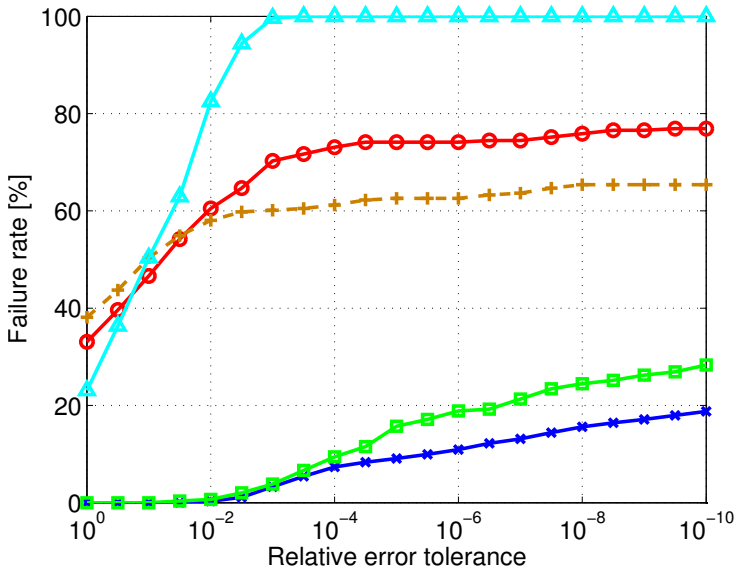
Another possibility is to solve the same problem but using the Euler angles formalism instead of a formulation with rotation matrices. In that case, there are only 9 real variables but the constraints $-\frac{\boldsymbol{\Delta}}{2} \leqslant R\mathbf{X}_i - \boldsymbol{\Xi} \leqslant \frac{\boldsymbol{\Delta}}{2}$ become nonlinear with respect to the Euler angles. The results obtained by `fmincon` with this formulation are also presented in the comparison below.

Finally, as a reference point, we have also included results from a naive random search that simply tries a large number of bounding box orientations. All these results can be found in Figures 3.12 and 3.13.

Let us look at the figures. First, it clearly appears that a pure random search is inefficient. There is no significant improvement when the

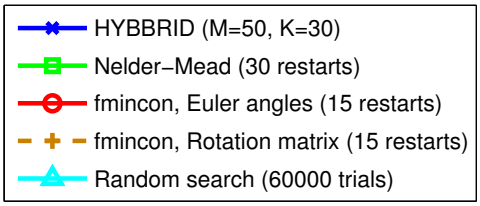


(a) Legend

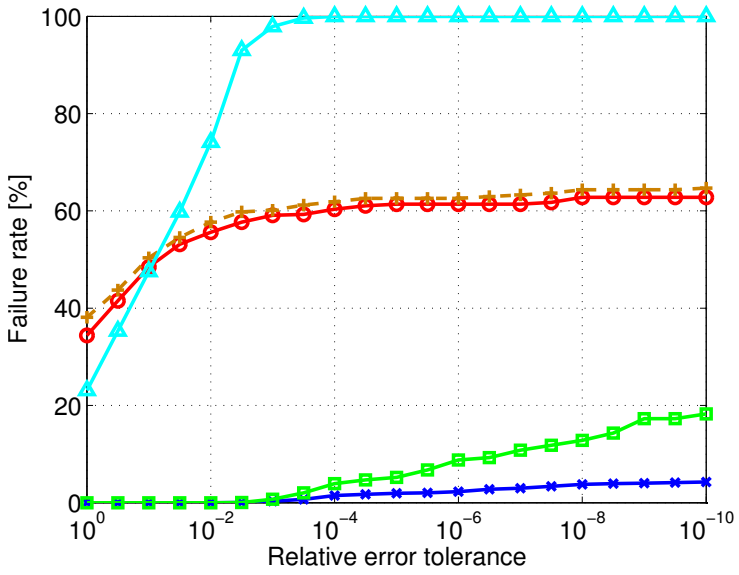


(b) Results

Figure 3.12: *Failure rate of iterative algorithms with restarts, for several tolerance thresholds. For each test case and each threshold τ, the run is considered a failure if the relative error is greater than τ. The parameters of HYBBRID and the number of restarts have been chosen so that the computation time is about* 500 *seconds for each algorithm.*



(a) Legend



(b) Results

Figure 3.13: *Failure rate of iterative algorithms with restarts, for several tolerance thresholds. For each test case and each threshold τ, the run is considered a failure if the relative error is greater than τ. The parameters of HYBBRID and the number of restarts have been chosen so that the computation time is about* 1000 *seconds for each algorithm.*

number of trials is increased from 30000 to 60000 and in fact, this is the case even with 300000 trials. Using the post-processing procedure would also be insufficient: 1000 random trials with post-processing takes about 2600 seconds for a performance close to the results obtained after 30000 trials without post-processing. This justifies the use of nontrivial optimization methods.

The constrained smooth optimization approach seems to be able to find an optimal bounding box in 40% of the test cases but the accuracy is much lower in the remaining instances. The general behavior of the performance as a function of the tolerance does not seem to depend on the formalism even though there is a small difference in Figure 3.12. This difference can be interpreted as a sign that approaching the constrained problem (3.1) with the Euler angles formalism seems to produce more local minima. Thus, a larger number of restarts is required in order to reach the same performance as with the rotation matrix formalism. In fact, running `fmincon` with an even smaller number of restarts shows a bigger gap: when the algorithm is only run once, the Euler angles version has a failure rate of 95% for a tolerance $\tau = 10^{-10}$, whereas the rotation matrix version has a failure rate of 73% at the same tolerance. The success rate is mostly unaffected by the post-processing step. This is not so surprising as `fmincon` is supposed to find a locally optimal solution.

Finally, the Nelder-Mead simplex algorithm applied to the problem (3.2) appears to yield much better results. This supports our choice of solving the unconstrained non-smooth problem instead of the constrained smooth problem. Still, the Nelder-Mead approach (with restarts) does not perform better than HYBBRID, which justifies the use of a genetic algorithm instead of just trying random initial points.

3.5.3 Comparison of HYBBRID to the state of the art

In this section, the proposed algorithm is compared to the other methods described in Section 3.3. First, observations are made on a few very simple examples to highlight some of the strengths and shortcomings of the different techniques. Then, all the methods are compared on the whole set of test cases to extract more information about how they compare in terms of computation time and reliability.

Basic properties of the different methods

The different methods with some of their general characteristics are presented in Table 3.3. It appears that HYBBRID is the only iterative method in the table. This is an advantage as iterative methods tend to be more robust than direct ones. Note that only all-pairs, Korsawe's and O'Rourke's methods are not linear with respect to N_V; the first two being much easier to implement than the third.

The computation of the convex hull is optional for HYBBRID. Indeed, keeping only the points on the convex hull does not change the result returned by HYBBRID but it may reduce the total computation time depending on the values of N and N_V. In the following results, the computation of $\mathrm{conv}(\mathcal{X})$ will be done as this gives a faster algorithm in nearly all test cases. The situation is different with All-PCA, Max-PCA and Min-PCA as taking the convex hull of the set of points may result in a different OBB which is not always smaller.

The number of lines specified in the table is roughly the number of MATLAB® code lines that were specifically written to implement the method. That is why some methods have a particularly low number of lines, for example Gottschalk et al.'s one. Using another language, where more low-level numerical operations would have to be either implemented from scratch or imported from a library would require more code. For other methods, namely PCA and Korsawe's, the number shown is for the whole set of variants. For example, writing a dedicated MATLAB® script for All-PCA would not require more than ten lines of code.

Note that both HYBBRID and Barequet and Har-Peled's algorithm are methods with parameters (M, K for HYBBRID and ε (or equivalently d) for Barequet and Har-Peled). It is thus possible to obtain several compromises between accuracy and computation time, depending on the application.

It is also interesting to investigate which methods are exact on very simple examples. A first example is given by an arbitrary rotation of the following set of points:

$$S = \{(-1, -0.1, 0), (-1, 0.1, 0), (1, 0, -0.1), (1, 0, 0.1)\},$$

which is taken from [BHP01]. The bounding boxes obtained by several methods are shown in Figure 3.14. Several observations can be

Method	Category	Accuracy	Complexity	Implementation
O'Rourke	Direct, enumeration	Guaranteed exact	$\mathcal{O}(N_V^3)^\star$	$\sim$ 500 lines
HYBBRID	Iterative, optimization	Often exact in practice Parametric	$\mathcal{O}(N_V)^{\star\star}$ (experimental)	$\sim$ 400 lines
GridSearch-MinVolBbx	Direct, enumeration	$(1+\varepsilon)$-approximation	$\mathcal{O}(\frac{N_V}{\varepsilon^3})^\star$	$\sim$ 100 lines
PCA	Direct	Suboptimal	$\mathcal{O}(N)$	$\sim$ 100 lines
Gottschalk et al.	Direct	Suboptimal	$\mathcal{O}(N_V)^\star$	$\sim$ 20 lines
All-pairs	Direct, enumeration	Suboptimal	$\mathcal{O}(N_V^3)^\star$	$\sim$ 100 lines
Korsawe	Direct, enumeration	Suboptimal	$\mathcal{O}(N_V^{\{2,3\}})^\star$	$\sim$ 200 lines

Table 3.3: *General characteristics of the methods. Columns 2 and 3 show the type and the accuracy of the methods. The fourth column gives the worst-case asymptotic time complexity of the algorithms, in terms of the number of points N in the set, and the number N_V of vertices of the convex hull. The computation of* $\mathrm{conv}(\mathcal{X})$ *(with complexity $\mathcal{O}(N \log N)$, not included in the term in the fourth column) is needed for methods labeled with $\star$. It is not necessary but generally recommended for HYBBRID, which is indicated by the label $\star\star$. The last column indicates the approximate number of lines of code of the implementation.*

made with regard to this example. First of all, O'Rourke's algorithm and HYBBRID are able to compute the optimal OBB. Conversely, most PCA-based methods are suboptimal without the post-processing, but all are optimal with it. In fact, the post-processing step rotates the bounding box around its three axes to try to find a better OBB. As at least one of the selected axes is correct in each case (the one corresponding to the main dimension of the dataset), a rotation around this axis is thus sufficient to obtain the optimal result. Note that Max-PCA is already optimal without this post-processing step since the main dimension of the dataset actually corresponds to the principal axis of the dataset in this case. A brute-force method like all-pairs or naive variants of Korsawe's are unable to find the correct solution. Indeed, the optimal OBB has no face orthogonal to an edge of $\text{conv}(\mathcal{X})$, nor parallel to a face of the convex hull. One would need a more elaborate brute-force method to reach the optimum.

Concerning the GridSearchMinVolBbx method by Barequet and Har-Peled's, the choice of the unit cell of the grid used in the algorithm influences the value of the parameter d that is required for a given accuracy. This unit cell is determined by an approximation of the diameter of $\mathcal{X}$, which is computed using the AABB. Hence, the grid and the performance of the algorithm depend on the initial orientation of the set of points. Even for this simple example rotated arbitrarily, GridSearchMinVolBbx is only guaranteed to yield the optimal volume to any accuracy for sufficiently large values of d. For example, with the unrotated set S, GridSearchMinVolBbx finds the optimal solution with a grid of size $d = 1$. However, with the rotated set $\widetilde{S}$ defined by:

$$\widetilde{S} = RS, \qquad R = \frac{1}{4}\begin{pmatrix} 0 & 2 & 2\sqrt{3} \\ 2\sqrt{3} & -\sqrt{3} & 1 \\ 2 & 3 & -\sqrt{3} \end{pmatrix},$$

GridSearchMinVolBbx returns a solution whose volume has a relative error of 14% with $d = 20$. Hence, although the algorithm will converge to the optimal value for $d \to \infty$, the orientation of $\mathcal{X}$ can significantly influence the quality of the results returned by this algorithm for finite values of d.

Another simple example is the tetrahedron obtained by a random rotation of the set:

$$T = \{(0,0,0),(0,1,1),(1,0,1),(1,1,0)\}.$$

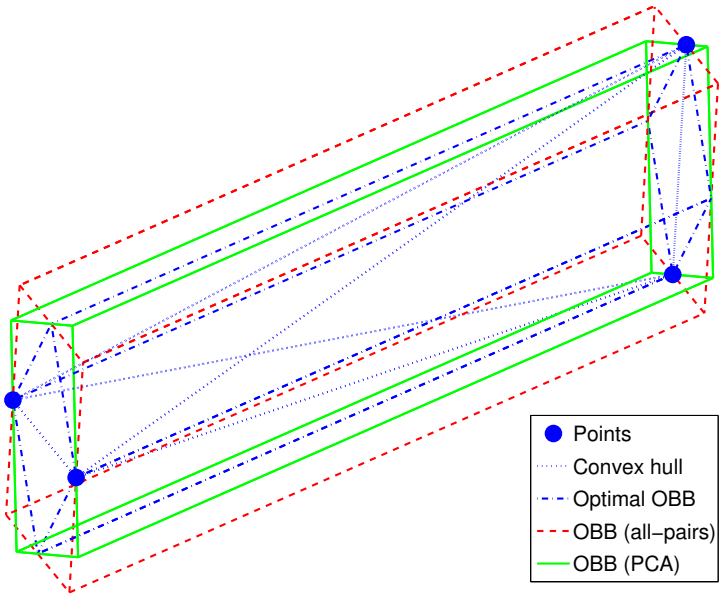


Figure 3.14: *Bounding boxes obtained by several methods, for the set S after rotation. O'Rourke's algorithm, HYBBRID, PCA-based and Gottschalk et al.'s method* with post-processing *and a variant of Korsawe's one are optimal (blue box). The all-pairs method gives the red suboptimal box that is two times larger, as does another variant of Korsawe's algorithm. The green box is obtained using PCA, but without post-processing.*

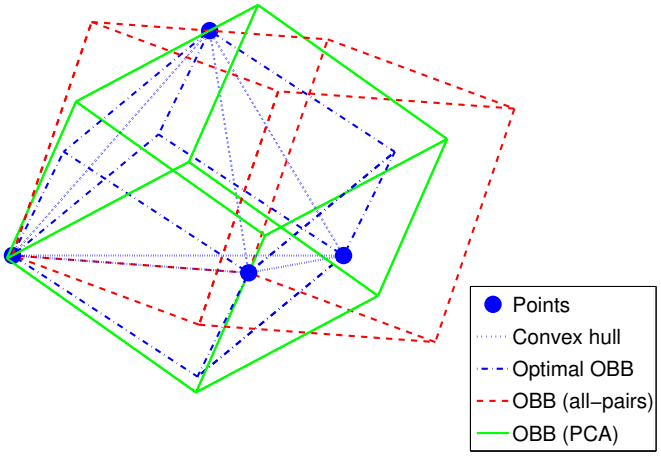


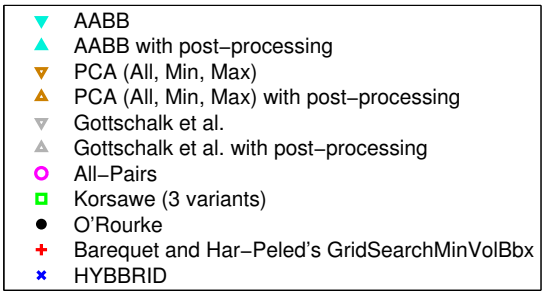
Figure 3.15: *Bounding boxes obtained by several methods, for the tetrahedron T after rotation. The optimal bounding box, in blue, is only obtained by O'Rourke's algorithm, HYBBRID and one variant of Korsawe's method. The red box can be obtained by using all-pairs method, whereas Min-PCA gives the green one.*

Some results for this test case are shown in Figure 3.15. Again, even if the geometry is really simple, PCA-based methods fail to find the optimal bounding box, as do the all-pairs heuristic. Variants of Korsawe's method trying to build a bounding box aligned with faces of the convex hull are also bound to fail: as for the previous case, the tetrahedron is a geometry whose optimal bounding box is only flush with edges of said convex hull. The only methods that manage to find the optimal bounding box are O'Rourke's, HYBBRID and one of the variants of Korsawe's method. Note that this example illustrates the singularity of the PCA methods, which arises when the multiplicity of one of the eigenvalues of the covariance matrix is greater than 1. Then, the corresponding eigenvectors are not well-defined. In this case, this is due to the symmetry of the dataset.

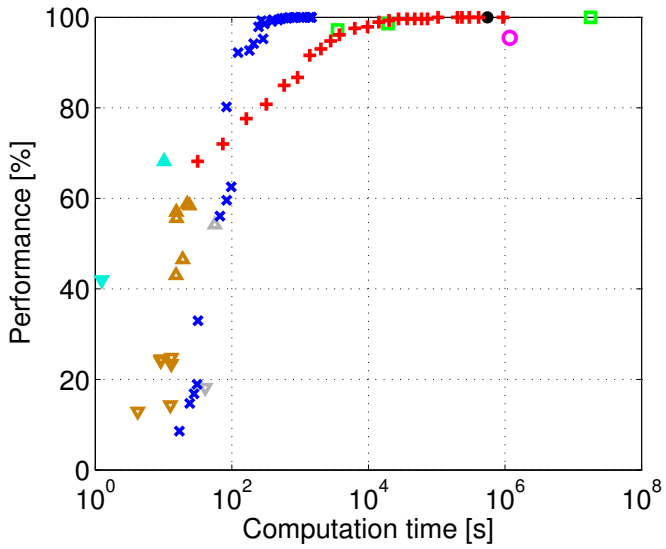
Results on the complete test set

All the methods have been tested on the whole set of test cases; the obtained results are shown in Figure 3.16 for three accuracy levels. This figure shows the computation time and the proportion of runs where the optimal volume has been obtained up to the given accuracy, as in Figure 3.11. Each method or variant is represented by a particular point in this time-performance diagram. In fact, Figure 3.11 corresponds to a close view of Figure 3.16 without the logarithmic scale and displaying only the points corresponding to HYBBRID.

Barequet and Har-Peled's GridSearchMinVolBbx method has been tested with the following values for the grid size parameter: $d = 1$, $2, \ldots, 9, 10, 12, 14, \ldots, 28, 30, 35, 40, 45, 50, 60$. If an accurate result is required, then the success rate (as defined in Section 3.5.1) of the method stalls at about 40% of the test cases for these values of d. Much higher values of d are thus needed in order to obtain an accurate result for most examples. Indeed, as $d \in \Theta(\frac{1}{\varepsilon})$, in order to improve the guaranteed accuracy by a factor k, one needs to increase the grid size d by a factor k. For instance, in order to reach a guaranteed accuracy of 10^{-8} instead of 10^{-3}, the value of d needs to be increased by a factor 10^5. However, if only a rough approximation such as 10^{-3} or less is required, this range of values of d may provide satisfactory results, as shown in Figure 3.16. Note that to the best of our knowledge, Barequet and Har-Peled's algorithm was the only method providing such a trade-off between accuracy and computation time. Moreover, it appears that the values $d = 1$ and 2

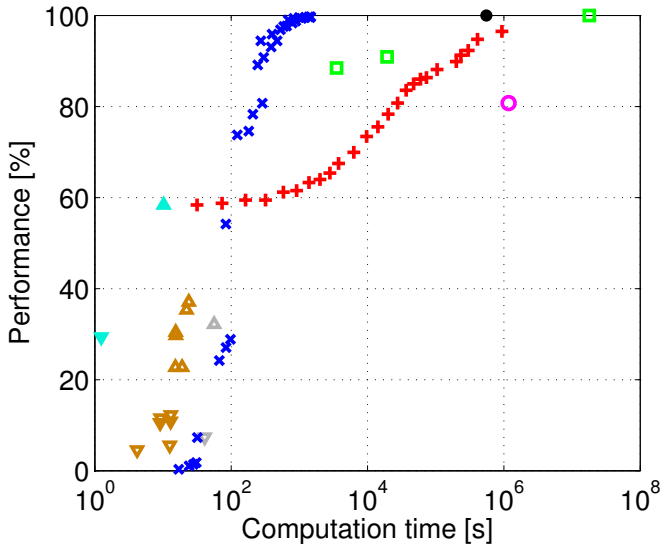


(a) Legend

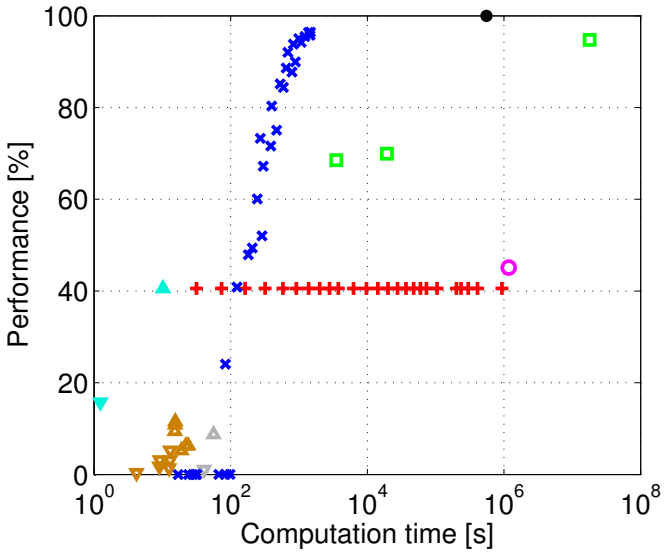


(b) Tolerance = 10^{-2}

Figure 3.16: *Performance and computation time of all the algorithms. Computer specifications are identical to what is described for Table 3.2. The computation time corresponds to the total time required to approximate an optimal OBB for each object among the 300 test cases, including the computation of the convex hulls if it is done by the algorithm. For HYBBRID, the volume of the OBB and the computation time of each test case have been obtained by running the algorithm 200 times and averaging the results.*



(c) Tolerance = 10^{-3}



(d) Tolerance = 10^{-6}

Figure 3.16: *(continued).*

yield smaller computation times than all other methods for their range of performance.

Another interesting observation is that the threshold value of 40% corresponds to the performance obtained by using an AABB on which the post-processing step is applied. This can be explained by the symmetry and the natural definition of some objects in the test set, which are usually oriented in a way that is usual for human beings, and thus aligned with the principal axes. As such, for these objects, the AABB has a common axis with an optimal OBB. As GridSearchMinVolBbx is based on the AABB for the diameter approximation, the symmetry and the definition of the set of points (more precisely their ordering) also imply that the orientation of the grid has an axis aligned with one axis of an optimal OBB. Hence, solving the associated 2D problem will return an optimal solution.

As far as Korsawe's method is concerned, the two faster variants are either faster or more reliable than the other methods, except when compared to the HYBBRID algorithm. The slower variant demonstrates very good performances, but unfortunately its computation time is much too large for practical purposes. The method is in fact even slower than O'Rourke's algorithm even if the latter has a guaranteed performance of 100%. Another method enumerating a large set of directions is the all-pairs method. Unfortunately, its performance is dominated by Korsawe's algorithm, i.e., the latter appears to be both faster *and* more reliable than all-pairs.

As expected, PCA-based methods are very fast but provide solutions with a very limited accuracy. The quality may even be worse than the result obtained by an axis-aligned bounding box. Note that All-PCA, Min-PCA and Max-PCA are each represented by four points. Indeed, the obtained OBB may vary depending on if one applies the preprocessing step and/or the post-processing step, or none of them. In the case of continuous PCA, the preprocessing step is required by the algorithm; hence, Gottschalk et al.'s method is only represented by two points.

It appears that HYBBRID is consistently the fastest method delivering the optimal volume on almost all test cases, with sufficiently large parameters. Faster methods present considerably lower performance levels. Some implementation details must also be taken into account while reviewing those results. Regarding the speed of the PCA-based methods, those only use a few matrix operations, most of which are written in

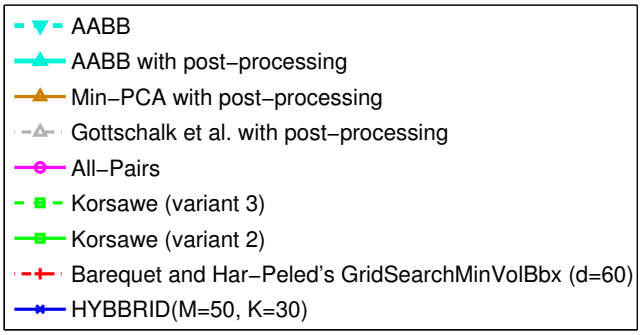
highly efficient C code in MATLAB®. On the other hand, the other algorithms such as HYBBRID and O'Rourke's mostly use MATLAB® code, which is slower. A well-written pure C implementation of those methods should reduce the execution time, making them better suited for real-life scenarios.

Another point of interest is the distribution of the error compared to the optimal solution over all runs for all test cases. This information is shown in Figure 3.17 for a selected subset of algorithms. To each method correspond at least one curve in the figure obtained with the variants yielding the best performance, independently of the computation time. Of course, O'Rourke's algorithm is not represented on this figure since it is optimal and thus, has a failure rate of 0%. A closer view with a logarithmic scale is also included in order to emphasize the exact performance of HYBBRID.

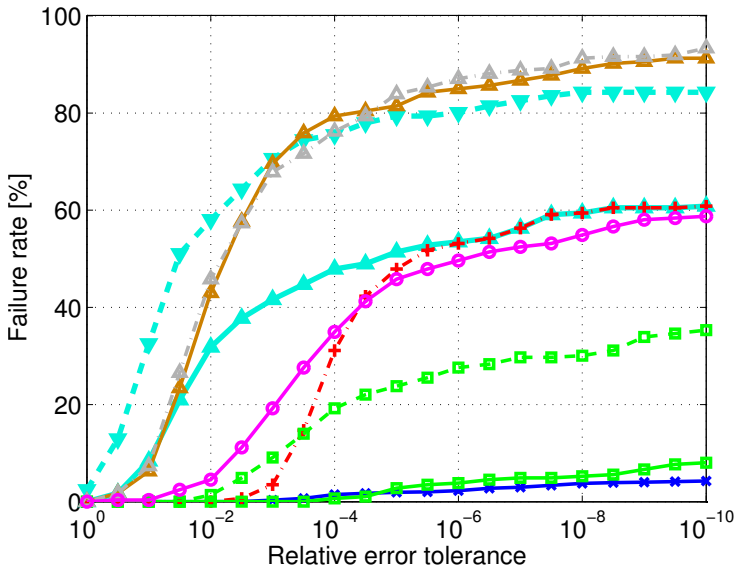
Note that the information about the failure rate for relative error tolerances of 10^{-2}, 10^{-3} and 10^{-8} was already contained in Figure 3.16. However, it is also interesting to see the evolution of the failure rate with the tolerance between and outside those values. For instance, this allows one to show that a failure rate of 50% can be achieved with Min-PCA for smaller tolerances than with AABB without post-processing. Nevertheless, the latter has a slightly smaller failure rate for tolerances tending to 0, even though both methods are clearly unsuitable for such tolerance levels. It also appears that AABB with post-processing has a failure rate equivalent to that of Min-PCA for relative error tolerances bigger than 0.05, but is 30% more reliable for very small tolerances.

A similar trade-off between the performances for small and large tolerances is observed for Barequet and Har-Peled's GridSearchMinVolBbx method (which is shown in Figure 3.17 with parameter $d = 60$) with respect to the third variant of Korsawe's method and all-pairs. For instance, GridSearchMinVolBbx is more reliable than Korsawe's method (resp. all-pairs) for relative error tolerances larger than about 3×10^{-4} (resp. 4×10^{-5}) but then, the failure rate increases and becomes worse for smaller tolerances. Indeed, the reliability of the method is then roughly equivalent to that of AABB with post-processing for tolerances smaller than 10^{-6}. Of course, the value of the tolerance where GridSearchMinVolBbx becomes equivalent to AABB with post-processing approaches 0 as d increases.

The behaviors of HYBBRID and the second, slower variant of Kor-

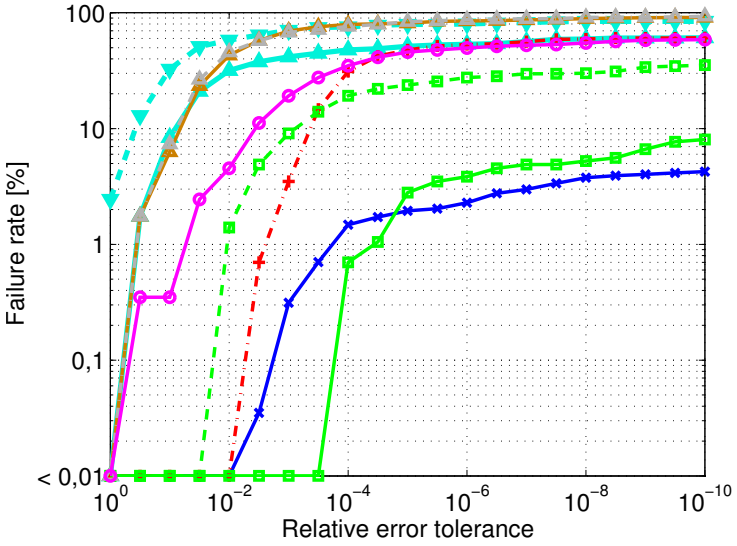


(a) Legend



(b) Linear view

Figure 3.17: *Failure rate of the different algorithms for several tolerance thresholds. The same results are displayed with a linear vertical axis in (b) and with a logarithmic vertical axis in (c). Computations have been done on the whole set of about 300 objects. For each test case and each threshold τ, the run is considered a failure if the relative error is greater than τ. For HYBBRID, the failure rate is based on 200 runs for each object.*

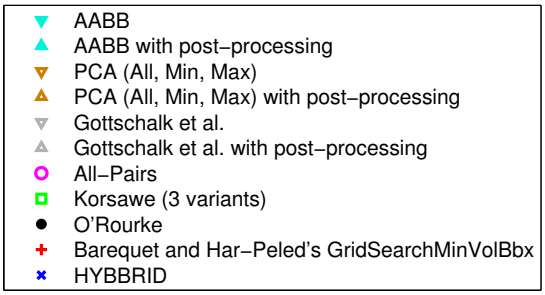


(c) Logarithmic view

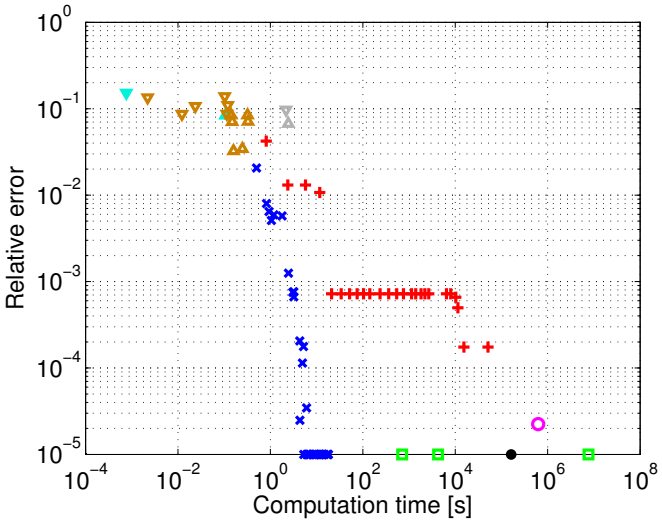
Figure 3.17: *(continued).*

sawe's method are better shown with a logarithmic scale. The thresholding of the failure rate at 0.01% is only due to the finite sampling issue, i.e., only about 300 test cases. It appears that HYBBRID is the most reliable method for tolerances smaller than 10^{-5} ; more precisely, its failure rate is about twice as small as the second variant of Korsawe's method. The fact that HYBBRID is slightly less reliable than the latter for larger tolerances can be nuanced by taking into account the computation time as Korsawe's method is about 10000 times slower than HYBBRID to complete the set of test cases.

It is also interesting to compare the efficiency of all these methods on the most complex test case, i.e., `globe9306` in Figure 3.9. The errors and computation times observed for this example are shown in Figure 3.18. Among the pairs of parameters of HYBBRID represented on this figure, the one yielding the slowest computation in average is $(M, K) = (50, 30)$ with a running time of less than 18 seconds. Conversely, O'Rourke's algorithm needs about 45 hours to find the optimal OBB, and the three variants of Korsawe's method have a computation time ranging from 12 minutes to about 90 days.



(a) Legend



(b) `globe9306`

Figure 3.18: *Accuracy and computation time of all the algorithms on* `globe9306`*, the example with the largest number of vertices on its convex hull. The computation time corresponds to the time required to approximate an optimal OBB, including the computation of the convex hull if it is done by the algorithm. For HYBBRID, the volume of the OBB and computation time of each test case have been obtained by running the algorithm 200 times and averaging the results. For the sake of readability, errors below* 10^{-5} *are displayed at this threshold value.*

3.6 Conclusion

In this chapter, HYBBRID, an algorithm to approximate the minimum-volume bounding box of a set of points, has been presented. It is a combination of two optimization components, namely the genetic algorithm and the Nelder-Mead algorithm. Combining those two methods leads to a method that shows a good convergence rate while still ensuring a good exploration of the search space in order to try to avoid local minima. The new idea was to use such a hybrid method on the rotation group $SO(3, \mathbb{R})$ to determine the orientation corresponding to the smallest enclosing box, or OBB, of a set of points.

The new method has been compared to currently used algorithms, and has been shown to be either much faster than the fastest exact algorithm (O'Rourke's) or more accurate than the fastest heuristics based on principal component analysis. HYBBRID was able to find the optimal volume for each of the test cases we tried it on, which shows the reliability of the method. In order to test this algorithm, a broad set of methods was implemented in MATLAB®. The whole codebase provides what we believe is a good framework to compare OBB fitting methods.

The scheme presented here could of course become the basis for further work. The same algorithm could be used to solve other problems on $SO(3, \mathbb{R})$, like for example, not finding the minimum-volume OBB, but the one with the smallest area. It could also be interesting to investigate if such hybrid combinations of optimization methods could be used for the computations of other bounding volumes such as spheres or k-DOPs with arbitrary normals. Moreover, extensions to dimensions higher than 3 may be considered in areas such as data mining.

Another possible extension is the case with several distinct optimal OBBs. It may be useful to be able to find all the optimal solutions and select the best one according to another criterion. Since nothing in HYBBRID prevents the simplices from converging to different local minima, we only have to analyze the populations and detect all the global solutions that have been found. The genetic algorithm should then be modified to help the simplices converge to different global minima, e.g., by introducing a repulsive component. As there would be no guarantee that *all* solutions have been found, it would also be important to modify the parameters such as the population size in order to increase the probability of detecting all global minima.

CHAPTER 4
The subset selection problem

In this chapter, we consider the *subset selection* problem, which consists in choosing a subset of columns of a matrix, according to some criterion to optimize. In many cases, we want the subset to be far from being rank deficient. In other terms, given an $m \times n$ matrix A and an integer k, we attempt to find the k "most linearly independent" columns of A in some sense. In our case, we will concentrate on finding a subset of k columns such that the *volume* of the parallelepiped spanned by these columns is maximized. For instance, the case with $k = 2$ can be interpreted as the selection of two vectors from a set of n vectors in $\mathbb{R}^m$, such that the area of the parallelogram spanned by the two selected vectors is maximized. We present an incremental algorithm for the subset selection problem, aiming at a low computational cost when k is small with respect to m and n. We also propose several variants, including a randomized version of the algorithm which combines a technique based on stochastic hill climbing, a pruning strategy and multiple passes. An article associated to this work is currently in preparation [CIBD12].

4.1 Introduction

We consider the problem of finding the k best columns of a real $m \times n$ matrix A, where the parameter k is fixed and given in advance. In many applications, k represents the numerical rank of A, which means that the k dominant singular values of A are much larger than the remaining singular values. We assume here that $k \leqslant \operatorname{rank}(A)$. In our work, the criterion for "best" is that the volume of the selected columns A_k has to be as large as possible. We define the volume of A_k as

$$\operatorname{vol}(A_k) := \det(A_k^T A_k)^{1/2} = \prod_{i=1}^{k} \sigma_i(A),$$

where $\sigma_i(A)$ denotes the i^{th} singular values of A.

Choosing k columns with maximal volume among n columns is clearly a combinatorial problem since there are $\binom{n}{k}$ possible choices. Çivril and Magdon-Ismail [cMI09] show that this problem is hard in the following sense:

Proposition 4.1 ([cMI09])**.** *Given a real matrix A with* $\operatorname{rank}(A) \geqslant k$ *where all columns have unit two-norm, and a number* $0 \leqslant \eta \leqslant 1$*, the problem of finding k columns A_k so that* $\operatorname{vol}(A_k) \geqslant \eta$ *is NP-hard.*

Moreover, unless $P = NP$, our subset selection problem (with the maximization of the volume) does not admit a polynomial-time approximation algorithm with an approximation factor of $\frac{2\sqrt{2}}{3} + \varepsilon$ [cMI09].

In general, subset selection consists thus in finding the most representative or most linearly independent columns of a matrix. This is very important in the area of data mining, where the selected columns typically correspond to a selection of the most relevant features in a particular database. In these applications, one needs to find the most representative features in order to reduce the dimension of the problem and drop redundant features. Of course, besides the maximization of the volume, other criteria can be used. Two others objectives are commonly used in subset selection problems:

(I) The maximization of the smallest singular value σ_k of A_k, the matrix formed by the k selected columns.

(II) The minimization of the residual $\|A - P_k A\|$, where P_k is the projection onto the space spanned by the columns of A_k. This may also be interpreted as minimizing $\min_X \|A_k X - \widetilde{A}_k\|$, where $\widetilde{A}_k$ is the matrix formed by the non-selected columns. This second formulation means that the best linear combination of A_k should be close to $\widetilde{A}_k$.

Note that maximizing the smallest singular value is not necessarily equivalent to maximizing the volume:

Example 4.2. *Let $k = 2$ and*

$$A = \begin{pmatrix} 3 & 1 & 1 \\ 2 & 2 & 0 \\ 4 & 3 & 2 \end{pmatrix}.$$

There are three subsets of size 2. The corresponding values for σ_2 and vol *are the following:*

Criterion	*Columns 1 and 2*	*Columns 1 and 3*	*Columns 2 and 3*
σ_2	1.036	0.849	1.086
vol	6.708	4.899	4.582

It appears that the two selected subsets are different.

Many algorithms have been proposed for different variants of the subset selection problem. In [DRVW06, DV06], Deshpande, Vempala et al. use a volume argument to derive Frobenius norm bounds to measure the accuracy of adaptively sampled low rank approximations. In [GE96], Gu and Eisenstat present a rank-revealing QR factorization algorithm that tries to find a good subset of columns, using the two criteria mentioned above ((I) and (II)). In [CI94], Chandrasekaran and Ipsen present a review of many existing strategies in an unified framework, for the same two criteria. They also propose hybrid algorithms for (I), (II) or both. Another approach is the backward greedy algorithm by Couvreur and Bresler [CB00] for the minimization of the residual. Their strategy consists in starting with the set of all columns of A and then, greedily removing columns until there are only k columns left.

Boutsidis, Mahoney and Drineas also proposed a randomized two-stage algorithm [BMD09] that tries to solve problem (II). Their algorithm first determines a subset of candidates by randomly selecting columns. The probabilities to select the different columns depend on estimations of their "quality". These scores are constructed by looking at the right singular vectors of A (see [BMD09] for the details). After this randomized stage, a deterministic subset selection procedure is done on the set of selected candidates. A deterministic version of this algorithm has also been proposed by Broadbent, Brown and Penner in [BBP10].

Let us look at the algorithm by Gu and Eisenstat. Even though the original goal is to solve problems (I) and (II), their algorithm is using the volume as decision criterion. Indeed, starting with an initial subset of k columns, the algorithm greedily modifies A_k until its volume cannot be increased by more than a factor $f \geqslant 1$ by exchanging a single column at a time. That is, the algorithm produces k columns A_k so that $\operatorname{vol}(\widetilde{A}_k) \leqslant f \operatorname{vol}(A_k)$ for all matrices $\widetilde{A}_k$ obtained by exchanging a single column of A_k with another column of A.

The singular values of the selected columns A_k can be bounded by the singular values σ_i of A as follows [GE96, Theorem 3.2]:

$$\frac{\sigma_i}{\sqrt{1 + f^2 k(n-k)}} \leqslant \sigma_i(A_k) \leqslant \sigma_i \qquad i = 1, \ldots, k.$$

The operation count of this algorithm is $\mathcal{O}\left((m + n \log_f n)n^2\right)$. The above singular value bounds imply the following for the volume of A_k:

$$\left(\frac{1}{1 + f^2 k(n-k)}\right)^k \leqslant \frac{\mathrm{vol}^2(A_k)}{\prod_{j=1}^k \sigma_j^2} \leqslant 1. \tag{4.1}$$

We propose a window-based technique for the subset selection problem. The main advantage of this approach is its speed: for an $m \times n$ matrix A with $k \ll m, n$, the running time is $\mathcal{O}\left(mk(n-k)\right)$. In contrast, Gu and Eisenstat's SRRQR algorithm [GE96] has complexity $\mathcal{O}(n^2 \max(m, n))$, provided that the tolerance f is chosen as a small power of n, e.g., $\sqrt{n}$. Its disadvantage is that it yields only a subset of given dimension k, whereas [GE96, Algorithm 5] proposes subsets of any order going from 1 to $\min(m, n)$. We also present a randomized variant of the algorithm which uses up to two different heuristics in order to reduce the computation time and/or to improve the volume of the selected columns.

In the following sections, we begin by presenting the key ideas of our windowed approach for subset selection. Section 4.3 presents different variants and improvements to the basic windowed approach, including our randomized algorithm. Implementation details and experimental results can be found in Section 4.4.

Notation

The identity matrix is denoted by I, the vector containing a single one at position i is e_i, all other entries being zeroes. $\|\cdot\|$ corresponds to the two (or Euclidean) norm. The singular values of the $m \times n$ matrix A are denoted by $\sigma_1(A) \geqslant \sigma_2(A) \geqslant \ldots \geqslant \sigma_{\min\{m,n\}}(A)$. By $v_{1:k}$ we denote the first k elements of the vector v. Similarly, $R_{1:k,1:k}$ denotes the upper-left $k \times k$ submatrix of the matrix R.

The *volume* of an $m \times n$ real matrix A with $m \geqslant n$ is defined as

$$\mathrm{vol}(A) := \det(A^T A)^{\frac{1}{2}} = \prod_{i=1}^n \sigma_i(A).$$

This corresponds to the usual volume of the parallelepiped spanned by the columns in A_k, as shown by the following proposition.

Proposition 4.3. *Let $a_1, \ldots, a_k \in \mathbb{R}^m$ be k vectors. Then, the k-volume of the parallelepiped $\mathcal{P}$ spanned by those vectors satisfies*

$$\mathrm{vol}(\mathcal{P})^2 = \det(A^T A),$$

where the $m \times k$ matrix A is defined by $A = \begin{pmatrix} a_1 & a_2 & \cdots & a_k \end{pmatrix}$.

Proof. We will prove this result by induction on k. Let us look at the base case $k = 1$. The matrix $A^T A$ has size 1×1, and its unique entry is equal to $a_1^T a_1 = \|a_1\|^2$, which is indeed the square of the length (1-volume) of the vector.

Now, suppose that the equality holds for any set of $k-1$ vectors. Let us consider the set $a_1, a_2, \ldots, a_k$. We may decompose a_k as $a_k = h + b$, where h is orthogonal to $a_1, \ldots, a_{k-1}$ and b is in the span of $a_1, \ldots, a_{k-1}$. Let

$$\begin{aligned} A &= \begin{pmatrix} a_1 & a_2 & \cdots & a_{k-1} & a_k \end{pmatrix}, \\ \widetilde{A} &= \begin{pmatrix} a_1 & a_2 & \cdots & a_{k-1} & h \end{pmatrix}, \\ B &= \begin{pmatrix} a_1 & a_2 & \cdots & a_{k-1} \end{pmatrix}. \end{aligned}$$

Geometrically, B is thus the base of the parallelepiped $\mathcal{P}$, and h is the corresponding height. Since $b = a_k - h$ is in the span of $a_1, \ldots, a_{k-1}$, we have $\det(A^T A) = \det(\widetilde{A}^T \widetilde{A})$. Furthermore,

$$\widetilde{A}^T \widetilde{A} = \begin{pmatrix} B^T \\ h^T \end{pmatrix} \begin{pmatrix} B & h \end{pmatrix} = \begin{pmatrix} B^T B & B^T h \\ h^T B & h^T h \end{pmatrix} = \begin{pmatrix} B^T B & 0 \\ 0 & h^T h \end{pmatrix}$$

since h is orthogonal to all columns in B. Hence, we have $\det(\widetilde{A}^T \widetilde{A}) = \|h\|^2 \det(B^T B)$. By the induction hypothesis, $\det(B^T B)$ corresponds to the $(k-1)$-volume of the base of $\mathcal{P}$ and $\det(\widetilde{A}^T \widetilde{A})$ corresponds thus to the k-volume of $\mathcal{P}$. □

4.2 The Windowed Subset Selection algorithm (WSS)

We present a polynomial-time algorithm that can be viewed as a "windowed" approach to the subset selection problem (see Algorithm 4.1). At each step, we select k columns of maximal volume from a window of $k+1$ columns. The single unproductive column in the window is then

replaced with the next column waiting outside the window. If the matrix has n columns then there are always $n-k-1$ columns waiting outside the window, so that after $n-k-1$ steps each column of A will have been considered for inclusion in the window.

Algorithm 4.1 Windowed Subset Selection (WSS)

Input: $m \times n$ matrix $A = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix}$
integer k with $k \leqslant \operatorname{rank}(A)$ and $k < n$
Output: matrix A_k with k columns of A so that $\operatorname{vol}(A_k)$ is "large"

```
A_k := (a_1 ... a_k)
for i = k+1 to n do
    W := (A_k  a_i)
    Determine permutation matrix P_i so that W P_i = (Â_k  â_{k+1})
    where Â_k has maximal volume among all sets of k columns of W
    if vol(Â_k) > vol(A_k) then
        A_k := Â_k
    end if
end for
```

In the next section we describe how to implement an iteration of the for loop in Algorithm 4.1. That is, we consider the problem of finding k columns of maximal volume amid a set of $k+1$ columns.

4.2.1 Throwing out one column

We consider the problem of removing one column from a set of $k+1$ columns so that the remaining k columns have maximal volume. There are $k+1$ sets of columns to choose from.

Specifically, let $A = \begin{pmatrix} a_1 & \cdots & a_{k+1} \end{pmatrix}$ be an $m \times (k+1)$ matrix with $m \geqslant k+1$, and let

$$A^{(j)} = \begin{pmatrix} a_1 & \cdots & a_{j-1} & a_{j+1} & \cdots & a_{k+1} \end{pmatrix}, \qquad 1 \leqslant j \leqslant k+1,$$

be the $m \times k$ matrix without column j. We want to determine which column l to remove so that

$$\operatorname{vol}(A^{(l)}) = \max_{1 \leqslant j \leqslant k+1} \operatorname{vol}(A^{(j)}).$$

The solution is unique if there exists a set of columns $A^{(l)}$ with

$$\mathrm{vol}(A^{(l)}) > \max_{j \neq l} \mathrm{vol}(A^{(j)}).$$

How large can $\mathrm{vol}(A^{(l)})$ be? The volume of *any* submatrix $A^{(j)}$ is bounded by the singular values $\sigma_1 \geqslant \ldots \geqslant \sigma_k$ of A, because the interlacing theorem for singular values:

Proposition 4.4 ([GvL96])**.** *Let A be an $m \times n$ matrix with singular values $\sigma_1 \geqslant \ldots \geqslant \sigma_{\min(m,n)}$. Let B be an $m \times (n-1)$ matrix obtained by removing one column from A, with singular values $\tau_1 \geqslant \ldots \geqslant \tau_{\min(m,n-1)}$. Then, for all $i \geqslant 1$,*

$$\sigma_i \geqslant \tau_i \geqslant \sigma_{i+1}.$$

Here, this result implies that

$$\mathrm{vol}(A^{(j)}) \leqslant \prod_{i=1}^{k} \sigma_i, \qquad 1 \leqslant j \leqslant k+1.$$

This bound can be achieved with equality if the columns of A can be permuted so that the right singular vector matrix has the form

$$V = \begin{pmatrix} V_{11} & 0 \\ 0 & 1 \end{pmatrix},$$

where V_{11} is $k \times k$ real orthogonal, and the $(k+1)^{\mathrm{st}}$ column of V is associated with σ_{k+1}. In this case, we have $\mathrm{vol}(A^{(l)}) = \prod_{i=1}^{k} \sigma_i$.

The obvious way of determining k columns of maximal volume would be a *brute-force approach* where we inspect the volume of all $k+1$ possible matrices $A^{(j)}$, $1 \leqslant j \leqslant k+1$. However, we do not need to explicitly compute $k+1$ determinants. Indeed, the Gu-Eisenstat Criterion in Section 4.2.2 allows us to find the optimal set of columns at a lower computational cost.

4.2.2 The Gu-Eisenstat Criterion

The Strong Rank Revealing QR (SRRQR) algorithm by Gu and Eisenstat [GE96, Algorithm 4] takes as input an $m \times n$ matrix A with $m \geqslant n$, and aims at determining a (partial) QR decomposition $AP = QR$, where P is a permutation matrix, Q is $m \times n$ with orthonormal columns, and R

is $n \times n$ with $R_{1:k,1:k}$ upper triangular. Once an initial factorization with $P = I$ has been determined, the algorithm works on the matrix R and repeatedly exchanges one pair of columns at a time until the determinant of the leading $k \times k$ principal submatrix of R does not increase anymore.

For the special case of selecting k columns from $k+1$ columns, we present in Algorithm 4.2 below a conceptual version of the SRRQR algorithm with $f = 1$. This algorithm determines a permutation that puts columns of maximal volume in positions $1, \ldots, k$, and the doomed column in position $k+1$. We denote by $A^{(j)}$ the matrix AP with column j removed.

Algorithm 4.2 Gu-Eisenstat Criterion

Input: $m \times (k+1)$ matrix $A = \begin{pmatrix} a_1 & \cdots & a_{k+1} \end{pmatrix}$ with $m \geqslant k+1$
Output: Permutation matrix P_2 so that the leading k columns of AP_2 have maximal volume among all sets of k columns of A

1: $A^{(k+1)} := \begin{pmatrix} a_1 & \cdots & a_k \end{pmatrix}$
2: $P_2 := I$
3: **while** there exists a j with $\mathrm{vol}(A^{(j)}) > \mathrm{vol}(A^{(k+1)})$ **do**
4: $\quad A := AP_{j,k+1}$, where the permutation matrix $P_{j,k+1}$ permutes columns j and $k+1$ of A
5: $\quad P_2 := P_2 P_{j,k+1}$
6: **end while**

Theorem 4.5. *Let P_2 be the permutation matrix produced by Algorithm 4.2 and denote by A_k the leading k columns of AP_2. Then* $\mathrm{vol}(A_k)$ *is maximal among all sets of k columns of A, and*

$$\left(\frac{1}{k+1}\right)^k \leqslant \frac{\mathrm{vol}^2(A_k)}{\prod_{j=1}^k \sigma_j^2} \leqslant 1.$$

Proof. This follows from Equation (4.1) with $f = 1$ and $n = k+1$. □

There are several ways to implement the idea of Gu and Eisenstat and, in particular, the special case in Algorithm 4.2. For example, if there are several j's such that $\mathrm{vol}(A^{(j)}) > \mathrm{vol}(A^{(k+1)})$, which one should we choose? In our implementation of SRRQR [GE96, Algorithm 4], we have chosen to select the j such that $\mathrm{vol}(A^{(j)})$ is maximized (see Algorithm 4.3). This is thus a greedy strategy. In the following, this

Algorithm 4.3 (Greedy) Strong Rank Revealing QR (SRRQR)

Input: $m \times n$ matrix $A = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix}$
integer k with $k \leqslant \operatorname{rank}(A)$ and $k < n$
Output: matrix A_k with k columns of A so that $\operatorname{vol}(A_k)$ is "large"
1: $A_k := \begin{pmatrix} a_1 & \cdots & a_k \end{pmatrix}$
2: **repeat**
3: Determine a permutation matrix P such that the first k columns of AP are A_k
4: Define $\widetilde{a}_{k+1}, \ldots, \widetilde{a}_n$ so that $AP = \begin{pmatrix} A_k & \widetilde{a}_{k+1} & \cdots & \widetilde{a}_n \end{pmatrix}$
5: $\{j_{\max}, i_{\max}\} := \arg\max_{\substack{1 \leqslant j \leqslant k \\ k+1 \leqslant i \leqslant n}} \operatorname{vol}\begin{pmatrix} A_k^{(j)} & \widetilde{a}_i \end{pmatrix}$
6: $B := \begin{pmatrix} A_k^{(j_{\max})} & \widetilde{a}_{i_{\max}} \end{pmatrix}$
7: **if** $\operatorname{vol}(B) > \operatorname{vol}(A_k)$ **then**
8: $A_k := B$
9: **end if**
10: **until** $\operatorname{vol}(B) \leqslant \operatorname{vol}(A_k)$

will be the version of the algorithm we are referring to when mentioning SRRQR.

Note that we are also using some greedy strategy in the windowed algorithm (Algorithm 4.1) since we are looking for the $\hat{A}_k$ with maximal volume at each iteration, instead of any $\hat{A}_k$ with a larger volume. In fact, repeating the windowed approach as long as it is possible to increase the volume could also be interpreted as an implementation of the Gu-Eisenstat criterion, since we would effectively be doing column exchanges until we reach a local maximum for the volume. This corresponds to the multipass algorithm presented in Algorithm 4.4 (Section 4.3). However, considering one single column at a time instead of allowing any exchange to occur at each iteration will yield interesting benefits (see results in Sections 4.4.3 and 4.4.4).

4.2.3 Selection of the doomed column

The while loop in Algorithm 4.2 terminates after one single iteration if j is adequately chosen. An explicit expression for the factor of increase in volume when two columns are permuted can be computed. Hence, one simply has to choose j such that this factor is maximal and larger

than 1. If no such j exists, then the leading k columns of A already have maximal volume and there is no permutation to be done.

Let us consider a permutation between columns k and $k+1$ for matrix A at a given iteration. Let

$$\begin{aligned} A &= \begin{pmatrix} A_{k-1} & a_k & a_{k+1} \end{pmatrix}, \\ A^{(k+1)} &= \begin{pmatrix} A_{k-1} & a_k \end{pmatrix}, \\ A^{(k)} &= \begin{pmatrix} A_{k-1} & a_{k+1} \end{pmatrix}, \end{aligned}$$

where A_{k-1} has $k-1$ columns.

Algorithm 4.2 permutes columns k and $k+1$ of A if

$$\frac{\mathrm{vol}(A^{(k)})}{\mathrm{vol}(A^{(k+1)})} > 1.$$

Gu and Eisenstat express this ratio in terms of a QR decomposition,

$$A = \begin{pmatrix} A_k & a_{k+1} \end{pmatrix} = Q \begin{pmatrix} R_{11} & r_{1:k,k+1} \\ & r_{k+1,k+1} \end{pmatrix}, \tag{4.2}$$

where Q is $m \times (k+1)$ with orthonormal columns, and R_{11} is a $k \times k$ upper triangular matrix. Note that the $(k+1) \times (k+1)$ upper triangular matrix in Equation (4.2) has the same singular values $\sigma_1 \geqslant \ldots \geqslant \sigma_{k+1}$ as A.

The ratio of volumes can be expressed (see [GE96, Lemma 3.1]) as

$$\frac{\mathrm{vol}^2(A^{(k)})}{\mathrm{vol}^2(A^{(k+1)})} = \left|(R_{11}^{-1} r_{1:k,k+1})_k\right|^2 + \|e_k^T R_{11}^{-1}\|^2 |r_{k+1,k+1}|^2.$$

In the general case, the factor of increase when swapping out column j for column $k+1$ is given by

$$\nu(j) := \frac{\mathrm{vol}(A^{(j)})}{\mathrm{vol}(A^{(k+1)})} = \sqrt{\left|(R_{11}^{-1} r_{1:k,k+1})_j\right|^2 + \|e_j^T R_{11}^{-1}\|^2 |r_{k+1,k+1}|^2}. \tag{4.3}$$

Hence, Algorithm 4.2 permutes columns j and $k+1$ of A if

$$\left|(R_{11}^{-1} r_{1:k,k+1})_j\right|^2 + \|e_j^T R_{11}^{-1}\|^2 |r_{k+1,k+1}|^2 > 1. \tag{4.4}$$

By choosing j such that this expression is maximal and larger than 1, no other permutation will be able to increase the volume further.

Once the volume of the leading k columns cannot increase anymore, Algorithm 4.2 guarantees that

$$\|R_{11}^{-1} r_{1:k,k+1}\|_\infty \leqslant 1,$$

otherwise the condition (4.4) is clearly not satisfied for some j.

4.3 Improving the performance of the windowed algorithm

The Windowed Subset Selection algorithm presented in the previous section has a running time of $\mathcal{O}(mk(n-k))$ as it uses an $m \times k$ window which slides $n-k$ times (see Section 4.4.1 for more details). For sufficiently small values of k, the algorithm is expected to find a good subset of columns in a short amount of time. That is, the volume of the columns it selects is expected to be approximately equivalent to the volume returned by the (non-windowed) SRRQR algorithm by Gu and Eisenstat, while being obtained much faster. When k grows, the performance of Algorithm 4.1 drops: since each column is only examined once, the final subset depends heavily on the ordering of the columns of the matrix. Several modifications can be done in order to increase the reliability of the method.

4.3.1 Multi-pass Windowed Subset Selection (MWSS)

An obvious way to improve the performance is to run Algorithm 4.1 several times while starting each iteration with the previous selected columns as initial window. This idea has been used in [BGD12] in the context of incremental methods for computing dominant singular subspaces.

The number of passes can be fixed if computation time is an issue, or we can keep sweeping the columns of A until no more change can be made without decreasing the volume. The second choice corresponds to Algorithm 4.4 presented below.

The index $j_{\max}$ at line 7 of Algorithm 4.4 may be found using expression (4.3).

Unlike Algorithm 4.1, this multi-pass version guarantees that the solution A_k is "locally optimal" in the sense that the volume of A_k cannot

Algorithm 4.4 Multi-pass Windowed Subset Selection (MWSS)

Input: $m \times n$ matrix $A = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix}$
integer k with $k \leqslant \operatorname{rank}(A)$ and $k < n$
Output: matrix A_k with k columns of A so that $\operatorname{vol}(A_k)$ is "large"

$A_k := \begin{pmatrix} a_1 & \cdots & a_k \end{pmatrix}$
unchanged := 0
repeat
  Determine a permutation matrix P such that the first k columns of AP are A_k, and the ordering of the other columns is preserved
  Define $\widetilde{a}_{k+1}, \ldots, \widetilde{a}_n$ so that $AP = \begin{pmatrix} A_k & \widetilde{a}_{k+1} & \cdots & \widetilde{a}_n \end{pmatrix}$
  for $i = k+1$ to n **do**
    $j_{\max} := \arg\max_{1 \leqslant j \leqslant k} \operatorname{vol}\left(\begin{pmatrix} A_k^{(j)} & \widetilde{a}_i \end{pmatrix}\right)$
    $B := \left(\begin{pmatrix} A_k^{(j_{\max})} & \widetilde{a}_i \end{pmatrix}\right)$
    if $\operatorname{vol}(B) > \operatorname{vol}(A_k)$ **then**
      $A_k := B$
      unchanged := 0
    else
      unchanged := unchanged + 1
    end if
  end for
until unchanged $\geqslant n - k$

be increased by exchanging *one* column of A_k with another column of A. This is the same stopping criterion as in [GE96]. Experimental results show that the volumes obtained by Algorithm 4.4 are as good as those obtained by the SRRQR algorithm in most cases, but the set of selected columns may differ.

Two heuristic approaches may be used in order to improve the performance of the Multi-pass Windowed Subset Selection algorithm. With the first one, we try to find a set of columns with a larger volume (see Section 4.3.2), whereas the second one aims at decreasing the computation time of the method (see Section 4.3.3). The two techniques can also be used at the same time so that the additional computational cost of the first strategy is more or less counterbalanced by the acceleration due to the second heuristic.

4.3.2 Non-greedy randomized variant

This small modification to Algorithm 4.4 aims at finding a set of columns with a larger volume. As shown previously, it is possible to compute the increase in volume when swapping out column j for column $k+1$, using expression (4.3). At each iteration, Algorithm 4.4 chooses the column j that maximizes this increase, if any exists. Since this greedy approach does not have to produce the best set of columns at the end of the algorithm, one possible variant would be to use a simulated annealing approach (see Section 2.2.1).

In the general case, instead of only allowing the best column to be chosen, one could perform a locally suboptimal operation, possibly even taking into consideration exchanges that decrease the volume. The column to be thrown out would be randomly chosen, "better" choices having a larger probability. Usually, as the number of iterations increases, the probabilities are modified such that the "better" choices are more and more favored. At the end, only the best exchanges are accepted.

A disadvantage of simulated annealing-based techniques is that it may sometimes require a large number of iterations before convergence to a solution. Furthermore, one has to choose the probability function, and how it evolves as the computation proceeds. As we want to keep a small number of iterations, we propose a simulated annealing-like variant with restricted choices at each step: when exchanging a column of A_k with the new column $\widetilde{a}_i$, instead of choosing the column $j_{\max}$ with the

largest increase of volume, all columns j that *increase* the volume are considered. Thus, the strategy does not allow permutations that decrease the volume.

Then, a single column is chosen among all eligible columns, using relative probabilities that only depend on the factors of increase in volume. For example, if $j_1, \ldots, j_p$ are the columns such that $\nu(j_1), \ldots, \nu(j_p) > 1$, then each column j_i may be taken with probability proportional to $\nu(j_i)^\gamma - 1$ for some fixed power γ.

This approach is close to the *stochastic hill climbing* method as it randomly chooses an improving move. However, in our case, the quality of *all* possible moves are evaluated and the probability of selecting a move depends on its relative quality when compared to the other possibilities.

Finally, it may be useful to run the full randomized algorithm several times and keep the best set of columns. The corresponding volume is typically equivalent or slightly better than the solution found by Algorithm 4.4. Since running the algorithm several times increases the computation time, it may be interesting to combine the stochastic hill climbing approach with the pruning heuristic presented below.

4.3.3 Pruning heuristic for the set of columns

The two variants described above improve the reliability of the algorithm, but also increase the expected computation time by increasing the number of times the window sweeps through the columns. Since the original objective of the windowed approach is speed (for $k \ll m, n$), it may be wise to use a heuristic to reduce the computational cost. One possible way to achieve this is to try to remove "bad" columns from the matrix. Indeed, experimental results tend to show that in most cases, only a small number of exchanges are done at each pass and moreover, most columns are never included in the candidate set.

In order to reduce the computation time, we propose to remove columns that would decrease the volume by a large factor when included in the current window, whatever column we swap out, i.e., columns such that $\nu(j) < \tau$ for all j and a given tolerance $\tau < 1$. Of course, this strategy may decrease the quality of the solution returned by the algorithm, but numerical experiments (see Section 4.4.3) show that in several cases, it is possible to obtain a significant improvement in computation time without sacrificing reliability.

The Randomized Multi-pass Windowed Subset Selection (RMWSS) algorithm which includes both heuristics is presented below in Algorithm 4.5.

The parameter R corresponds to the total number of trials. We recall that in lines 12 and 17, $\nu(j)$ is defined as

$$\nu(j) := \frac{\mathrm{vol}\left(A_k^{(j)} \quad \widetilde{a}_i\right)}{\mathrm{vol}(A_k)}$$

and can be computed using Equation (4.3). Note that each *pass* in Algorithm 4 involves at most the same number of columns as the previous one, and this holds even across different *trials*. This is a design choice in order to speed up the algorithm.

Another possibility would be to prune the columns for a limited "amount of time". This strategy can be viewed as a kind of tabu search (see Section 2.2.2). The notion of "amount of time" can be interpreted in different ways, e.g., a number of column exchanges, of iterations for the sliding window, of passes or even of trials. In this case, we have decided *not* to choose this approach for two reasons. On the one hand, preliminary results from numerical experiments have shown that pruned columns tend to remain outside the window when the pruning heuristic is disabled. On the other hand, the algorithm would be slower because of the additional cost due to the management of such a tabu list and the fact that the number of eligible columns would be larger.

4.4 Numerical experiments

In this section, we compare the performance of our three algorithms — Windowed Subset Selection (WSS), Multi-pass Windowed Subset Selection (MWSS) and Randomized Multi-pass Windowed Subset Selection (RMWSS). As reference for the comparison, we use the Strong Rank Revealing QR algorithm (SRRQR) by Gu and Eisenstat [GE96, Algorithm 4], with a tolerance parameter $f = 1$ in Equation (4.1).

All four algorithms (WSS, MWSS, RMWSS, SRRQR) have been implemented using MATLAB® 7.4.0.336 (R2007a) and computations have been performed on an Intel® Core™ 2 Quad Q9300 at 2.50 GHz with 3 GiB RAM, running Ubuntu Linux 10.04 LTS. Our implementation of

Algorithm 4.5 (Non-greedy) Randomized Multi-pass Windowed Subset Selection (RMWSS)

Input: $m \times n$ matrix $A = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix}$
integer k with $k \leqslant \operatorname{rank}(A)$ and $k < n$
integer R with $R \geqslant 1$
parameter $\gamma > 0$
tolerance τ with $0 < \tau < 1$

Output: matrix A_k with k columns of A so that $\operatorname{vol}(A_k)$ is "large"

1: $\hat{A} := A$
2: **for** $r = 1$ to R **do**
3: $A_k := \begin{pmatrix} a_{j_1} & \cdots & a_{j_k} \end{pmatrix}$ where the columns of A_k are k distinct, randomly selected columns of $\hat{A}$.
4: unchanged := 0
5: **repeat**
6: Determine a permutation matrix P such that the first k columns of $\hat{A}P$ are A_k, and the ordering of the other columns is preserved
7: Define $p, \widetilde{a}_{k+1}, \ldots, \widetilde{a}_p$ so that $\hat{A}P = \begin{pmatrix} A_k & \widetilde{a}_{k+1} & \cdots & \widetilde{a}_p \end{pmatrix}$
8: **for** $i = k+1$ to p **do**
9: $W := \begin{pmatrix} A_k & \widetilde{a}_i \end{pmatrix}$
10: $\mathcal{J} := \left\{ j \mid \operatorname{vol}\begin{pmatrix} A_k^{(j)} & \widetilde{a}_i \end{pmatrix} > \operatorname{vol}(A_k), 1 \leqslant j \leqslant k \right\}$
11: **if** $\mathcal{J} \neq \emptyset$ **then**
12: Select an index j_{sel} from the set $\mathcal{J}$ where each index $j_c \in \mathcal{J}$ can be selected with a probability equal to $(\nu(j_c)^\gamma - 1) \Big/ \left(\sum_{j \in \mathcal{J}} (\nu(j)^\gamma - 1) \right)$
13: $A_k := \begin{pmatrix} A_k^{(j_{\text{sel}})} & \widetilde{a}_i \end{pmatrix}$
14: unchanged := 0
15: **else**
16: unchanged := unchanged + 1
17: **if** $\max_{1 \leqslant j \leqslant k} \nu(j) < \tau$ **then**
18: Remove column $\widetilde{a}_i$ from $\hat{A}$
19: **end if**
20: **end if**
21: **end for**
22: **until** unchanged $\geqslant p - k$
23: **end for**

SRRQR uses the updating formulas given in [GE96, Section 4], reducing the running time of SRRQR.

In the following sections, we begin by explaining how our algorithms have been implemented. Then, we present the test matrices used in the experiments. Finally, Section 4.4.3 contains the numerical results and the comparison between the four algorithms.

4.4.1 Implementing the Windowed Subset Selection algorithm

The crucial step in the algorithm is the evaluation of all the $\nu(j)$'s, with $j = 1, \ldots, k$. This operation has to be done in $\mathcal{O}(mk)$ time in order to have a total time complexity of $\mathcal{O}(mk(n-k))$. Let us recall that $\nu(j)$ is defined by

$$\nu(j) = \sqrt{\left|(R_{11}^{-1} r_{1:k,k+1})_j\right|^2 + \|e_j^T R_{11}^{-1}\|^2 |r_{k+1,k+1}|^2},$$

with the window

$$W = QR = \begin{pmatrix} Q_1 & q_{k+1} \end{pmatrix} \begin{pmatrix} R_{11} & r_{1:k,k+1} \\ & r_{k+1,k+1} \end{pmatrix},$$

where W is $m \times (k+1)$, Q is $m \times (k+1)$ with orthonormal columns, Q_1 is $m \times k$ and R_{11} is $k \times k$ upper triangular.

The idea is to keep track of Q_1, the first k columns of Q, and R_{11}, i.e., the two factors associated to the current set of k columns, and update them every time we modify the subset. This is also done with R_{11}^{-1} as this matrix is required for the computation of $\nu(j)$. These three matrices can be initialized with a cost of $\mathcal{O}\left(mk^2\right)$ before the first iteration of the for loop. At each iteration, both vectors q_{k+1} and $r_{1:k+1,k+1}$ can be computed using a Gram-Schmidt step (including the "twice is enough" strategy) in $\mathcal{O}(mk)$ time.

When the algorithm exchanges columns j and $k+1$, we first move the doomed column j into position k by shifting columns in position $j+1, \ldots, k$ one step to the left, and putting column j in position k. This translates into a column permutation of R_{11} and a row permutation of R_{11}^{-1}. The matrix Q is unaffected by the operation.

Then, the new matrix R has to be triangularized again. This can be done by using $(k-j)$ Givens rotations $G_j, \ldots, G_{k-1}$, each Givens rotation

G_i being associated with column i and operating on rows i and $i+1$ of the matrix R. The corresponding updates for the two matrices Q and R_{11}^{-1} are of course G_i^T for $i = j, \ldots, k-1$, each rotation operating on columns i and $i+1$. The total cost for these operations is $\mathcal{O}\left(m(k-j)\right) \subset \mathcal{O}\left(mk\right)$.

After these manipulations, we swap the two columns in position k and $k+1$, retriangularize matrix R using a Givens rotation G_k acting on rows k and $k+1$, and update matrix Q using G_k^T. Partitioning R_{11}^{-1} as

$$R_{11}^{-1} = \begin{pmatrix} \widetilde{R} & \widetilde{r}_{1:k-1} \\ & \widetilde{r}_k \end{pmatrix},$$

where $\widetilde{R}$ is $(k-1)\times(k-1)$ upper triangular, the new expression for R_{11}^{-1} is then given by

$$R_{11,\text{new}}^{-1} = \begin{pmatrix} \widetilde{R} & -\left(\widetilde{R}\bar{r}_{1:k-1}\right)\Big/\bar{r}_k \\ & 1/\bar{r}_k \end{pmatrix},$$

where $\bar{r} = (\bar{r}_1, \ldots, \bar{r}_k)^T$ corresponds to the k^{th} column of $R_{11,\text{new}}$, the new value of R_{11}. Indeed, we have

$$\begin{pmatrix} \widetilde{R} & -\left(\widetilde{R}\bar{r}_{1:k-1}\right)\Big/\bar{r}_k \\ & 1/\bar{r}_k \end{pmatrix} \begin{pmatrix} \bar{r}_{1:k-1} \\ \bar{r}_k \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Remark 4.6. *Updating the matrix R_{11}^{-1} may not always be a good idea. If the problem is ill-conditioned, significant numerical imprecisions may arise, for example when the current subset of columns has near-zero volume and swapping in the new column results in a huge factor of increase. This can be detected by checking the value of the $\nu(j)$'s. Due to the nature of the algorithm, this is expected to happen only if the starting subset is nearly linearly dependent such as with Kahan matrices or GKS matrices (see Section 4.4.2). In this case, the inverse of the triangular matrix R_{11} is recomputed instead of updated.*

4.4.2 The test matrices

The different algorithms have been tested on the following sets of matrices:

1. *Kahan matrices.* These are upper triangular matrices with all columns having unit two-norm. The $n \times n$ Kahan matrix K_n is given by $K_n = S_n T_n$, where

$$S_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \psi & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \psi^{n-1} \end{pmatrix} \quad \text{and} \quad T_n = \begin{pmatrix} 1 & -\phi & \cdots & -\phi \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\phi \\ 0 & \cdots & 0 & 1 \end{pmatrix},$$

   where ϕ and ψ satisfy $\phi^2 + \psi^2 = 1$, with $\phi, \psi > 0$. These matrices have been introduced by Kahan [Kah66] and are well-known test matrices for the subset selection problem. Indeed, many algorithms based on QR factorization with column pivoting do not perform any exchange and return a set containing the k first columns.

2. *GKS matrices.* These are upper triangular matrices G_n where the j^{th} diagonal entry is equal to $\frac{1}{\sqrt{j}}$, and where the element in position (i, j) for $i < j$ above the diagonal is equal to $-\frac{1}{\sqrt{j}}$. Their columns also have all unit two-norm. These matrices have been introduced in [GKS76].

3. *Gap matrices.* These are $n \times n$ matrices possessing k large singular values (about 10^3), and $n - k$ very small singular values (about 10^{-12}).

4. *Random matrices.* These are $n \times n$ matrices where each entry is drawn from the interval $[-1, 1]$ with a uniform distribution.

Most of these families of matrices are the same type as those used in [GE96]. Kahan and GKS matrices have been tested with sizes $n = 100$, 500 and 1000. For the Kahan matrices, we took $\psi = 0.9$. For the Gap matrices, we also took $n = 100$, 500 and 1000 and we used $k \in \{15, 30, 50, 150, 300\}$ with $k \leqslant \frac{n}{2}$. Five matrices have been generated for each pair (n, k). For the random matrices, we took $n \in \{30, 100, 300, 1000, 3000\}$ and $k \in \{3, 5, 15, 30, 50, 150, 300, 500, 1000\}$, again with $k \leqslant \frac{n}{2}$. Depending on the pair (n, k), up to fifty examples have been generated.

k	$\kappa(A_{:,1:k})$	Volume		Computation time	
		Initial set	Final set	SRRQR	WSS/MWSS
20	1.4×10^{4}	3.0×10^{-5}	1.6×10^{-2}	0.4	0.1
50	3.0×10^{10}	6.0×10^{-10}	1.7×10^{-2}	1.3	0.1
100	6.3×10^{20}	8.5×10^{-18}	1.7×10^{-2}	3.0	0.2
150	2.2×10^{25}	1.2×10^{-25}	1.8×10^{-2}	6.8	0.3
200	1.9×10^{28}	1.8×10^{-33}	1.8×10^{-2}	8.9	0.5
300	1.6×10^{33}	3.6×10^{-49}	1.9×10^{-2}	24.2	1.0

Table 4.1: *Computation time (in seconds) required to return a subset of k columns for a 1000×1000 Kahan matrix $A = K_{1000}$. The condition number of the default initial subset of columns is also shown.*

4.4.3 Results

In this section, we compare the performance of the different algorithms on the test matrices, the performance being related to the volume of the selected columns, and the computation time required to return this subset. In all results, the volume measure will be normalized such that $\sigma_1\sigma_2 \ldots \sigma_k = 1$. The computation times are not normalized as the raw measures already provide an interesting information.

Kahan matrices

We begin by presenting results for Kahan matrices. For these matrices, all algorithms (SRRQR, WSS, MWSS, RMWSS) return the same value for the objective function. In fact, the same set of columns is selected each time, namely the set $S = \{a_2, a_3, \ldots, a_{k+1}\}$, where a_i is the i^{th} column of the tested matrix A. This set can be obtained after one single interchange.

Table 4.1 shows the computation time for several values of k and a 1000×1000 Kahan matrix. Results for other values of n present the same behavior. The two-norm condition number of the first k columns is also shown to indicate the bad quality of the default starting point.

Even though the same set of columns is selected by all algorithms, windowed methods appear to be much faster than SRRQR, as intended. The multi-pass windowed algorithm takes about the same computation time as the simple windowed algorithm in this case. Indeed, there is

only one exchange of columns during the algorithm. Furthermore, this exchange happens at the first iteration of the sliding window. Hence, at the second pass, only the first column has to be tested. As including this column does not improve the volume, the algorithm immediately aborts the search. Note that with this new set of columns, the condition number immediately decreases by a significant factor. For example, with $k = 100$, we have $\kappa(A_{:,1:100}) \approx 6.3 \times 10^{20}$ and $\kappa(A_{:,2:101}) \approx 2.7 \times 10^{5}$. This can also be observed when we compare the volume of the initial set and the volume of the final set. The randomized version can also be used, but there is no improvement in this case. The computation time will simply be multiplied by the number of trials.

GKS matrices

Results for a GKS matrix of size 1000×1000 are presented in Tables 4.2 and 4.3. The numbers shown for the RMWSS algorithm are the minimum and maximum values obtained across 20 runs. The subscript corresponds to the value of R, the number of trials in a single run. In this experiment, there is no heuristic pruning.

Several observations can be done. First, it is clear that a single-pass WSS algorithm is very fast (as expected), but has a very poor performance in volume, especially when k grows. Indeed, the number of subsets is such that several passes are required to find a good solution. For example, with $k = 40$, SRRQR did about 150 column interchanges before returning its solution. For the same matrix, MWSS had to perform about 2000 interchanges across a total number of 12 passes. Doing a single pass is clearly insufficient in this case.

This contrasts with the previous case: the optimal set of columns for Kahan matrices is $\{a_2, a_3, \ldots, a_{k+1}\}$, which is only one swap away from the initial set $\{a_1, a_2, \ldots, a_k\}$. The optimal set for GKS matrices is more difficult to reach and does not correspond to a set of contiguous columns in the general case. For example, the best subset for $k = 4$ and $n = 100$ is $\{a_2, a_3, a_6, a_{100}\}$.

If we compare MWSS to SRRQR, it appears that the two methods are able to find subsets of similar quality. The windowed algorithm returns slightly higher volumes, but the difference is only apparent for larger values of k. This is also due to the fact that SRRQR may be very close to the optimal solution for smaller problems, hence it is not possible to find

k	$\kappa(A_{:,1:k})$	Volume		
		Initial set	SRRQR	WSS
20	1.3×10^{6}	5.4×10^{-18}	1.3×10^{-11}	1.8×10^{-12}
40	2.1×10^{12}	3.5×10^{-30}	1.1×10^{-16}	1.5×10^{-18}
60	9.5×10^{18}	9.6×10^{-41}	3.1×10^{-20}	7.1×10^{-23}
80	8.2×10^{18}	2.5×10^{-50}	5.9×10^{-23}	2.7×10^{-26}
100	9.5×10^{18}	3.1×10^{-59}	3.4×10^{-25}	5.8×10^{-29}

k	Volume/Volume(SRRQR)		
	MWSS	$\text{RMWSS}_{R=1}$	$\text{RMWSS}_{R=2}$
20	1.008	[1.009; 1.009]	[1.009; 1.009]
40	1.009	[0.978; 1.010]	[0.994; 1.010]
60	1.001	[0.984; 1.012]	[1.001; 1.015]
80	1.025	[1.014; 1.034]	[1.016; 1.036]
100	1.079	[1.078; 1.111]	[1.091; 1.114]

Table 4.2: *Volumes and volume ratios for the subsets of columns returned by the different algorithms for a* 1000×1000 *GKS matrix* $A = G_{1000}$. *The condition number of the default initial subset of columns is also shown. For the multi-pass windowed algorithms, the results corresponds to the ratio between the volume obtained by the algorithm, and the volume obtained by SRRQR. This allows us to compare the performances more easily.*

k	Computation time				
	SRRQR	WSS	MWSS	$\text{RMWSS}_{R=1}$	$\text{RMWSS}_{R=2}$
20	3.1	0.1	0.8	[0.5; 0.6]	[1.0; 1.2]
40	9.4	0.1	1.2	[0.5; 0.7]	[1.1; 1.5]
60	4.5	0.2	3.2	[1.4; 3.2]	[3.0; 5.7]
80	6.0	0.3	2.0	[1.6; 3.3]	[3.8; 6.7]
100	18.8	0.3	2.4	[1.5; 2.1]	[3.5; 4.4]

Table 4.3: *Computation time (in seconds) required to return a subset of* k *columns for a* 1000×1000 *GKS matrix* $A = G_{1000}$.

a significantly better subset of columns. This can also be seen from the fact that the volume ratios between MWSS and SRRQR are close to 1. In fact, the same observations can be done when comparing the randomized windowed algorithm to SRRQR. However, since the computational cost of MWSS is much smaller than SRRQR, the windowed algorithm seems to be more suitable for the problem in this case.

Note that the computation times presented in Table 4.3 do not have to grow when k grows. Indeed, smaller values of k may sometimes result in a slower algorithm if the number of interchanges is larger. For example, SRRQR exchanged about 150 pairs of columns in the case $k = 40$, but only 65 pairs in the case $k = 60$.

In this case, MWSS and RMWSS have similar performance measures. In general, using a randomized version of the algorithm may be a good idea if the maximization of the volume has priority over the speed of the algorithm. The trade-off between the two can for example be set using the parameter R. This will become more apparent in the next set of test matrices.

Gap matrices

Tables 4.4 and 4.5 contain results for Gap matrices. As the results for smaller matrices present the same behavior as those for larger matrices, we only show the $n = 1000$ case. The randomized algorithm has been run with $R = 1, 2, 3, 4$ with no heuristic pruning. Note that Table 4.4 also shows the average volumes obtained by RMWSS across 20 runs in addition to the min-max values as the distribution is significantly asymmetrical.

This third set of matrices behaves quite differently than the previous ones. Let us look at the volumes in Table 4.4. First, the WSS algorithm is returning worse results than SRRQR. This is expected, but in this case, the ratios of volumes are much larger. This may be an indication that SRRQR itself is returning a subset of columns that is far from being optimal. Indeed, this is confirmed by the fact that both MWSS and RMWSS are able to find subsets with a significantly larger volume, unlike in the previous examples where the ratios of volumes were all approximately equal to 1.

Another important observation is that the variability of the results is much higher with these Gap matrices. Even though, in average, the

k	Volume	Volume/Volume(SRRQR)					
	SRRQR	WSS	MWSS	$\text{RMWSS}_{R=1}$	$\text{RMWSS}_{R=2}$	$\text{RMWSS}_{R=3}$	$\text{RMWSS}_{R=4}$
15	1.6×10^{-13}	0.5	1.3	1.2	1.4	1.5	1.6
				[0.8; 1.8]	[1.0; 1.9]	[1.0; 2.0]	[1.2; 2.0]
30	4.2×10^{-24}	0.3	1.1	1.6	1.8	2.1	2.3
				[0.6; 1.8]	[1.0; 2.4]	[1.0; 2.6]	[1.3; 2.7]
50	1.3×10^{-36}	0.2	1.7	1.9	2.4	2.9	3.2
				[0.6; 4.6]	[0.9; 5.4]	[1.3; 5.5]	[1.4; 7.4]
150	1.9×10^{-82}	0.4	17.9	17.7	25.2	24.6	34.3
				[1.6; 73.4]	[3.9; 60.4]	[6.0; 78.0]	[8.2; 101.7]
300	1.5×10^{-121}	0.2	52.7	60.9	107.1	140.6	140.6
				[2.2; 256.4]	[11.7; 347.5]	[18.6; 344.1]	[26.9; 569.8]

Table 4.4: *Volumes and volume ratios returned by the different algorithms for sets of* 1000×1000 *Gap matrices with exactly k large singular values. All results are averages across the different test matrices. For RMWSS, the minimal and maximal obtained volumes are also indicated in addition to the average result.*

k	Computation time						
	SRRQR	WSS	MWSS	RMWSS$_{R=1}$	RMWSS$_{R=2}$	RMWSS$_{R=3}$	RMWSS$_{R=4}$
15	1.6	0.1	0.2	[0.1; 0.4]	[0.3; 0.7]	[0.5; 1.0]	[0.8; 1.3]
30	2.7	0.1	0.4	[0.3; 0.7]	[0.6; 1.5]	[1.1; 2.0]	[1.6; 2.6]
50	3.3	0.2	0.7	[0.6; 1.4]	[1.2; 2.8]	[2.0; 3.6]	[2.8; 4.8]
150	4.8	0.7	2.8	[2.0; 5.0]	[4.5; 8.8]	[7.4; 12.5]	[10.4; 16.4]
300	6.2	2.1	10.4	[7.5; 22.0]	[18.9; 37.0]	[29.4; 55.1]	[41.7; 66.4]

Table 4.5: *Computation time (in seconds) associated to sets of* 1000×1000 *Gap matrices with exactly* k *large singular values.*

randomized algorithm finds volumes that are similar to the deterministic version, the min-max intervals are quite large. Furthermore, increasing the number of trials R significantly improves the obtained volumes, especially with large values of the parameter k.

If we now look at the computation times in Table 4.5, it appears that MWSS (and thus RMWSS with $R = 1$) is still faster than SRRQR, except in the $k = 300$ case. However, windowed algorithms were designed for $k \ll n$, so this is not really a surprise. Even though it is not always the case, a larger value of k often implies a larger number of interchanges, and thus a larger number of passes. Hence, the time saved by using a window-based technique may be lost with multi-pass algorithms if the number of passes is too high. For example, with $k = 15$, the average number of passes was about 5 whereas with $k = 300$, the number rose to about 10. Of course, if a low computation cost is crucial to the application, it would be wise to put a limit to the maximal number of passes, especially for large values of k.

Note that until now, RMWSS has been run without the pruning heuristic, which reduces the computation time. The obvious disadvantage is that this may also decrease the volume of the subset returned by the algorithm. For Gap matrices, using this heuristic may reduce the computation time by a factor up to 2 without losing too much performance in volume by pruning about half of the columns. However, these results are quite variable depending on the tested matrices. Nevertheless, in this case, the obtained volumes are still of the same order of magnitude as results obtained without pruning. The last set of examples, presented below, will show the effect of this heuristic on larger matrices. In this case, the size of the subset is thus smaller than the numerical rank of the matrix.

Random matrices

Let us look at the performance measures for random dense matrices. Table 4.6 shows the ratios of volumes for WSS and MWSS while Table 4.7 shows the computation times.

As with Gap matrices, the single-pass windowed algorithm returns subsets with decreasing volume ratios when k is growing. However, the ratios are even larger than with Gap matrices. This suggests that there are a large number of possible subsets with a volume comparable to the

WSS	$n = 100$	$n = 300$	$n = 1000$
$k = 15$	0.98	0.99	1.00
$k = 30$	0.92	0.98	1.00
$k = 50$	0.90	0.95	0.99
$k = 150$	—	0.72	0.91
$k = 300$	—	—	0.75
$k = 500$	—	—	0.54
MWSS	$n = 100$	$n = 300$	$n = 1000$
$k = 15$	1.00	1.00	1.00
$k = 30$	1.00	1.00	1.00
$k = 50$	1.02	1.00	1.00
$k = 150$	—	1.02	1.00
$k = 300$	—	—	1.03
$k = 500$	—	—	1.09

Table 4.6: *Volume ratios obtained by WSS and MWSS on matrices with uniformly distributed random entries. All results have been averaged across 50 test cases.*

result obtained by SRRQR. This contrasts with Gap matrices where the range of volumes seems to be much larger.

This is supported by the results of MWSS. Indeed, the multi-pass algorithm returns subsets of columns with roughly same volumes, even though the subsets themselves are different. In fact, small improvements appear only for large values of k. The results with the randomized version of the algorithm are similar to those of MWSS: all ratios are close to 1, with slightly better results when k is large.

When comparing computation times, the conclusions are generally similar to the case with Gap matrices, at least for the deterministic methods: WSS is the fastest, and MWSS is faster than SRRQR for sufficiently small values of k. Note that with these random matrices, the multi-pass algorithm is still significantly faster than SRRQR for $n = 1000$, $k = 300$, which differs from the Gap case. However, for Gap matrices, this was partially due to a much larger ratio which required a larger number of passes to reach. In the general case, we conclude that MWSS is able to obtain good subsets of columns while having a significantly lower computational cost than SRRQR, provided that k is small with respect to n.

SRRQR	$n = 100$	$n = 300$	$n = 1000$
$k = 15$	0.04	0.16	1.38
$k = 30$	0.06	0.31	2.36
$k = 50$	0.07	0.46	3.79
$k = 150$	—	0.85	10.31
$k = 300$	—	—	17.73
$k = 500$	—	—	23.98

WSS	$n = 100$	$n = 300$	$n = 1000$
$k = 15$	0.01	0.02	0.07
$k = 30$	0.02	0.04	0.14
$k = 50$	0.03	0.09	0.27
$k = 150$	—	0.30	1.27
$k = 300$	—	—	4.02
$k = 500$	—	—	8.26

MWSS	$n = 100$	$n = 300$	$n = 1000$
$k = 15$	0.01	0.03	0.14
$k = 30$	0.02	0.07	0.28
$k = 50$	0.04	0.14	0.57
$k = 150$	—	0.48	2.77
$k = 300$	—	—	10.70
$k = 500$	—	—	23.89

Table 4.7: *Computation time required by SRRQR, WSS and MWSS on matrices with uniformly distributed random entries. All numbers are in seconds and have been averaged across 50 test cases.*

Finally, let us look at the results of RMWSS, depending on the pruning threshold and the number of trials. We concentrate on the case $n = 3000$, $k = 150$ in order to make the differences more apparent. The observations are similar for other sizes of matrices.

Figures 4.1 and 4.2 present the usual performance measures for the RMWSS algorithm on a test set of 25 random matrices. Three different levels of pruning are presented here. The deterministic version of the algorithm, not represented in the figures, has an average volume ratio of 1.000. This is consistent with resulted shown in Table 4.6 for smaller matrices.

The randomized version has an average volume ratio of about 1.002 or 1.004 depending on the parameters. Figure 4.1 shows that with these test matrices, neither the number of trials nor the pruning level have a significant impact on the volume of the selected subset. In the general case for other values of n and k, a slight increase in volume can sometimes be observed when increasing the number of trials, but the volume ratio remains close to 1.

Regarding computation times, things are more interesting. For reference, SRRQR needs more than 80 seconds to select a subset, while MWSS requires about 25 seconds. The two figures show that with random matrices, the pruning heuristic can decrease the computational cost by a factor of at least 1.5 without losing anything in practice. This contrasts with results obtained from Gap matrices where the increased speed came at the expense of lower volumes.

Taking into account all test examples, we conclude that the randomized algorithm with a single trial and without pruning presents similar behavior to the deterministic case. However, the two parameters allow a trade-off between volume quality and computation time, which may be useful depending on the application. For instance, RMWSS is able to obtain subsets with a larger volume than MWSS in several examples without losing too much speed, especially with the pruning algorithm.

4.4.4 Comparison between the windowed and the non-windowed approaches

Among the four algorithms presented in the previous section, it was expected that $MWSS$ produces subsets with better volumes than WSS, as doing several passes can only improve the result. Both $MWSS$ and

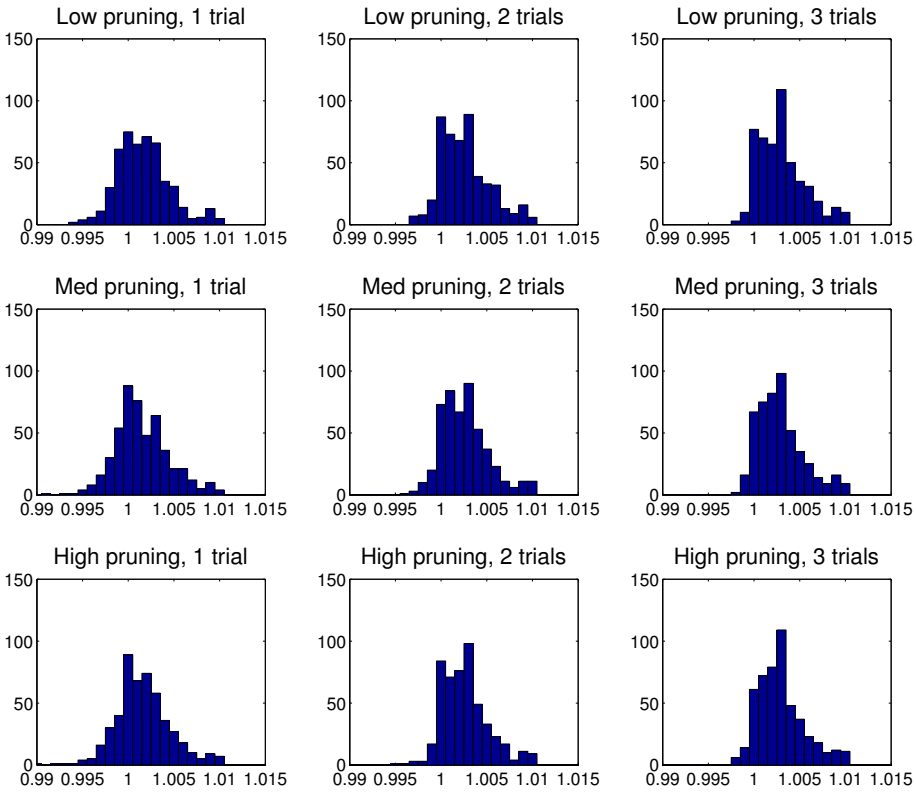


Figure 4.1: *Distribution of volume ratios obtained by RMWSS on* 3000×3000 *matrices with uniformly distributed random entries. The algorithm has been run 20 times with each set of parameters, for each test case (out of 25). Horizontal axis is the volume ratio, vertical axis is the number of occurrences (out of 500).*

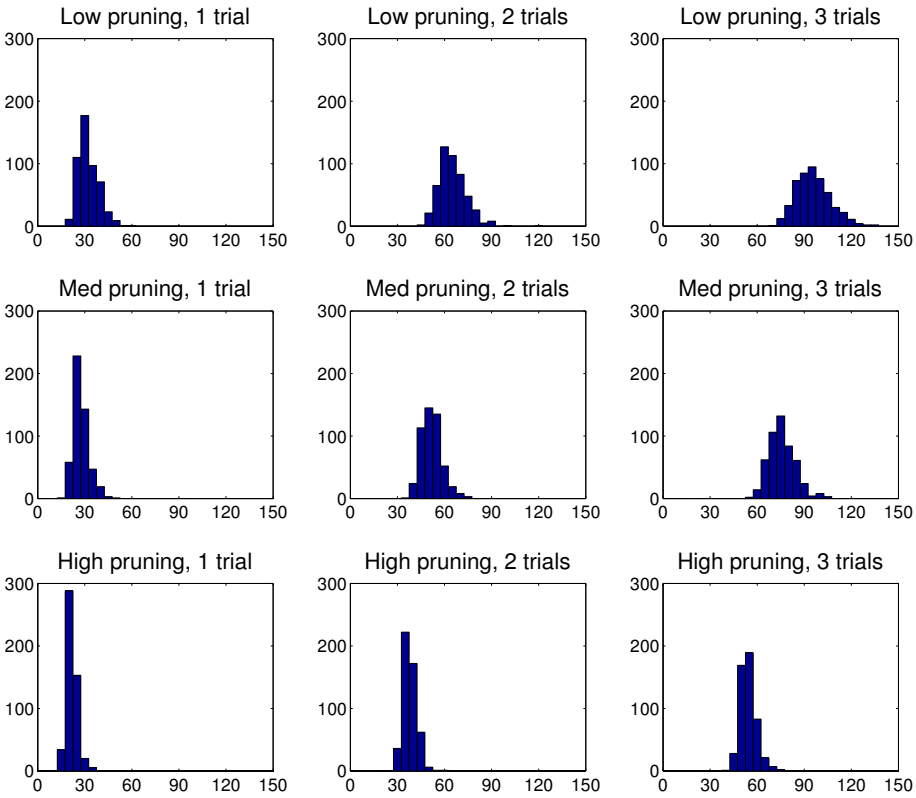


Figure 4.2: *Distribution of computation times used by RMWSS on* 3000×3000 *matrices with uniformly distributed random entries. The algorithm has been run 20 times with each set of parameters, for each test case (out of 25). Horizontal axis is the computation time in seconds, vertical axis is the number of occurrences (out of 500).*

$SRRQR$ satisfy the same local optimality property: it is not possible to improve the volume with only one exchange. However, in most (but *not* all) test cases, $MWSS$ returned subsets with a larger volume than those obtained by our implementation of $SRRQR$: the volume ratios shown in the tables are often larger than 1. This seems surprising but in fact, it can be explained by the simple fact that the greedy approach is suboptimal. Our greedy version of $SRRQR$ always chooses the exchange with the largest improvement on the volume. Hence, although the volume of the selected subsets will rapidly increase, the search space will be less explored and we will get more quickly stuck in a local maximum. The windowed approach only considers a fraction of all possibles exchanges when a single column is analyzed. Hence, the volume will increase much slower, but the number of exchanges will be much larger. This also means that the search space is better explored than with the greedy SRRQR. Moreover, as an exchange in MWSS is cheaper than an interchange in SRRQR, having to do more exchanges does not necessarily mean a higher computational cost.

Let us look at several test cases. We recall that for the Kahan matrices, both algorithms are doing the same exchanges. Things are already different for GKS matrices. Figure 4.3 shows the evolution of the volume of the subsets analyzed by SRRQR and MWSS during the computation. A time-normalized curve for MWSS is also shown for comparison purposes. This means that the red dashed line is an horizontally compressed version of the red solid curve, with a scaling factor chosen such that the ratio between the x-ranges of the two curves corresponds to the ratio between the computation times of SRRQR and MWSS. Indeed, even though MWSS has done 2402 swaps (in 10 passes) and SRRQR only 113, the first algorithm is faster than the second in this test case. These curves confirm that the volume is increasing much faster in SRRQR, but also quickly stalls. The windowed algorithm tries a large number of subsets before climbing to a local maximum, which in this case is a little bit better than the solution of SRRQR.

This behavior can be seen in Figure 4.4, which shows the number of distinct subsets during the computation that included each column of the matrix A. For the non-windowed algorithm, only 207 columns were included in at least one subset at some point during the computation. For the windowed algorithm, all 1000 columns have been included at least once in some subset. This confirms that MWSS does a better exploration of the search space than SRRQR in this case, which explains why it is

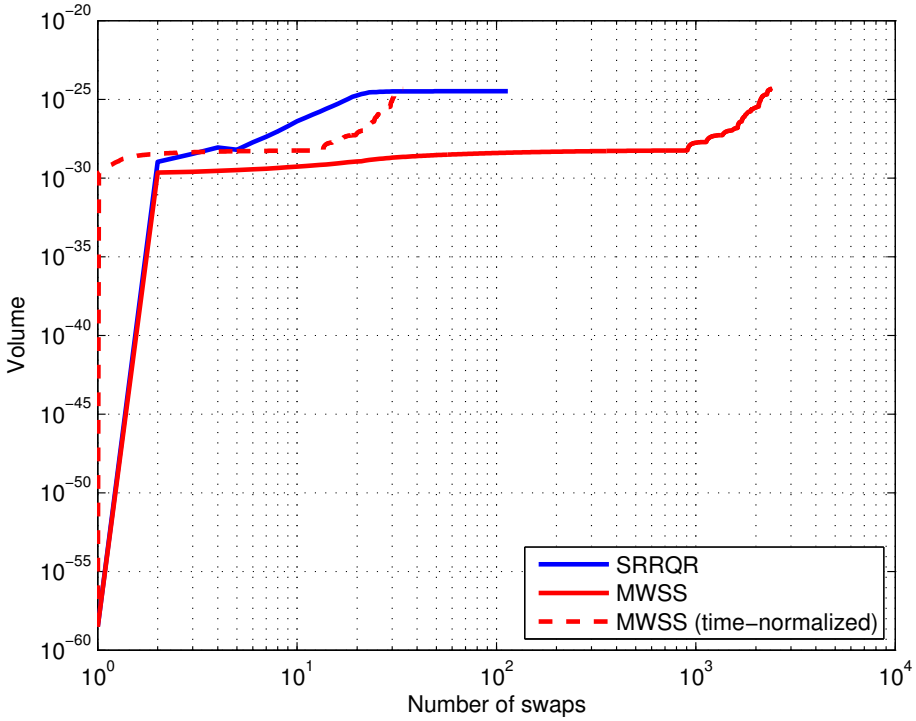


Figure 4.3: *Evolution of the volume of the subsets for a GKS matrix* G_{1000} *with* $k = 100$*, in function of the number of exchanges done by the algorithm. The red dashed line corresponds to the results obtained by MWSS, but normalized with respect to the ratio of computation times between MWSS and SRRQR, since an iteration in SRRQR is more expensive than an iteration in MWSS.*

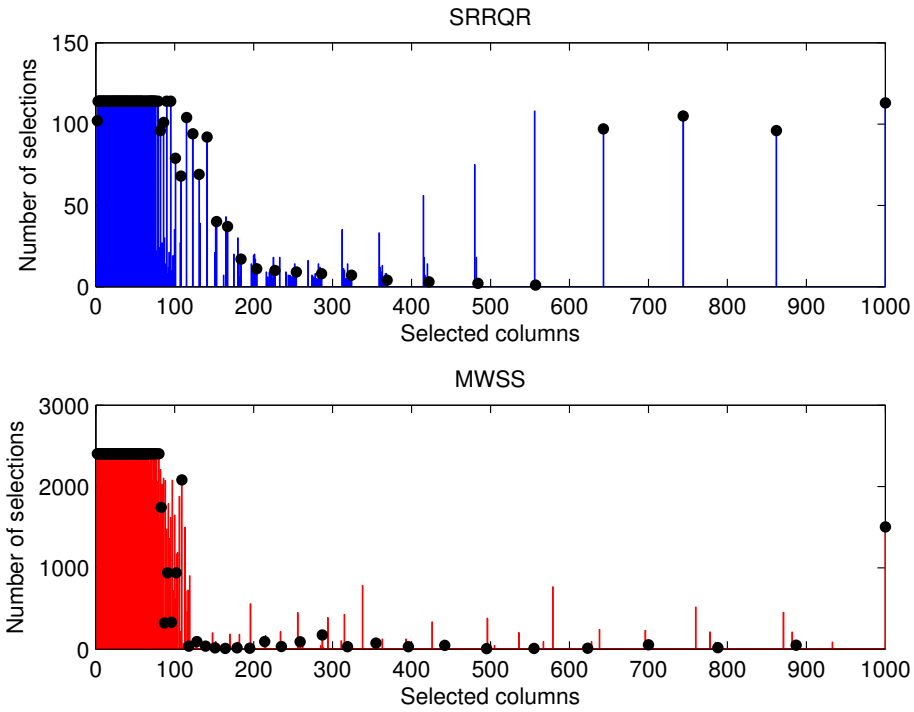


Figure 4.4: *Evolution of the subsets for a GKS matrix* G_{1000} *with* $k = 100$. *The histogram shows the number of times each column has been selected in a distinct subset. The black markers correspond to the columns present in the final subset.*

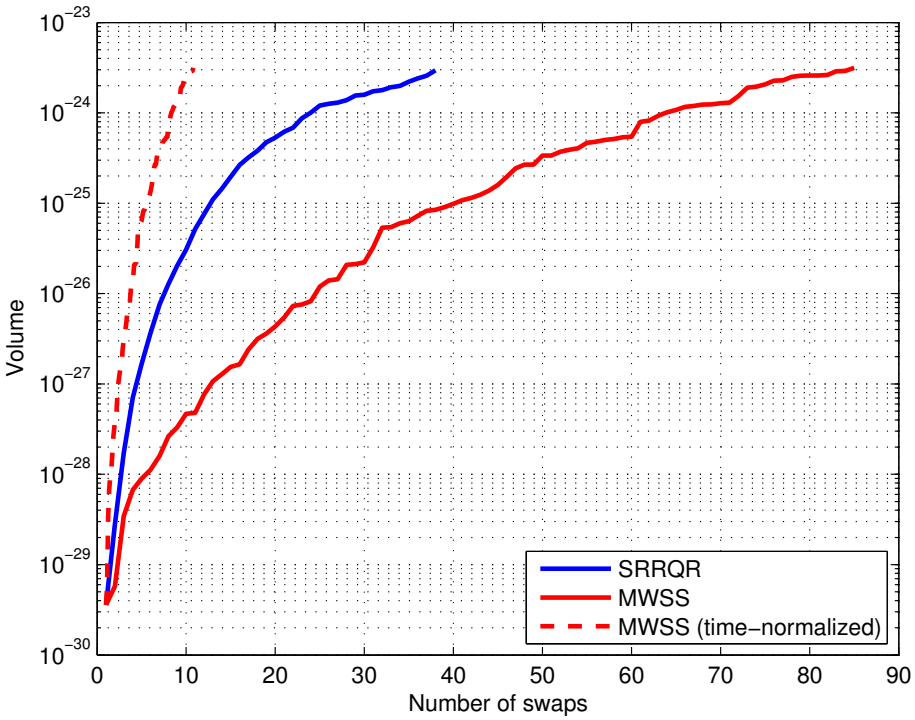


Figure 4.5: *Evolution of the volume of the subsets for a* 1000×1000 *Gap matrix with* $k = 30$*, in function of the number of exchanges done by the algorithm. The red dashed line corresponds to the results obtained by MWSS, but normalized with respect to the ratio of computation times between MWSS and SRRQR, since an iteration in SRRQR is more expensive than an iteration in MWSS.*

possible to obtain better volumes.

The same remarks hold for other types of matrices. For example, Figures 4.5 and 4.6 show the corresponding results for a Gap matrix of size 1000×1000, with subsets of 30 columns. For this example, SRRQR has found a subset with a volume of 2.96×10^{-24} after 37 swaps whereas MWSS reached 3.15×10^{-24} after 84 swaps in 5 passes. Only 67 columns have been selected at least once by SRRQR. The corresponding number for MWSS is 109 columns.

Figures 4.7 and 4.8 present another test case involving a 1000×1000 Gap matrix with $k = 300$. Here, the difference between the behaviors

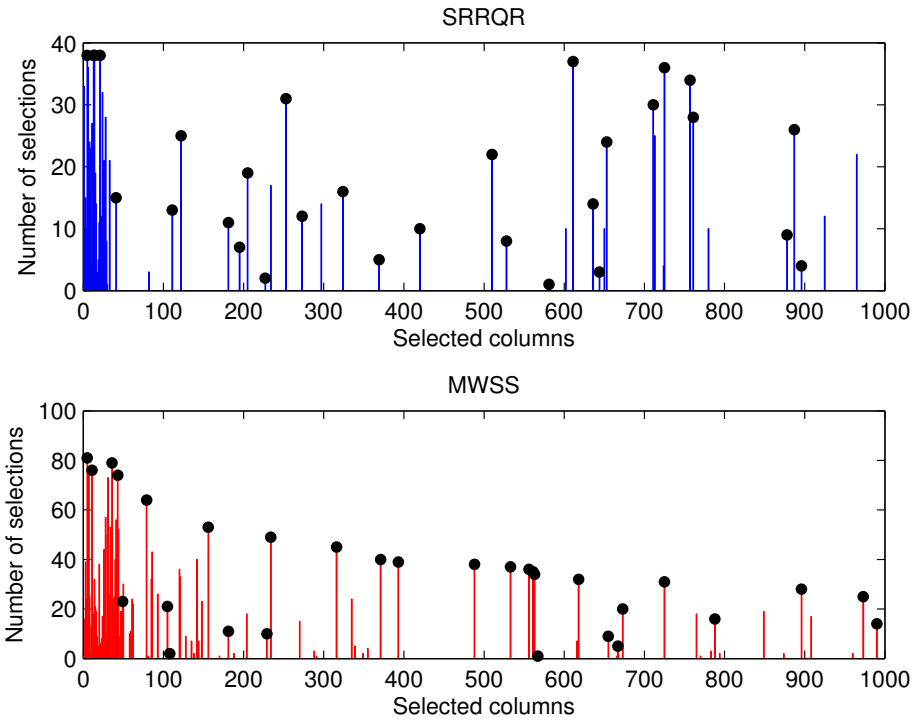


Figure 4.6: *Evolution of the subsets for a* 1000×1000 *Gap matrix with* $k = 30$. *The histogram shows the number of times each column has been selected in a distinct subset. The black markers correspond to the columns present in the final subset.*

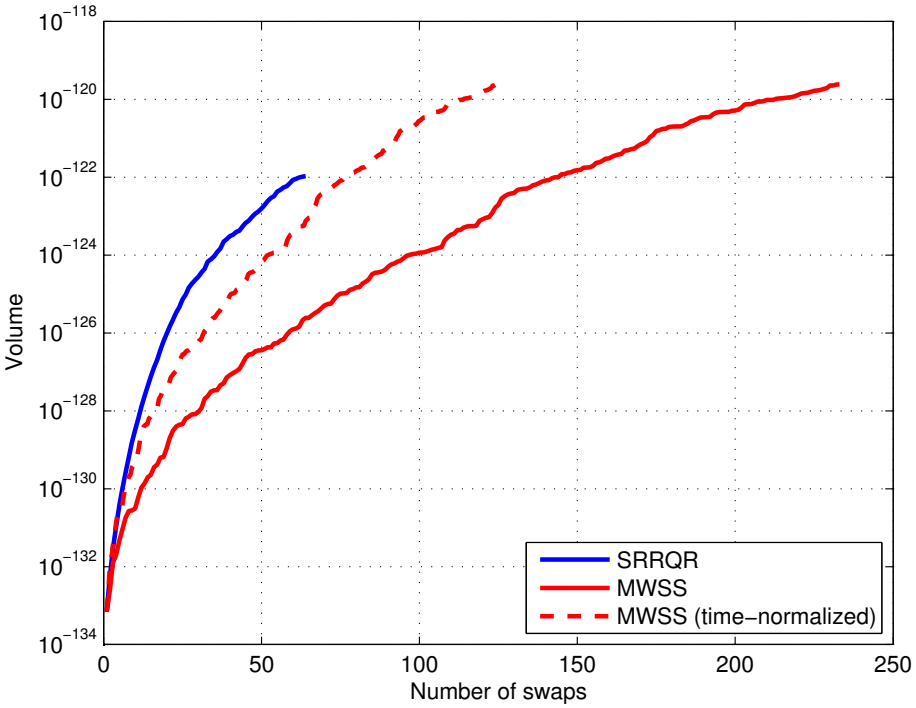


Figure 4.7: *Evolution of the volume of the subsets for a* 1000×1000 *Gap matrix with* $k = 300$*, in function of the number of exchanges done by the algorithm. The red dashed line corresponds to the results obtained by MWSS, but normalized with respect to the ratio of computation times between MWSS and SRRQR, since an iteration in SRRQR is more expensive than an iteration in MWSS.*

of the two algorithms is much more apparent. Out of the 1000 columns, only 360 have been considered by SRRQR and 505 by MWSS. Note that in this example, the windowed algorithm is slower than SRRQR. However, this is counterbalanced by a much larger volume: 2.43×10^{-120} instead of 1.07×10^{-122}.

Finally, it may be interesting to see what happens with the randomized algorithm on this example (a 1000×1000 Gap matrix with $k = 300$), since the biggest differences in the previous section were for Gap matrices with a large value of k. On the one hand, the non-greedy variant (without pruning) has been previously shown to be able to find subsets with larger volumes. On the other hand, one may wonder how does the

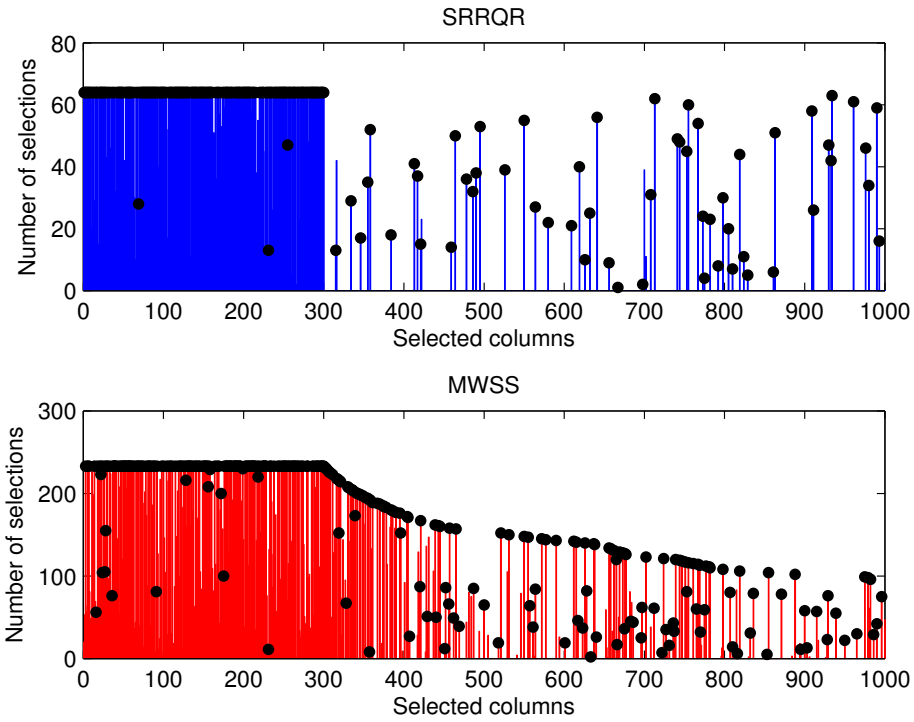


Figure 4.8: *Evolution of the subsets for a* 1000×1000 *Gap matrix with* $k = 300$. *The histogram shows the number of times each column has been selected in a distinct subset. The black markers correspond to the columns present in the final subset.*

τ	Number of columns pruned				
	in the matrix	that were considered by		in the final subset of	
		SRRQR	MWSS	SRRQR	MWSS
0.15	0/1000	0/360	0/505	0/300	0/300
0.17	5/1000	1/360	1/505	1/300	0/300
0.20	48/1000	12/360	17/505	6/300	6/300
0.22	67/1000	14/360	23/505	10/300	9/300
0.25	150/1000	32/360	45/505	22/300	22/300
0.30	452/1000	112/360	180/505	89/300	90/300
0.35	536/1000	128/360	209/505	100/300	110/300
0.40	633/1000	155/360	245/505	124/300	143/300

τ	Fraction of columns pruned				
	in the matrix	that were considered by		in the final subset of	
		SRRQR	MWSS	SRRQR	MWSS
0.15	0.0%	0.0%	0.0%	0.0%	0.0%
0.17	0.5%	0.3%	0.2%	0.3%	0.0%
0.20	4.8%	3.3%	3.4%	2.0%	2.0%
0.22	6.7%	3.9%	4.6%	3.3%	3.0%
0.25	15.0%	8.9%	8.9%	7.3%	7.3%
0.30	45.2%	31.1%	35.6%	29.7%	30.0%
0.35	53.6%	35.6%	41.4%	33.3%	36.7%
0.40	63.3%	43.1%	48.5%	41.3%	47.7%

Table 4.8: *Number of columns pruned by the heuristic on several runs of RMWSS with* $R = 1$*, on a* 1000×1000 *Gap matrix with* $k = 300$*. The second table shows the same information using percentages.*

pruning strategy behave on this example. Table 4.8 shows some information about pruned columns, for a set of runs where the randomized algorithm was able to find a better subset with $R = 1$.

Note that the set of pruned columns is not necessarily supposed to be disjoint with the set of columns considered or selected by the other algorithms. Indeed, we know that their solutions are not optimal. It appears that there are relatively more pruned columns among columns not considered by the deterministic algorithms rather than among considered columns. For example, with $\tau = 0.25$, 15% of all columns have been pruned. However, if we look at the columns examined by SRRQR

or MWSS, the pruning rate drops to about 9%. The ratio is even smaller (about 7%) among columns present in the final subset. This indicates that the pruning heuristic tends to remove "bad" columns that would not be included in the current subset even if they are kept, which fortunately is the expected behavior.

4.5 Conclusion

In this chapter we propose a window-based approach to the subset selection problem consisting in the selection of k columns of an $m \times n$ matrix such that the volume of the spanned parallelepiped is as large as possible. The computational cost of the algorithm is $\mathcal{O}(mk(n-k))$ if the number of passes is bounded.

Experimental results show that our Multi-pass Window Subset Selection algorithm is able to find subsets with a volume comparable, if not larger, than the ones obtained by the non-windowed algorithm SRRQR, while requiring a much shorter amount of time if $k \ll m, n$.

We also introduce a randomized version of the multi-pass algorithm which uses two heuristics to improve its performance. On the one hand, a strategy close to the stochastic hill climbing is used to find subsets with larger volumes, possibly with several trials. On the other hand, a pruning heuristic is used to reduce the computational cost. The combination of these two techniques offers a trade-off between quality of the subset of columns and computational cost which can be adjusted depending on the application. In particular, using a single trial without the pruning heuristic produces results similar to the deterministic case.

Of course, several extensions could be studied. Instead of having explicit parameters such as the pruning tolerance and the number of trials, one could consider another algorithm where these parameters are adaptively modified during the computation. We can also consider multi-level algorithms: instead of directly selecting k columns among n, such a method would build a decreasing sequence of nested subsets ending with a subset of size k.

Furthermore, it could also be interesting to investigate the behavior of these algorithms with other objective functions. For example, one could choose to maximize the smallest singular value instead of maximizing the volume of the subset of columns, or consider minimizing the residual between the selected columns and the discarded columns.

CHAPTER 5
The joint spectral radius

In this chapter, we present our contributions about the *joint spectral radius*, which is a generalization of the spectral radius to sets of matrices. We present a survey of different approaches and methods for the approximation of the joint spectral radius. These methods are now implemented in MATLAB® in the *JSR Toolbox* [VHJ+11]. We also propose a new method based on a genetic algorithm, which is shown to be able to find lower bounds on the joint spectral radius with a low computational cost. Our main publications related to this chapter are [CB12a, CB12b, CB11a].

5.1 Introduction

The *joint spectral radius* characterizes the maximal asymptotic growth rate of products of matrices drawn from a given set, generalizing the notion of spectral radius for a single matrix. It was initially introduced by Rota and Strang in [RS60] and has since then appeared in many applications such as system theory [Gur95, Koz07], the study of wavelets [DL92, Mae98], combinatorics and language theory [JPB09], the capacity of some types of codes [MOS01, MOS07, BJP06], etc. More examples and a general overview of what is known for the joint spectral radius can be found in [Jun09]; see also [SWM+07] for more specific results.

In particular, the joint spectral radius is closely related to the stability of *switching systems* such as

$$x_0 \in \mathbb{R}^n, \qquad x_{t+1} = M_t x_t \quad M_t \in \Sigma \subset \mathbb{R}^{n\times n}, \tag{5.1}$$

where the transformation M_t applied to the state x_t may change at each iteration, and Σ corresponds to the set of possible transformations. Such systems appear in many applications, for example when the state

variables are updated asynchronously, when the updates depend on the current state, or in the presence of uncertainty (see [Lib03] for an introduction on switching systems). Of course, the sequence of transition matrices M_t may also be controlled.

The stability of the system (5.1) *under arbitrary switching*, in the sense that $\lim_{t\to\infty} x_t = 0$ for all sequences $(M_t)_{t\in\mathbb{N}}$, is governed by the joint spectral radius of the set Σ. In the particular case where M_t is constant, and thus Σ consists of a single matrix, the joint spectral radius corresponds to the usual spectral radius $\rho(\cdot)$. Note that having $\rho(M_t) < 1$ for all t does not imply stability for the switching system, as can be seen with the following simple example.

Example 5.1. *Let us consider the matrices*

$$A = \begin{pmatrix} \frac{3}{4} & 0 \\ 1 & \frac{3}{4} \end{pmatrix}, \qquad B = \begin{pmatrix} \frac{3}{4} & 1 \\ 0 & \frac{3}{4} \end{pmatrix}.$$

It is clear that A and B are both stable: $\rho(A) < 1$ and $\rho(B) < 1$. However, alternating the two matrices results in a growth rate of $\rho(AB)^{\frac{1}{2}} = \left(\frac{17}{16} + \frac{1}{4}\sqrt{13}\right)^{\frac{1}{2}}$ in this case, which is greater than 1. Hence, the system is unstable.

The problem of approximating the joint spectral radius has been widely studied. It is known that this quantity is NP-hard to compute [TB97]. A dozen algorithms have been proposed this last decade for approximating the joint spectral radius but little is known about their practical efficiency. Indeed, some of them have only been studied from a theoretical point of view, and were never implemented in practice. In this work, we are interested in the practical behavior of these approximation algorithms. Our implementations can be found in a MATLAB® toolbox [VHJ+11], which also contains an algorithm that combines different existing approaches.

We also propose a new heuristic algorithm whose goal is to compute lower bounds on the joint spectral radius while aiming at a very short computation time. This is achieved by the use of a genetic algorithm: starting with a set of random products of limited length, the method iteratively generates new products by combining existing ones, based on the value of their spectral radius. The maximal allowed product length is slowly increased during the computation so that the search space is not restricted to products of small length. The lower bound returned by

the algorithm corresponds to the best result found by considering all the products generated during the process.

Even though there is no guarantee on the accuracy of the result, according to our simulations, the method may in fact be faster than existing techniques, depending on the parameters, and can still be used when the problem size becomes large, i.e., when Σ contains a large number of matrices and/or the size of the matrices themselves is large. The numerical experiments we present also show that, for large size problems, our genetic algorithm tends to provide better bounds on the joint spectral radius than existing techniques, or that it provides bounds for situations where the computation is simply too expensive with other methods.

This chapter is organized as follows. Section 5.2 presents the basics about the joint spectral radius. In Section 5.3, we survey the different approaches and methods for the problem of approximating the joint spectral radius. After that, we present our genetic algorithm in Section 5.4. The remaining sections detail our numerical experiments and compare the different algorithms based on the results obtained from these simulations. We also comment on several details related to the implementation of the methods.

5.2 Basic properties and results

Let $\Sigma \subset \mathbb{R}^{n\times n}$ be a set of real matrices. The *joint spectral radius* $\rho(\Sigma)$ is formally defined by

$$\rho(\Sigma) = \lim_{t\to\infty} \rho_t(\Sigma, \|\cdot\|), \tag{5.2}$$

with

$$\rho_t(\Sigma, \|\cdot\|) = \max\left\{ \|M\|^{1/t} \mid M \in \Sigma^t \right\}.$$

Here, Σ^t denotes the set of products of length t of matrices in Σ. One can show that the limit in Equation (5.2) exists and that it does not depend on the matrix norm that is used, provided this norm is submultiplicative. This is due to Fekete's Subadditive Lemma:

Lemma 5.2 ([Fek23])**.** *Let $(a_n)_{n\in\mathbb{N}}$ be a subadditive sequence of real numbers, i.e., $a_{m+n} \leqslant a_m + a_n$ for all $m, n \in \mathbb{N}$. Then, the limit*

$$\lim_{n\to\infty} \frac{a_n}{n}$$

exists and is equal to $\inf_n \frac{a_n}{n}$. This limit may be equal to $-\infty$.

Let us prove the existence of the limit in Equation (5.2). We consider the sequence $(a_n)_{n\in\mathbb{N}}$ defined by

$$a_n = \log \rho_n(\Sigma, \|\cdot\|)^n = \log \max_{M\in\Sigma^n} \|M\|.$$

This sequence is subadditive because of the submultiplicativity of the norm $\|\cdot\|$. Indeed, let M_{opt} be a product in Σ^{m+n} with maximal norm and let us write $M_{opt} = M_1 M_2 \dots M_{m+n}$ with $M_i \in \Sigma$ for all i. Then,

$$\begin{aligned} a_{m+n} &= \log \|M_{opt}\| \\ &\leqslant \log \left(\|M_1 \dots M_m\| \|M_{m+1} \dots M_{m+n}\|\right) \\ &\leqslant \log \max_{M\in\Sigma^m} \|M\| + \log \max_{M\in\Sigma^n} \|M\| \\ &= a_m + a_n. \end{aligned}$$

Then, Lemma 5.2 allows us to conclude that the limit in Equation (5.2) exists since

$$\lim_{n\to\infty} \frac{a_n}{n} = \lim_{n\to\infty} \log \rho_n(\Sigma, \|\cdot\|).$$

Note that if $\rho_n(\Sigma, \|\cdot\|) = 0$ for some value of n, then the corresponding term a_n is undefined. However, $\rho_n = 0$ implies that $\rho_t = 0$ for all $t \geqslant n$, hence the limit still exists in this case.

In the particular case where Σ contains only one matrix M, the joint spectral radius is equal to the usual spectral radius, i.e., the largest magnitude of the eigenvalues. Equation (5.2) is thus a generalization of the well-known Gelfand formula for the spectral radius of a single matrix:

$$\rho(M) = \lim_{t\to\infty} \|M^t\|^{1/t}.$$

The first algorithms proposed consisted in constructing products of increasing length and using ρ_t as upper bounds. Indeed, Fekete's lemma also tells us that the limit of the sequence of ρ_t's corresponds to its infimum. Lower bounds can be obtained by considering the *generalized spectral radius* $\bar{\rho}(\Sigma)$ defined by

$$\bar{\rho}(\Sigma) = \limsup_{t\to\infty} \bar{\rho}_t(\Sigma), \tag{5.3}$$

with

$$\bar{\rho}_t(\Sigma) = \max\left\{\rho(M)^{1/t} \mid M \in \Sigma^t\right\}.$$

For all lengths t, the following inequalities are satisfied [BW92]:

$$\bar{\rho}_t(\Sigma) \leqslant \bar{\rho}(\Sigma) \leqslant \rho(\Sigma) \leqslant \rho_t(\Sigma, \|\cdot\|). \tag{5.4}$$

It was moreover proved in [BW92] that the generalized spectral radius is equal to the joint spectral radius whenever Σ is bounded (or, in particular, finite). For bounded sets, the inequalities in Equation (5.4) can thus be used to derive arbitrarily precise approximations for the joint spectral radius. The fact that this does not hold for unbounded sets can be illustrated with the following well-known example.

Example 5.3. *Let us consider the set Σ defined by*

$$\Sigma = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}, \ldots \right\}.$$

All products of matrices in Σ have a spectral radius equal to 1, hence the generalized spectral radius satisfies $\bar{\rho}(\Sigma) = 1$. However, the joint spectral radius satisfies $\rho(\Sigma) = \infty$. Hence, $\bar{\rho}(\Sigma) < \rho(\Sigma)$ in this case.

Other spectral quantities can be defined. For example, if we consider the minimal growth rate, the joint spectral subradius (also called the lower spectral radius) and the generalized spectral subradius can be defined analogously by replacing the max with a min and the lim sup by a lim inf in the definitions given previously. Several methods presented in the following sections can be adapted for the computation of the joint spectral subradius. However, we will concentrate on the joint spectral radius of finite sets Σ.

The sequence of upper bounds in (5.4) converges very slowly to $\rho(\Sigma)$ except in some particular cases. Hence, any approximation algorithm directly based on the inequalities (5.4) is bound to be inefficient. This is not so surprising since the problem of approximating the joint spectral radius is known to be NP-hard:

Theorem 5.4 ([TB97])**.** *Unless $P = NP$, there is no algorithm that, given a set of matrices Σ and a relative tolerance ε, is able to return an $(1+\varepsilon)$-approximation of $\rho(\Sigma)$ in a time polynomial in the size of Σ and ε, that is, an approximation of $\rho(\Sigma)$ with a relative error smaller than ε. This result holds even if Σ is composed of matrices with binary entries.*

Another result supports the fact that the joint spectral radius is a quantity that is difficult to compute:

Theorem 5.5 ([BT00]). *Given a set of matrices Σ, the problem of determining whether the set of all products of matrices in Σ is bounded is Turing-undecidable. The problem of determining if $\rho(\Sigma) \leqslant 1$ is also Turing-undecidable. These results remain true even if the matrices in Σ have nonnegative rational entries.*

In other words, it is impossible to construct an algorithm that is able to determine, in finite time and for all sets Σ, whether the set of all products of matrices in Σ is bounded, or whether the joint spectral radius $\rho(\Sigma)$ is strictly less than 1.

Extremal norms

As the sequence of upper bounds tends to converge very slowly, one may want to speed up the procedure, for example by finding an appropriate norm that gives a faster convergence rate in the expression (5.2). In some cases, it is even possible to find a norm that is *extremal* with respect to the set of matrices, that is, a norm such that the joint spectral radius is reached with a product of length one. More precisely, a norm $\|\cdot\|$ is said to be extremal for a set of matrices Σ if $\|M\| \leqslant \rho(\Sigma)$ for all $M \in \Sigma$.

Example 5.6. *Let us consider the set of matrices $\Sigma = \{A, B\}$ with*

$$A = \begin{pmatrix} 1 & -2 \\ -2 & 4 \end{pmatrix}, \qquad B = \begin{pmatrix} 4 & 2 \\ 2 & -5 \end{pmatrix}.$$

We have $\rho(A) = 5$ and $\rho(B) = \frac{1}{2}(1 + \sqrt{97}) \approx 5.424$. Hence,

$$\rho(\Sigma) \geqslant \bar{\rho}_1(\Sigma) = \max\{\rho(A), \rho(B)\} = \frac{1}{2}(1 + \sqrt{97}).$$

Let us consider the Frobenius norm. We have $\|A\|_F = 5$ and $\|B\|_F = 7$. Hence,

$$\rho(\Sigma) \leqslant \rho_1(\Sigma, \|\cdot\|_F) = \max\{\|A\|_F, \|B\|_F\} = 7.$$

Better upper bounds can be obtained by considering products of several matrices in Σ:

t	1	2	3	4	5	6
$\rho_t(\Sigma, \|\cdot\|_F)$	7.000	5.945	5.6635	5.547	5.491	5.462

Let us now consider the two-norm. We have $\|A\|_2 = 5$ and $\|B\|_2 = \frac{1}{2}(1 + \sqrt{97}) \approx 5.424$. Indeed, the two-norm of a symmetric matrix is equal to its spectral radius. Thus, the two-norm is extremal in this case, and $\rho(\Sigma) = \frac{1}{2}(1 + \sqrt{97})$.

This example also shows that the joint spectral radius of a set of symmetric matrices is equal to the largest spectral radius of these matrices.

It can be proved that a bounded set Σ that is commonly irreducible admits an extremal norm ([Bar88, Koz90], see also [Wir02]). We recall that a set of matrices is *commonly reducible* if there exists a nontrivial linear subspace that is invariant under all matrices in Σ. This also means that the matrices of a commonly reducible set Σ can be simultaneously block-triangularized by a single invertible matrix:

$$\exists\, T\ \forall\, M_i \in \Sigma : TM_iT^{-1} = \begin{pmatrix} A_i & B_i \\ 0 & C_i \end{pmatrix}.$$

In the commonly reducible case, the joint spectral radius $\rho(\Sigma)$ is equal to $\max\{\rho(\Delta_1), \rho(\Delta_2)\}$, where $\Delta_1 = \{A_i\}$ and $\Delta_2 = \{C_i\}$. This is due to the following properties:

- The usual spectral radius satisfies $\rho(A) = \rho(TAT^{-1})$ for all invertible matrices T. This still holds for the joint spectral radius since if we consider any product $M_1M_2 \dots M_k \in \Sigma^k$, we have

$$\begin{aligned}\rho((TM_1T^{-1})(TM_2T^{-1}) \dots (TM_kT^{-1})) \\ = \rho(T(M_1M_2 \dots M_k)T^{-1}) = \rho(M_1M_2 \dots M_k),\end{aligned}$$

  where T is an invertible matrix.

- The usual spectral radius of a block-triangular matrix safisties

$$\rho\left(\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}\right) = \max\{\rho(A), \rho(C)\}.$$

- A product $M_1M_2 \dots M_k$ of such block-triangular matrices has the same structure, with the two diagonal blocks being $A_1A_2 \dots A_k$ and $C_1C_2 \dots C_k$.

Finally, we also have the useful property [RS60, BW92]

$$\rho(\Sigma) = \inf_{\|\cdot\|} \sup_{M \in \Sigma} \|M\|, \tag{5.5}$$

where the infimum is taken over the set of all induced matrix norms.

The Finiteness Conjecture

It has also been observed that the lower bound in (5.4) often reaches the joint spectral radius for some finite t. A set of matrices Σ is said to possess the *finiteness property* if there exists some product $M \in \Sigma^t$ such that $\rho(\Sigma) = \rho(M)^{1/t}$. It was first conjectured by Lagarias and Wang in [LW95] that all sets of matrices do have the finiteness property — a conjecture known as the *Finiteness Conjecture* — but this conjecture was proved to be false [BM02, BTV03, Koz05].

The proofs in these articles are nonconstructive and recently an explicit counterexample has been provided by Hare et al. in [HMST11] by extending the results in [BM02, BTV03, Koz07]. Note that the question of determining if all sets of *rational* matrices have the finiteness property is still open [BJP06, JB08]. This "restricted" finiteness conjecture is interesting because matrices that appear in applications often have rational, integer or even binary entries.

Note that stability is decidable for sets of matrices possessing the finiteness property. Indeed, using (5.4), $\rho(\Sigma) < 1$ can be concluded as soon as we have an upper bound smaller than 1. Moreover, $\rho(\Sigma) \geqslant 1$ can be detected if we find a (finite) product with a unit spectral radius.

5.3 Methods of computation

The problem of approximating the joint spectral radius is far from trivial, except in some particular cases. A first naive approach would be to consider the relations (5.2), (5.3), (5.4) and evaluate all products in Σ^t for increasing values of t. Of course, these bounds will converge to the exact value of the joint spectral radius, but since the number of products to consider grows exponentially with the product length, one has to reduce the search space in some way. Several arguments may be used, e.g., the fact that the spectral radius of a product of matrices is invariant under cyclic permutations of its factors [Mae96]. Methods based on enumeration of products are presented in Section 5.3.1.

As mentioned in the previous section, the sequence of upper bounds may converge very slowly even if a lower bound may reach $\rho(\Sigma)$ for a small product due to the finiteness property. A detailed analysis of the quality of the upper bounds may by found in [Koz09]. Thus, another possible approach would be to try to find an extremal norm associated

to the set Σ. Methods for approximating the joint spectral radius with techniques based on the optimization problem (5.5) are presented in Section 5.3.2, whereas Section 5.3.3 describes geometric algorithms that iteratively approximate an extremal norm.

5.3.1 Product enumeration methods

Algorithms based on enumeration of products are mainly branch-and-bound methods based on (5.4).

Gripenberg's algorithm

Gripenberg's branch-and-bound method [Gri96] was one of the first algorithms proposed for approximating the joint spectral radius. Given a target tolerance ε and a norm $\|\cdot\|$, this algorithm starts with an initial set $\Pi_1 = \Sigma$ of candidates, and the corresponding natural bounds, i.e., $\alpha_1 = \max_{M\in\Sigma} \rho(M)$ as lower bound and $\beta_1 = \max_{M\in\Sigma} \|M\|$ as upper bound. In general, the upper bounds will depend on the chosen norm.

At the k^{th} iteration of the algorithm, let us define the new set of candidates:

$$\Pi_k = \{MP \mid M \in \Sigma, P \in \Pi_{k-1}, \mu(M, P_1, \ldots, P_{k-1}) > \alpha_{k-1} + \varepsilon\},$$

where $P = P_1 \ldots P_{k-1}$ and $\mu(M_1, \ldots, M_k) = \min_{1\leqslant i\leqslant k} \|M_1 \ldots M_i\|^{1/i}$. The bounds are then updated as follows:

$$\begin{aligned} \alpha_k &= \max\left\{\alpha_{k-1}, \max_{P\in\Pi_k} \rho(P)^{1/k}\right\}, \\ \beta_k &= \min\left\{\beta_{k-1}, \max\left\{\alpha_{k-1}+\varepsilon, \max_{P\in\Pi_k} \mu(P_1, \ldots, P_k)\right\}\right\}, \end{aligned}$$

with $P = P_1, \ldots, P_k$ in this last expression.

At each iteration, the bounds satisfy $\alpha_k \leqslant \rho(\Sigma) \leqslant \beta_k$ and we have $\lim_{k\to\infty}(\beta_k - \alpha_k) \leqslant \varepsilon$. Hence, this method provides arbitrarily precise approximations of the joint spectral radius $\rho(\Sigma)$ when given sufficient computational resources. The number of iterations required to achieve a given absolute accuracy of ε is not known a priori and may depend on the choice of the norm $\|\cdot\|$. Even when it is possible to discard a large number of products at each step, it may be necessary to reach very long products in order to obtain an interval of length ε.

Moision et al.'s pruning algorithm

In [MOS01], Moision et al. introduces a so-called *pruning algorithm* for sets of nonnegative matrices. This method is based on several dominating relations, e.g., the fact that if $A \geqslant B \geqslant 0$ componentwise, then $\rho(A) \geqslant \rho(B)$. Similar conditions can also be found for the upper bounds, depending on the norm used in the algorithm. Of course this does not lead to a polynomial-time algorithm in the general case, but the number of products may be significantly reduced in several particular cases.

The pruning algorithm works as follows: at the k^{th} iteration, i.e., when products of length k are considered, instead of checking all $|\Sigma|^k$ candidates, only a smaller subset Γ_k is analyzed. Let $\Gamma_0 = \{I\}$. Given a subset Γ_k, the next subset Γ_{k+1} is obtained by building $\Gamma_k\Sigma$, then removing all products $B \in \Gamma_k\Sigma$ that are *dominated* by another product A in the same set, i.e., products B such that $\|AM\| \geqslant \|BM\|$ for all $M \geqslant 0$ componentwise.

This condition ensures that $\rho_k(\Sigma, \|\cdot\|)$ is correctly computed. Indeed, by construction, we have

$$\max\left\{\|M\|^{1/k} \;\middle|\; M \in \Sigma^k\right\} = \max\left\{\|M\|^{1/k} \;\middle|\; M \in \Gamma_k\right\}$$

since we have only removed products with smaller norm. A similar strategy can be used for the lower bounds.

Product enumeration methods in practice...

The main disadvantages of this family of methods are the large number of products that we may have to consider when reaching very long products, which may result in a large amount of computation time when a small interval is required, and the influence of the choice of the norm on the convergence of the sequence of upper bounds. In the general case, the sequence of upper bounds converges very slowly to the joint spectral radius. However, it may be possible to find tight lower bounds in a short amount of time as the set of matrices may have an optimal product of small length. Moreover, implementing these methods is usually straightforward as it mainly consists in manipulating products of matrices. Hence, this should not give rise to more numerical errors than basic arithmetic operations, except for one detail: if the length of the products grows too much, a direct application of the method may reach

the overflow threshold, depending on the set of matrices. In order to avoid this issue, it may be wise to use some scaling of the matrices.

5.3.2 Norm optimization methods

Since the value of $\rho_t(\Sigma, \|\cdot\|)$ converges slowly to $\rho(\Sigma)$ in the general case, another approach is try to find a so-called extremal norm in order to obtain better upper bounds. This is usually done by considering a modified version of (5.5). Indeed, optimizing on the space of all matrix norms is not an easy task, so in practice one may instead consider a subset of norms. Of course, this subset should preferably be chosen so that the restricted optimization problem is easy to solve.

The ellipsoidal norm approximation

A first method introduced in [BNT05] is the class of *ellipsoidal norms* (also called ellipsoid norms).

Given a positive definite matrix $P \in \mathbb{R}^{n\times n}$ (denoted by $P \succ 0$), let us consider the so-called *ellipsoidal* vector norm $\|x\|_P = \sqrt{x^T P x}$ for $x \in \mathbb{R}^n$. The associated ellipsoidal matrix norm is the corresponding induced matrix norm:

$$\|M\|_P = \max_{x\neq 0} \frac{\|Mx\|_P}{\|x\|_P} = \max_{x\neq 0} \frac{\sqrt{x^T M^T P M x}}{\sqrt{x^T P x}}. \tag{5.6}$$

Similarly to (5.5), the ellipsoidal norm approximation $\widehat{\rho}_{\text{Ell}}(\Sigma)$ is then defined by:

$$\widehat{\rho}_{\text{Ell}}(\Sigma) = \inf_{P\succ 0} \max_{M\in\Sigma} \|M\|_P,$$

which is obviously an upper bound on the joint spectral radius. This approximation can be easily computed using semidefinite programming (SDP). Indeed, Equation (5.6) implies that for each matrix $M \in \Sigma$ we have:

$$x^T \left(\|M\|_P^2 P - M^T P M\right) x \geqslant 0, \qquad \forall x \in \mathbb{R}^n.$$

The ellipsoidal norm approximation corresponds thus to the minimum value of γ such that there exists a positive definite matrix $P \succ 0$ with $\gamma^2 P - M^T P M \succeq 0$ for all matrices $M \in \Sigma$. For a given value of γ, the problem of finding a matrix P corresponds to an SDP feasibility

problem that can be solved efficiently, and thus the optimal value of γ can be found by bisection.

The following bounds are known to hold for the ellipsoidal norm approximation $\widehat{\rho}_{\text{Ell}}(\Sigma)$:

Proposition 5.7 ([AS98, BNT05])**.** *Let Σ be a set of real $n \times n$ matrices. Then, the ellipsoidal norm approximation of its joint spectral radius satisfies*

$$\frac{1}{\sqrt{\eta}}\widehat{\rho}_{\text{Ell}}(\Sigma) \leqslant \rho(\Sigma) \leqslant \widehat{\rho}_{\text{Ell}}(\Sigma),$$

with $\eta = \min\{n, |\Sigma|\}$, where $|\Sigma|$ corresponds to the size of Σ, i.e., the number of matrices in the set.

However, in practice, better lower bounds can easily be obtained by considering products of small lengths.

The ellipsoidal norm approach can be interpreted as the search for a quadratic common Lyapunov function in order to prove the stability of a switched system when the matrices are scaled by $\frac{1}{\gamma}$. Indeed, if the scaled system $x_{t+1} = M_t x_t$ with $M_t \in \frac{1}{\gamma}\Sigma$ is asymptotically stable for all switching schemes, then we have $\rho(\Sigma) < \gamma$. A generalization of this approach using multiple Lyapunov functions for the approximation of the joint spectral radius has been proposed in [AJPR11] (see also [DB01]).

The sum-of-squares approximation

The ellipsoidal norm method has been generalized by several authors, including Parrilo and Jadbabaie. In [PJ08] these authors propose a generalization by replacing the norms by positive polynomials, based on the following result:

Proposition 5.8. *Let Σ be a set of $n \times n$ matrices and $p(x)$ be a (strictly) positive homogeneous polynomial of degree $2d$ with n variables that satisfies $p(Mx) \leqslant \gamma^{2d} p(x) \forall x \in \mathbb{R}^n$, for all $M \in \Sigma$. Then, $\rho(\Sigma) \leqslant \gamma$.*

Even though positive polynomials are hard to characterize, a positivity constraint can be relaxed into a sum-of-squares (SOS) constraint: instead of $p(x) \geqslant 0$, we require the existence of a decomposition $p(x) = \sum_i p_i(x)^2$. Note that although a sum of squares is obviously nonnegative, most but *not all* nonnegative polynomials can be rewritten as sums of squares (see [Rez00] for a survey on the problem).

The sum-of-squares decomposition can also be written as $p(x) = \left(x^{[d]}\right)^T P x^{[d]}$, where $x^{[d]}$ is a vector containing all monomials of degree d with n variables, and $P \succeq 0$ is a positive semidefinite matrix. The problem of checking if a polynomial is a sum-of-squares is thus equivalent to a SDP feasibility problem and the sum-of-squares approximation can thus be expressed as follows:

$$\begin{aligned} \widehat{\rho}_{\mathrm{SOS},2d}(\Sigma) \quad &= \quad \min\ \gamma \\ &\text{s.t.} \quad \exists\, p(x) \in \mathbb{R}[x]_{2d} \text{ homogeneous} \\ &\qquad p(x) \text{ is SOS} \\ &\qquad \gamma^{2d} p(x) - p(Mx) \text{ is SOS } \forall M \in \Sigma. \end{aligned}$$

As with the ellipsoidal approximation, the optimal value of γ can be found by bisection. In fact, in the particular case $d = 1$, the problem is equivalent to the ellipsoidal norm case because all quadratic nonnegative polynomials can be written as sums of squares.

In the general case, we have the following bounds on the approximation accuracy:

Proposition 5.9 ([PJ08])**.** *Let Σ be a set of real $n \times n$ matrices. Then, the sum-of-squares approximation of degree $2d$ of its joint spectral radius satisfies*

$$\left(\frac{1}{\sqrt{\eta}}\right)^{\frac{1}{d}} \widehat{\rho}_{\mathrm{SOS},2d}(\Sigma) \leqslant \rho(\Sigma) \leqslant \widehat{\rho}_{\mathrm{SOS},2d}(\Sigma),$$

where $\eta = \min\left\{\binom{n+d-1}{d}, |\Sigma|\right\}$.

In theory, one can thus obtain arbitrarily sharp approximations by taking polynomials of sufficiently large degree since $\lim_{d\to\infty} \eta^{-\frac{1}{2d}} = 1$, but the computational cost increases accordingly.

The joint conic radius

Another generalization of the ellipsoidal norm approximation was presented in [PJB10], where the authors extend the SDP problem to general conic programming. This extension allows one to derive upper and lower bounds on the joint spectral radius, provided that the matrices in Σ leave

a common cone K *invariant*, i.e., $MK \subset K$ for all $M \in \Sigma$. In this case, the *joint conic radius* $\widehat{\rho}_{\mathrm{Conic},K}(\Sigma)$ is defined as

$$\widehat{\rho}_{\mathrm{Conic},K}(\Sigma) = \inf\left\{\lambda \geqslant 0 \mid \exists v \in \operatorname{int} K : \lambda v - Mv \in K \ \forall M \in \Sigma\right\}.$$

For example, if all entries of the matrices in Σ are nonnegative, then $\mathbb{R}^n_+$ is an invariant cone.

The joint conic radius satisfies the following property:

Proposition 5.10 ([PJB10])**.** *Given a set Σ of real $n \times n$ matrices which leave a common cone K invariant, let $\alpha(K)$ be the largest number such that for any compact set $G \subset K$, there exists $v \in G$ for which we have $v - \alpha(K)g \in K$ for all $g \in G$. Then,*

$$\alpha(K)\widehat{\rho}_{\mathrm{Conic},K}(\Sigma) \leqslant \rho(\Sigma) \leqslant \widehat{\rho}_{\mathrm{Conic},K}(\Sigma).$$

For example, the constant $\alpha(K)$ corresponding to the cone $\mathbb{S}^n_+$ of positive semidefinite $n \times n$ matrices has value $\alpha(\mathbb{S}^n_+) = \frac{1}{n}$. The same can be said for cones in $\mathbb{R}^n$ bounded by n hyperplanes passing through the origin.

Note that for an arbitrary set Σ, it is always possible to build a set $\widetilde{\Sigma}$ leaving the cone $\mathbb{S}^n_+$ invariant and satisfying $\rho(\widetilde{\Sigma}) = \rho(\Sigma)^2$. This is done by considering the *semidefinite lifting* where the lifted matrices $\widetilde{M_i}$ act as follows on $n \times n$ matrices (instead of vectors $\mathbb{R}^n$):

$$\widetilde{M_i} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : X \mapsto M_i^T X M_i,$$

or, if we use the vectorized formalism,

$$\widetilde{M}_{i,\mathrm{vec}} : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2} : x = \mathrm{vec}(X) \mapsto (M_i^T \otimes M_i^T)x = \mathrm{vec}(M_i^T X M_i),$$

where $\vec{(}X)$ is a vector formed by stacking the columns of X. When applying this method on the lifted set $\widetilde{\Sigma}$ with respect to the cone $\mathbb{S}^n_+$, the result obtained is equivalent to the ellipsoidal norm approximation.

Norm optimization methods in practice...

The main idea of all these methods is to formulate an approximation of the problem in conic optimization and use efficient solving techniques to obtain the bounds on the joint spectral radius. The implementation of these methods usually makes use of an external optimization solver.

The situation is roughly the opposite of product enumeration algorithms in the sense that norm optimization methods are sometimes able to find tight *upper* bounds, but are unable to provide lower bounds of good quality. However, this approach is usually inadequate for problems of large size. Indeed, the size of these optimization problems may grow very fast and computational limitations may thus become a significant issue since the optimization solver will require a large amount of memory and of computation time. Moreover, with high-dimensional problems, numerical issues tend to appear while solving the optimization problem with numerical solvers.

5.3.3 Extremal norm construction methods

A third approach to joint spectral radius approximation is to explicitly approximate an extremal norm instead of optimizing over a chosen set of matrix norms. Several methods have been proposed by different authors, including Protasov, Guglielmi and Kozyakin. The main idea is to start with some matrix norm or, equivalently, some region corresponding to the unit ball of a norm. Iterative algorithms are then applied in order to produce an adequate sequence of norms that converges to an extremal norm. These algorithms may explicitly work with the norms, or with the corresponding unit balls.

Protasov's method

In [Pro96], Protasov proposes to approximate the unit ball of an extremal norm with a sequence $(\mathcal{P}_i)_{i\in N}$ of polytopes chosen in order to obtain an $(1+\varepsilon)$-approximation of the joint spectral radius after a number of iterations which can be determined a priori, depending on ε and Σ. The first polytope $\mathcal{P}_0$ is taken as

$$\mathcal{P}_0 = \left\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \sum |x_i| \leqslant 1\right\}.$$

At the i^{th} iteration, the new polytope $\mathcal{P}_i$ is obtained as an approximation of the convex hull of the polytopes obtained by applying the different matrices of Σ on $\mathcal{P}_{i-1}$. Note that the computation of a convex hull may be a significant issue when dealing with high-dimensional sets of points.

The approximation of the joint spectral radius is then obtained depending on the largest distance from the origin to the vertices of the last polytope $\mathcal{P}_i$.

Guglielmi et al.'s balanced polytope methods

Another geometric approach involving polytopes has been studied in [GZ09] and in [CGSCZ10]. The method is based on balanced polytopes, which are polytopes $\mathcal{P} \subset \mathbb{K}^n$, with $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$, such that there exists a finite set of vectors $\mathcal{V} = \{v_1, \ldots, v_k\}$ satisfying $\operatorname{span} \mathcal{V} = \mathbb{K}^n$ and:

$$\mathcal{P} = \operatorname{absconv}(\mathcal{V}) := \left\{ x \in \mathbb{K}^n \;\middle|\; x = \sum_{i=1}^{k} \lambda_i v_i \text{ with } \sum_{i=1}^{k} |\lambda_i| \leqslant 1, \lambda_i \in \mathbb{K} \right\}.$$

Here, the abbreviation "absconv" corresponds to *absolutely convex hull*.

Depending on the choice for $\mathbb{K}$, the polytope is called a *balanced real polytope* or a *balanced complex polytope*. A *polytope norm* is any norm whose unit ball is a balanced polytope. It is known [GZ07] that the set of induced matrix polytope norms is dense in the set of induced matrix norms, so (5.5) holds even if we take the infimum on all polytope norms.

In order to find such a polytope norm, the authors present an algorithm that essentially considers the trajectories of a vector $\widetilde{x}$ under all possible products of matrices in Σ. If $\widetilde{x}$ is well-chosen and some hypotheses hold, then the convex hull of the trajectories will describe a balanced polytope, that will give the value of the joint spectral radius. In particular, it is supposed that the set Σ does possess the finiteness property. The vector $\widetilde{x}$ is then taken as a leading eigenvector of a candidate product. Note that this eigenvector may be complex even if $\Sigma \subset \mathbb{R}^{n \times n}$.

The main idea of the algorithm is given below:

- Let P be the candidate product (of length m) and let the set $\widetilde{\Sigma} = \rho(P)^{-1/m}\Sigma$ be a scaled version of Σ. Define the set $\mathcal{V}_0$ as $\mathcal{V}_0 = \{x, \bar{x}\}$, where x is a (possibly complex) leading eigenvector, and let $\mathcal{P}_0 = \operatorname{absconv}(\mathcal{V}_0)$.
- Recursively compute $\mathcal{V}_{k+1} = \widetilde{\Sigma}\mathcal{V}_k$.
- At each iteration, define $\mathcal{P}_k$ as $\mathcal{P}_k = \operatorname{absconv}(\mathcal{V}_k)$. If $\mathcal{V}_{k+1} \subset \mathcal{P}_k$ for some k then terminate the algorithm with the conclusion that the product P is optimal.

More details can be found in [GZ09]. This method requires a good initial guess, but if the candidate product is indeed optimal, then the algorithm stops after a finite number of steps and provides a certificate for the product optimality. In [CGSCZ10], a stopping criterion ($x \in \operatorname{int} \mathcal{P}_k$ for some k) is given so that the iteration can be stopped and the candidate product discarded if it is not optimal. Apart of the requirement of a starting point, the main drawback is that this algorithm usually requires a large amount of computation time, especially with matrices of larger sizes. Efficiently implementing the algorithm may also be nontrivial (see Section 5.5.2). Another issue is that the operations such as the computation of the convex hulls may be subject to numerical inaccuracies or a large memory usage with high-dimensional sets of points.

Finally, it should be noted that improved versions of these balanced polytope methods have recently been presented in [GP11] by Guglielmi and Protasov. The improved algorithms involve a new organization of computations as well as a new stopping criterion.

Kozyakin's Barabanov norm construction methods

Two different iterative schemes are proposed in [Koz10] and [Koz11] for the construction of an extremal norm. More precisely, these algorithms are mainly designed for building Barabanov norms, a special kind of extremal norms, but they also give the joint spectral radius as a byproduct.

In the first algorithm (called the *Linear relaxation iteration* or *LR-procedure*), starting with an arbitrary initial norm on $\mathbb{R}^n$, we build a sequence of norms by using linear combinations of already computed norms. Bounds on the joint spectral radius are available at every iteration. More precisely, the LR-procedure defines a sequence of norms $(\|\cdot\|_k)_{k\in\mathbb{N}}$ according to the following rules:

- Start with a norm $\|\cdot\|_0$ on $\mathbb{R}^n$ and let e be a vector such that $\|e\|_0 = 1$. Let us also choose λ^-, λ^+ such that $0 < \lambda^- \leqslant \lambda^+ < 1$.

- At every iteration, bounds on the joint spectral radius are given by $\rho(\Sigma) \in [\rho_k^-, \rho_k^+]$ with:

$$\rho_k^+ = \max_{x\neq 0} \frac{\max_i \|M_i x\|_k}{\|x\|_k},\ \rho_k^- = \min_{x\neq 0} \frac{\max_i \|M_i x\|_k}{\|x\|_k}.$$

- Let λ_k be in the interval $[\lambda^-, \lambda^+]$ and define the new norm $\|\cdot\|_{k+1}$ as follows:

$$\|x\|_{k+1} = \lambda_k \|x\|_k + (1-\lambda_k)\frac{\max_i \|M_i x\|_k}{\max_i \|M_i e\|_k}.$$

This procedure converges to an extremal norm, and the two sequences $\left(\rho_k^{\pm}\right)_{k\in\mathbb{N}}$ converge to $\rho(\Sigma)$. Alternatively, one can also apply the *Max-relaxation iteration* that replaces the linear combination with a maximum operation and an averaging function μ:

$$\|x\|_{k+1} = \max\left\{\|x\|_k, \frac{\max_i \|M_i x\|_k}{\mu(\rho_k^+, \rho_k^-)}\right\}.$$

In the general case, the averaging function μ has to satisfy

$$\min\{s,t\} \leqslant \mu(s,t) \leqslant \max\{s,t\}, \qquad \forall\, s,t > 0,$$

with equalities only if $s = t$. Common examples are the arithmetic mean, the geometric mean, or the harmonic mean.

The MR-procedure has similar convergence properties as the LR-procedure. More details can be found in [Koz10] and [Koz11].

In practice, producing an efficient implementation of these two methods is quite challenging due to the geometric nature of the operations and the fact that the algorithms are described at a rather high level. One possibility would be to discretize the space but this may reduce the practical accuracy of the algorithms, due to discretization errors. Moreover, this approach can become difficult to manage with matrices of large sizes as the required number of discretization points would grow exponentially. More details about our implementation can be found in Section 5.5.2. This implementation seems to produce acceptable results of limited accuracy provided that the problem size is small enough.

5.3.4 Lifting techniques

Whatever method is used to approximate the joint spectral radius, it is always possible to apply a *lifting* on the set of matrices Σ. This provides better bounds at the price of a higher computational cost, for example when combined with the bounds in Proposition 5.7 or 5.9. The main

idea is to build a set of matrices $\widetilde{\Sigma}$ such that the relation between $\rho(\widetilde{\Sigma})$ and $\rho(\Sigma)$ is known, and then apply the algorithms on $\widetilde{\Sigma}$.

Some examples of liftings are Σ^t, the set of products of length t, $\Sigma^{\otimes t}$, the set of t^{th} Kronecker powers of the matrices in Σ, or $\Sigma^{[t]}$, the set of t-lifts of matrices in Σ (see [BN05, PJ08]). Indeed, it can be proved that

$$\rho(\Sigma)^t = \rho(\Sigma^t) = \rho(\Sigma^{\otimes t}) = \rho(\Sigma^{[t]}).$$

In these examples the lifted set contains either a larger number of matrices, or matrices that have a larger dimension. Note that in [BN05], the authors also propose a recursive approximation method based on successive liftings of the initial set. In theory, this may provide arbitrarily accurate bounds on the joint spectral radius, but the exponential growth of the problem size renders the method intractable except for a very small number of liftings.

5.4 A heuristic approach using a genetic algorithm

The main objective of the algorithm we propose is to provide approximations of the joint spectral radius at a low computational cost. Thanks to Equation (5.4), any product of finite or infinite length produces a lower bound. Hence, the idea of our method is to initially generate a given number of products and then try to combine them in order to obtain new products that can hopefully provide "good" lower bounds.

We have chosen the genetic algorithm framework for these purposes (see Section 2.3.1). Although there will be no guarantee on the quality of the bounds returned by the algorithm, this heuristic approach does not require expensive computation steps, in contrast to classical methods. Note that the results can also be combined with another algorithm, e.g., a method that checks if a given candidate is optimal, or that returns an upper bound on the joint spectral radius.

When applying the genetic algorithm to the joint spectral radius, each element of the population P will naturally correspond to a product of matrices. The different steps of the algorithm are presented below.

Initialization

Evaluate the spectral radius of all products of matrices of length at most κ, where κ is chosen depending on $|\Sigma|$ in order to limit this step to a small number of products. The best spectral radius provides an initial lower bound on the joint spectral radius of Σ. Let $K = 2\kappa$ be an initial bound on the length of the products and create a population P_0 of fixed size S. Each element of the set P_0 is thus a product of matrices $M_{i_1} \dots M_{i_k}$, satisfying $k \leqslant K$. We suppose that the population size is at least 10, as tiny populations are not really meaningful for this approach. Indeed, the idea is to combine a large number of different solutions to produce new ones. Of course, it would be possible to use a smaller population but several choices mentioned below will have to be modified as the number of population members produced at each generation may exceed the chosen population size with the parameters presented below. For example, in a default setting, we keep at least the two best solutions of the current population and introduce at least 4 new random products in the new population. This would be impossible to satisfy if the population size is less than 6.

Evaluation and controlled mutation

At each generation g, the spectral radius of all products of the corresponding population P_g is evaluated in order to obtain lower bounds on the joint spectral radius $\rho(\Sigma)$. If this improves the current lower bound on $\rho(\Sigma)$, the algorithm explores the neighborhood of the corresponding product and tries to obtain a better lower bound.

More precisely, we are using the Levenshtein neighborhood which corresponds to all products that can be obtained with a single deletion, substitution or insertion of a matrix in the candidate product. We only keep products of length of at most K. If an improvement is possible, then the corresponding product is inserted into the population, replacing the worst product in the set. In fact, all these operations may also be interpreted as a controlled mutation on the best element of P_g.

Selection

In order to obtain the next population P_{g+1}, the current population P_g is partitioned into several subsets. The n_A best products are kept

unchanged and the n_B worst population elements are discarded and replaced by randomly generated products, for some values of n_A and n_B. This elitist strategy allows the most promising products to keep producing offsprings, while new random elements are continuously inserted in order to ensure diversity. The remaining products are obtained using crossover operations so that the new population still has a size of S. In our algorithm, $n_A \in \{2, 3\}$ depending on the population size, and n_B has been chosen to $\max\{4, \lfloor \frac{S}{50} \rfloor\}$.

Crossover

A first crossover operation generates a bunch of products following the pattern $M_{a_1} \dots M_{a_c} M_{b_{c+1}} M_{b_k}$ for some value of c, and where both products $M_{a_1} \dots M_{a_k}$ and $M_{b_1} \dots M_{b_k}$ are among the 50% best products in P_g. This corresponds thus to swapping operations between two products. The number of elements of the next population P_{g+1} obtained using this first crossover is $\lfloor \frac{S}{2} \rfloor$.

A second crossover gives products of the form $M_{i_1} \dots M_{i_k}$. For each product, the matrix M_{i_j} at position j corresponds to either M_{a_j} or M_{b_j}, where $M_{a_1} \dots M_{a_k}$ and $M_{b_1} \dots M_{b_k}$ are parent products present in P_g. This corresponds thus to mixing operations between two elements. The number of products produced by this crossover is such that the total population still has a size of S.

Random mutation

A mutation operation is applied on the new population P_{g+1}: each element has a given probability μ to be mutated. If this is the case, then some matrices in the product are replaced by matrices randomly chosen in Σ. The number of modified positions can be fixed but it can also depend on the value of K, e.g., if we choose to modify x% of the positions. In our simulations, the following values have been used: $\mu = 0.3$, $x = 20$.

Stopping criterion

If there is no improvement on the lower bound during T_1 generations, then the maximum allowed length K is increased. Indeed, this could mean that the lower bound is optimal, but also that the optimal product

has a length that is larger than the current value of K. If the best lower bound continues to stall and the total number of generations reaches another specified threshold $T_2 > T_1$, then the algorithm terminates and returns the best lower bound found.

Comments

To summarize, the algorithm considers a set of S products of length at most K, with K slowly increasing during the computation. New candidates are generated by heuristically combining existing products according to their performance. The current lower bound is updated with each new product and the algorithm terminates if there is no improvement during several iterations. The generation of new candidates is mainly done by the crossover steps, and some random perturbation may be applied at each generation to ensure an exploration of the search space.

Note that during this process, a significant amount of time is spent building products of matrices and evaluating their spectral radii. Depending on the size of the matrices, it may sometimes be useful to store these intermediate results and retrieve them instead of rebuilding the products from scratch. This cache can be built at the initialization step when all products of matrices of length at most κ are built.

One particular characteristic of the algorithm is the number of parameters. This is a priori a disadvantage as it is difficult to find the optimal values for these parameters, but it can also be seen as an advantage as this tuning allows us to choose between a very low computation time, a large exploration of the search space, or some trade-off between time and quality of the bound. The most important parameters are S, the size of the population, and (T_1, T_2), the stalling thresholds. Indeed, increasing the size of the population leads to a better exploration of the search space, with a roughly linear increase of the computation time.

Independently, the stalling thresholds determine the duration of this exploration. One should avoid a threshold T_1 that is too low as the breeding process may not have enough time to reach a candidate in the neighborhood of an optimal product, while a very high value is useless if the length of the shortest optimal product is higher than the current maximum length K. At the same time, T_2 should be large enough,

relatively to T_1, in order to allow several increases of K while stalling. A smaller value produces a faster but of course less reliable algorithm.

The main drawback of this algorithm is obviously the absence of guarantee on the quality of the bound. However, the product associated to this bound can be used as candidate product in other geometric algorithms described in Section 5.3. Indeed, as previously mentioned, there exist several methods able to derive upper and lower bounds on the joint spectral radius by approximating an extremal norm, but that require a good starting point, i.e., a potentially optimal product. If this product is indeed optimal then both bounds converge to the value of the joint spectral radius; if it is suboptimal then this can be detected and a better product is produced and used as a new starting point. These methods are thus well-suited as a post-processing step to our algorithm as their iterations typically have a much higher computational cost, and one wants to avoid having to reset the algorithm with a new starting point a large number of times. Another possibility is to compute a small number of iterations of Gripenberg's algorithm to obtain rough upper bounds on the joint spectral radius if a geometric approach is too expensive.

5.5 Experimental analysis

In practice, the performance of all the methods that we have described in Section 5.3 varies widely. A branch-and-bound method such as Gripenberg's algorithm provides an interval containing the value of the joint spectral radius, but the computation time becomes prohibitive when a small interval is desired due to the growth of the number of products to consider. The same thing can be said about the pruning algorithm, when the matrices are nonnegative. Optimization methods based on the ellipsoidal norm or the sum-of-squares approximations mainly give an upper bound, and even if they may find the exact value in some cases, the size of the semidefinite optimization problem becomes huge when one tries to lift the matrices to improve the bounds in Proposition 5.7 or to increase the degree d in Proposition 5.9. The computation time increases rapidly and numerical problems may become a significant issue. Geometric algorithms such as the LR-procedure or Protasov's method require the manipulation of geometric objects. The accuracy of the results may thus be significantly influenced by numerical issues such as discretization errors.

Problem size	$n = 2$	$n = 4$	$n = 8$
$\|\Sigma\| = 2$	100 sets	100 sets	—
$\|\Sigma\| = 3$	100 sets	100 sets	—
$\|\Sigma\| = 4$	100 sets	100 sets	100 sets
$\|\Sigma\| = 8$	—	—	100 sets

Table 5.1: *Characteristics of the randomly generated sets of matrices. Each instance corresponds to a set* Σ *of* $n \times n$ *matrices whose entries have been randomly chosen between* -5 *and* 5 *using a uniform distribution.*

5.5.1 Details about the test sets

The different methods presented in the previous section have been tested on a large number of sets of matrices. Most examples are randomly generated matrices but the test set also includes instances obtained from applications such as the computation of the capacity of codes subject to forbidden difference patterns [MOS01]. Indeed, the computation of a capacity may involve very large sets of matrices and/or high-dimensional matrices. More information about the random generation may be found in Table 5.1.

In the case of 2×2 and 4×4 matrices, an optimal periodic product reaching the exact value of the joint spectral radius has been found for every test case. As the computational cost increases with the size of the matrices, obtaining the same result for *every* 8×8 matrix in the test set was not possible due to a high computational cost. Note that all these results have been obtained using either the JSR Toolbox (more details in Section 5.5.3), or our genetic algorithm, combined with a certification algorithm as post-processing step.

The exact value of the joint spectral radius allows us to compute the actual approximation errors for the different algorithms. The minimal period associated to the different optimal products for a given set Σ may be considered a rough measure of the difficulty of the problem. In Table 5.2, the different sets of matrices are classified according to their minimal period. Note that even with randomly generated matrices, the test set contains nontrivial instances, e.g., several sets of two 2×2 matrices with an optimal product whose length is more than 20. The smallest minimal period is 1 and the largest is 32.

Problem size		Minimal period				
		[1; 2]	[3; 5]	[6; 9]	[10; 19]	20+
$n = 2$	$\lvert\Sigma\rvert = 2$	90%	5%	2%	1%	2%
	$\lvert\Sigma\rvert = 3$	87%	11%	1%	1%	—
	$\lvert\Sigma\rvert = 4$	77%	20%	3%	—	—
$n = 4$	$\lvert\Sigma\rvert = 2$	74%	17%	6%	3%	—
	$\lvert\Sigma\rvert = 3$	54%	25%	12%	6%	3%
	$\lvert\Sigma\rvert = 4$	63%	20%	13%	3%	1%

Table 5.2: *Classification of the different instances according to the minimal period associated to the optimal products.*

5.5.2 Details about the implementations

The approximation methods presented in Section 5.3 have been implemented using MATLAB® 7.4.0.336 (R2007a) and the computations have been performed on an Intel® Core™ 2 Quad Q9300 at 2.50 GHz with 3 GiB RAM, running Ubuntu Linux 10.04 LTS. The solver used for the SDP problems is SeDuMi 1.3. For Gripenberg's method, the implementation used in the tests is the one written by G. Gripenberg[1]. This implementation allows the use of an accuracy threshold and/or a limit on the number of evaluations of norms and eigenvalues.

Gripenberg's algorithm

Since the computations are performed with limited numerical accuracy, several issues may arise in practice. For example, as the upper bound in Gripenberg's method converges quite slowly in general, very long products may be required in order to reach the desired accuracy on the approximation of the joint spectral radius. However, if these products are too long, the implementation may fail due to overflow problems. This indeed happens for a small but non-negligible fraction of the test set, especially with sets containing only two small matrices.

More precisely, given the same amount of computational resources, this problem is expected to happen less often for larger sets of matrices since the number of different products of a given length would be much larger. Hence, the maximal reachable product length given the

[1] available at `http://math.tkk.fi/~ggripenb/ggsoftwa.htm`

same computational resources would be lower. This will be analyzed in Section 5.6 (more precisely, see Table 5.4 in Section 5.6.3).

One possible way to avoid these overflow problems is to rescale the matrices as often as required during the computation, in order to keep the growth of the matrices under control. This is done in the implementation of Gripenberg's algorithm in the JSR Toolbox.

Norm optimization methods

Norm optimization methods have been implemented as direct translations of the SDP problems described in the literature into SeDuMi format. The optimization problem is solved using SeDuMi, which is based on primal-dual interior-point methods and is freely available at `http://sedumi.ie.lehigh.edu`.

When doing an iteration of the bisection method, the description of the SDP problem (in SeDuMi format) is updated instead of being recomputed. Hence, there is only a single construction of the full problem, which is done at the first iteration. Note that the tolerance threshold of the SDP solver is left as a parameter.

Guglielmi et al.'s balanced polytope methods

For the balanced polytope algorithm, we mainly follow the description given in [GZ09]: a first remark is that we do not really need to compute $\mathcal{V}_{k+1} = \widetilde{\Sigma}\mathcal{V}_k$ at each iteration: it is possible to prune vectors from the set. This is done by constructing a so-called *essential system of vertices* of $\mathcal{P}_k$, which corresponds to a minimal set of vertices generating $\mathcal{P}_k$ via absolutely convex combinations. The minimality of the set is not required, but it lets us keep a smaller set of vectors during the computation.

The computation of this essential system of vertices can be done via a convex hull operation in the real case (which is done by Qhull [BDH96] with the function `convhulln` in MATLAB®), or by solving a quadratic real program in the complex case, by separating the real and imaginary parts. Testing whether all vectors in $\mathcal{V}_{k+1}$ are inside the polytope $\mathcal{P}_k$ can be done by solving a linear program in the real case, or again a quadratic real program in the complex case, as we already have an essential system of vertices describing the polytope $\mathcal{P}_k$. In our implementation, we are

using the MATLAB® general functions `linprog` and `quadprog` but other solvers can be used.

Kozyakin's Barabanov norm construction methods

In general, geometric algorithms are subject to more implementation issues since most geometric operations, e.g., the computation of convex hulls, are done with limited precision. Describing objects such as polytopes or ellipsoids is possible, but representing the unit ball of an arbitrary norm may be problematic. In these cases, the geometric objects are usually approximated using some kind of discretization.

This is the case in our implementation of the LR and MR-procedures. This may lead to two practical problems. On the one hand, as the dimension of the space increases, one would obviously need an exponentially increasing number of points, which is not always reasonable. On the other hand, one has to choose the position of the sampling points. Hence, while a uniform sampling of a circle (in a space of dimension 2) is easy to perform, the general problem of uniformly sampling an n-dimensional hypersphere is difficult.

A possible solution is to use a random sampling with uniform distribution on the hypersphere but then, the number of points has to be increased even more in order to obtain good approximations with high probability. In our implementation, we have chosen a uniform random sampling by generating vectors using independent standard gaussian distributions for every component, then rescaling them in order to obtain unit norm vectors [Knu97].

The different unit balls are then managed by keeping the values of the norms of the discretization points. The effects of the matrices acting on these discretization points are pre-computed before the first iteration. Finding the nearest discretization point is done via the MATLAB® function `dsearchn`, which also uses Qhull. Of course in the two-dimensional case, we can simply use a uniform sampling of the circle and the problem of finding the nearest discretization point becomes trivial.

5.5.3 The JSR Toolbox

The JSR Toolbox [VHJ+11] is a MATLAB® toolbox that provides a large set of methods for the approximation of the joint spectral radius, but also includes several helper functions, for example for comparison or analysis purposes, and several demonstration functions. One important feature is that the toolbox offers a "default" algorithm that computes bounds on the joint spectral radius by combining several approaches presented in this article. This may be useful if one does not know which algorithm is more suitable. The behavior of this algorithm may also be parameterized as needed, for instance by setting a maximal computation time or by fine-tuning a particular step in an algorithm.

The main steps of the default algorithm are the following:

- Try to transform the problem into a set of smaller independent problems. This is possible when the matrices in the set Σ are simultaneously block-triangularizable.

- If the matrices are nonnegative, start with the pruning algorithm in order to get some bounds $[\beta^-, \beta^+]$ on the joint spectral radius, then compute the joint conic radius, using the positive orthant as cone, and the ellipsoidal norm approximation using $[\beta^-, \beta^+]$ as initial bounds.

- If some matrices have negative entries, start with a variant of Gripenberg's algorithm in order to get some initial bounds and a candidate product. This variant may rescale the matrices during the computation in order to avoid overflows. After this first step, compute the ellipsoidal norm approximation and, if needed, try to certify optimality or to find a better candidate product using a balanced complex polytope method or a conitope method, which is a lifted polytope method.

The results corresponding to this algorithm will also be included in the following sections for comparison purposes. The parameters will be left at their default values, except for the allowed computation time per instance.

Problem size	$\lvert\Sigma\rvert = 2$	$\lvert\Sigma\rvert = 3$	$\lvert\Sigma\rvert = 4$
$n = 2$	$\sim 99\%$	$\sim 99\%$	$\sim 100\%$
$n = 4$	$\sim 99\%$	$\sim 97\%$	$\sim 96\%$

Table 5.3: *Percentage of the test instances where the genetic algorithm was able to find an optimal product.*

5.6 Results

In this section, we present numerical results obtained from our experiments. We begin by looking at the behavior of our genetic algorithm. After that, we proceed with the comparison between the different methods, including the default algorithm of the JSR Toolbox and the genetic algorithm.

5.6.1 Accuracy of the genetic algorithm

As there is no a priori guarantee on the quality of the bounds obtained by the algorithm, it is interesting to examine the practical quality of the results. Table 5.3 shows the results for sets of 2×2 and 4×4 matrices. The optimality may be certified when it is possible to obtain an upper bound matching the lower bound provided by the genetic algorithm, or when the candidate has been successfully certified as optimal in the post-processing step with another algorithm. We recall that these geometric algorithms may have an excessive computational cost in some cases, hence it is not always practical to check the optimality of such a candidate, for example in several 8×8 instances.

The values of the parameters of the genetic algorithm used to obtain the results shown in Table 5.3 are $S = 150$ for the population size, and $(T_1, T_2) = (15, 100)$ for the two thresholds. This choice has been done so that the algorithm runs in about one or two seconds for each set of matrices. Of course, increasing the values of the parameters would lead to better results in general as more products would be tested, but these results show that it is often possible to obtain an optimal product in a small amount of time even for sets of four 4×4 matrices, and the combination with geometric algorithms may be interesting as the problem of choosing an initial product would be taken care of.

5.6.2 Influence of the main parameters of the genetic algorithm

Figure 5.1 shows how the performance of the algorithm varies when the parameters are modified. Each point corresponds to the average result of 25 runs on a set of 600 instances (sizes 2×2 and 4×4 in Table 5.1), with the same values for the three main parameters: S, the population size, T_1, the stalling threshold before increasing the maximum allowed product length, and T_2, the stalling threshold before aborting the algorithm.

Note that the computation time may sometimes decrease when the population size increases. Indeed, even though each iteration is more expensive, the total number of iterations may be smaller if an optimal product is found faster. The performance might also slightly decrease when the population size increases, but the difference is not significant and may simply be due to the randomness of the method.

A first global observation is that the success rate is significantly high even if the amount of allowed computation time is very small compared to usual methods (less than one second per set of matrices in average for the rightmost part of the figure). This is consistent with the main objective of the algorithm: being able to provide good lower bounds in a short amount of time. Even though these results have been averaged over all sizes of sets and matrices in the test set, the worst success rate is still at about 87%, which is reached for sets of four 4×4 matrices, with $S = 10$ and $(T_1, T_2) = (5, 50)$. In this case and for this set of parameters, the average computation time is less than 0.1 second per set of matrices.

When looking at the effects of the variation of the T_1 parameter, it appears that for sufficiently large population sizes, $T_1 \in \{10, 15, 20\}$ gives roughly equivalent results. The value $T_1 = 5$ seems to be sometimes inadequate and gives results of lower quality. This is due to the fact that the algorithm may require several iterations in order to find a good combination of products, and extending the search space after only 5 stalling iterations may be too early.

Taking $T_2 = 100$ instead of $T_2 = 50$ decreases the failure rate by about $\frac{1}{3}$ while increasing the computation time by a factor of about 2, as expected. In fact, the choice for the values of S and T_2 mainly results from a compromise between computation time and performance, depending on the needs. The combination used in the Section 5.6.1 ($S = 150$ and $(T_1, T_2) = (15, 100)$) gives an algorithm that is often optimal while running in a reasonably short amount of time, at least on

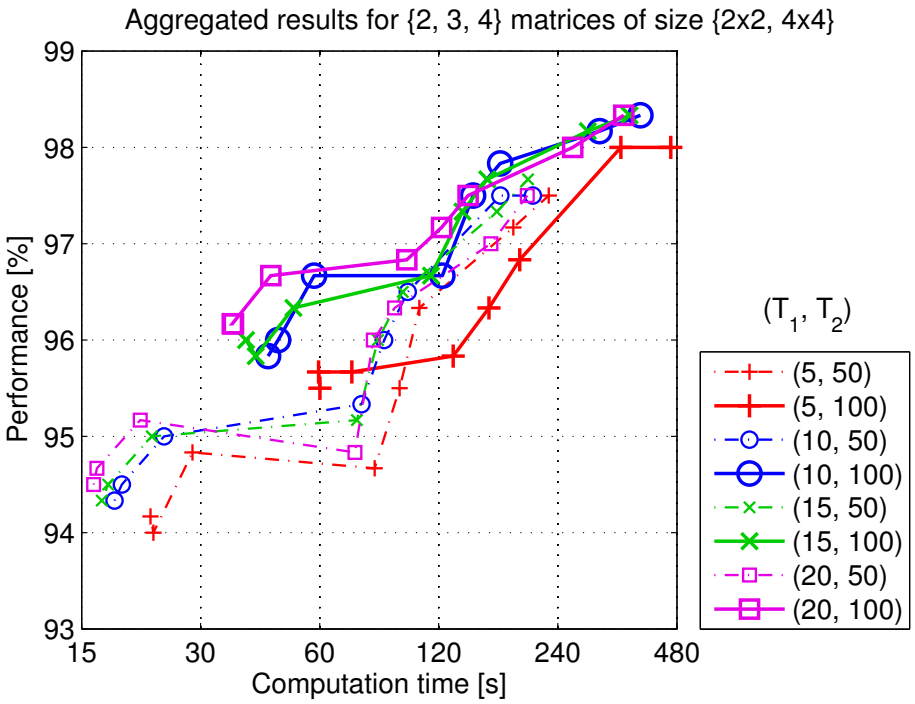


Figure 5.1: *Performance and computation time for different parameter values of the genetic algorithm. The performance corresponds to the average percentage of the test cases where the final bound returned by the algorithm matches the value of the joint spectral radius. The computation time corresponds to the average total amount of time required to process the 600 sets of matrices and is represented using a logarithmic scale. Each line represents the results obtained with one pair of values* (T_1, T_2)*, and the markers on each line are associated to the following population sizes: 10, 15, 20, 30, 40, 50, 100, 150.*

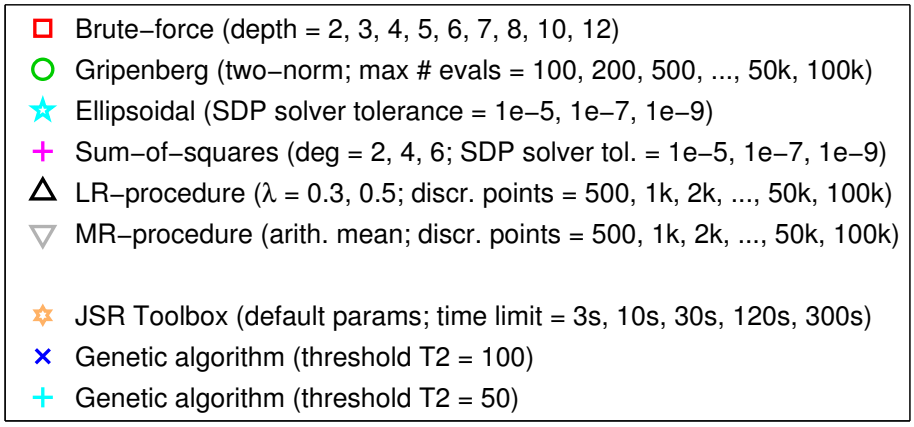


Figure 5.2: *Legend associated to the tested methods. For the genetic algorithm, the other parameters have been tested on the following ranges:* $T_1 \in \{5, 10, 15, 20\}$ *and* $S \in \{10, 15, 20, 30, 40, 50, 100, 150\}$. *This legend is related to Figures 5.3, 5.6 and 5.7.*

this test set. For example, if the product found by the genetic algorithm is to be used as an initial step of another method that tries to prove its optimality, choosing a very large value for the population size may be more adequate. Indeed, it may take a large amount of time to detect the suboptimality of a candidate using the second method whereas the genetic algorithm may find a better product in a shorter amount of time.

5.6.3 Comparison to the other algorithms

In order to compare the different methods, the performances of a chosen subset of methods is shown in Figures 5.3, 5.6 and 5.7. All these figures show the number of test cases where the algorithm was able to find a lower bound (left column) or upper bound (right column) close enough to the value of the joint spectral radius, and the average computation time required to obtain a result for a single set of matrices with the corresponding sizes. The common legend can be found in Figure 5.2.

The algorithms used for the comparison are the following. First, the red squares correspond to a naive brute-force method, i.e., the evaluation of all products of length at most t, with $t \in \{2, 3, 4, 5, 6, 7, 8, 10, 12\}$, each square corresponding to one value of t, in this order from the left

to the right. In Figure 5.7, values with $t > 8$ are not displayed as the computation time is then too large. The upper bounds are obtained using (5.4) with the standard two-norm.

Green circles are associated to Gripenberg's original branch-and-bound algorithm, using the two-norm and a limitation on the number of evaluations of norms and eigenvalues. The maximal number of evaluations is set to a value among $\{100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000\}$. More precisely, each set of matrices has been tested with each one of these values. In the different figures, each green circle is associated to one of these values. Hence, the left-most green circle corresponds to the run with maximum 100 evaluations as it is the fastest, the second one corresponds to 200 evaluations, etc.

The ellipsoidal norm and sum-of-squares approximations (with $d \in \{1, 2, 3\}$) are represented by cyan pentagrams and magenta plusses ('+' symbol), respectively. As these techniques involve solving a sequence of semidefinite optimization problems, three tolerance values have been tested for the SeDuMi part: $10^{-5}, 10^{-7}, 10^{-9}$. Indeed, this may influence the computation time and the result, for example if numerical problems are encountered. The initial bounds associated to the bisection method are obtained by taking the maximal spectral radius and the maximal two-norm among all matrices in the set.

The points associated to LR and MR-procedures are displayed using black and gray triangles. The parameters $\lambda^{\pm}$ for the Linear relaxation iteration have been chosen as $\lambda^{+} = \lambda^{-} \in \{0.3, 0.5\}$. For the Max-relaxation iteration, the chosen averaging function correspond to the arithmetic mean $\mu(x, y) = \frac{x+y}{2}$. Random uniform sampling is used except for 2×2 matrices, where the natural deteministic uniform sampling of the circle is used. The algorithms have been tested with all the following values for the number of discretization points: $500, 1000, 2000, 5000, 10000, 25000, 50000, 100000$. Moreover, as the result may vary due to the random component, each run has been repeated 25 times. The corresponding results have been averaged over these 25 repetitions.

The results obtained by the JSR Toolbox are shown using orange hexagrams. All parameters have been kept at their default values, except for the maximal allowed computation time. This limit has been set to 3, 10, 30, 120 and 300 seconds, so that bounds are obtained in a reasonable time.

Finally, the genetic algorithm has been run with the following com-

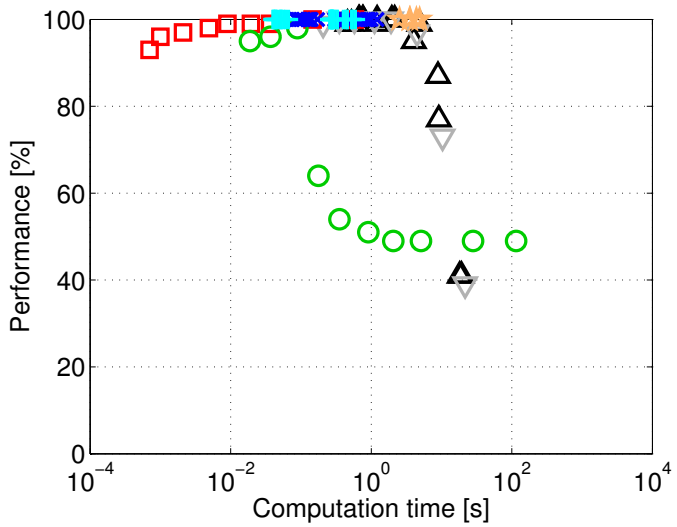
binations of parameters: the values used for the population size are 10, 15, 20, 30, 40, 50, 100 and 150, and the values used for the T_1 threshold are 5, 10, 15, 20. The corresponding points in the figures are the blue crosses and cyan plusses.

Guglielmi's balanced polytope methods are not represented in these figures. Indeed, they correspond more to certification algorithms used to check whether a *given* candidate product is optimal. This means that their performance would heavily depend on the choice of this starting point, especially when we consider the amount of computation time. Nevertheless, it should be noted that such certification algorithms are implicitely used by the JSR Toolbox (see Section 5.5.3). For example, in our experiments, they allowed us to find the exact value of the joint spectral radius by producing optimality certificates for given products.

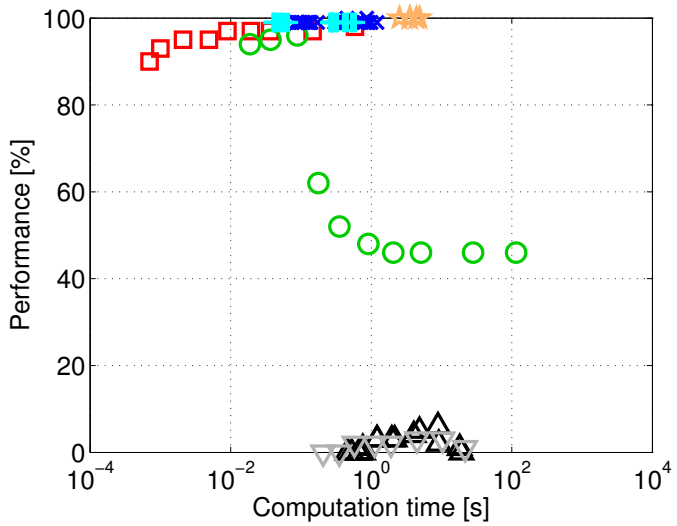
Sets containing two 2×2 random matrices

A first comparison of the behavior of the different methods can be seen in Figure 5.3. This corresponds to a set of low-dimensional test cases as we have $n = 2$ and $|\Sigma| = 2$, which can be considered "easy". Indeed, a simple brute-force method is able to quickly reach an optimal lower bound in nearly all cases, which is consistent with Table 5.2. As expected, the upper bounds converge very slowly. Even with products of length 12, the error is higher than 10^{-2} more than 40% of the time and an accuracy of 10^{-6} is never reached. One might hope that this can be improved by using a branch-and-bound technique, such as Gripenberg's algorithm. Indeed, the figure shows that better upper bounds are obtained, at least when the number of evaluations is low. There is no clear improvement for lower bounds as the brute-force approach was already nearly optimal.

When the number of allowed evaluations is increased, the results show a significant drop in performance, which can be explained by overflow problems. Indeed, since the problem size is very small, a large number of evaluations corresponds to the evaluation of very long products, and the algorithm fails as it is unable to manage the corresponding matrices. Hence, increasing the size of the products may not be a good idea in practice *in this case* as this problem occured for a large number of instances. Fortunately, the issue is expected to happen much less often with instances of larger sizes (see Figures 5.6 and 5.7). Nevertheless, in order to avoid these numerical issues with small sets of matrices, one

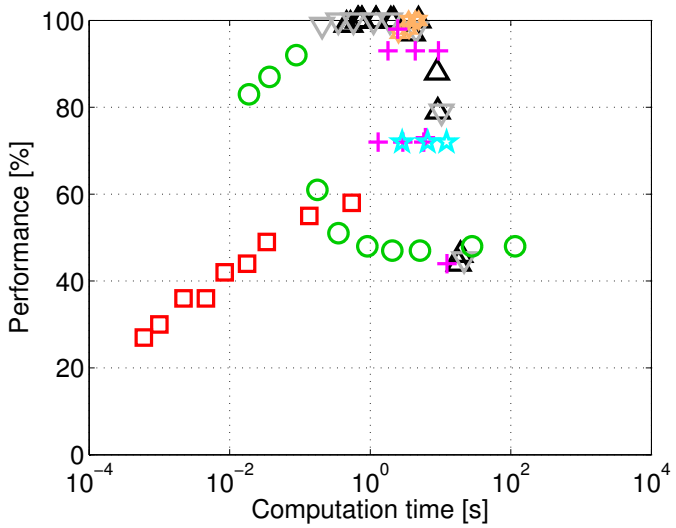


(a) Lower bounds, tolerance of 10^{-2}

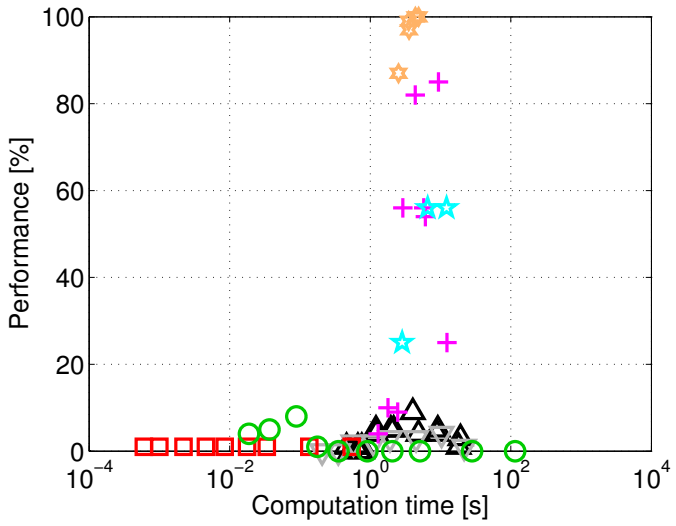


(b) Lower bounds, tolerance of 10^{-6}

Figure 5.3: *Results on small test instances (sets of two 2×2 random matrices). The performance of an algorithm corresponds to the number of test cases where it was able to find a bound that is within a given tolerance of the exact value of the joint spectral radius. The computation time corresponds to the average time required by the algorithm to produce a result and is represented using a logarithmic scale.*



(c) Upper bounds, tolerance of 10^{-2}



(d) Upper bounds, tolerance of 10^{-6}

Figure 5.3: *(continued).*

Max number of evaluations	500	1000	2000	5000	50000
Sets of two 2×2 matrices	—	34	44	48	50
Sets of four 2×2 matrices	—	—	29	49	53
Sets of four 4×4 matrices	—	4	8	11	12
Sets of four 8×8 matrices	—	—	—	1	1

Table 5.4: *Number of instances of the given sizes where Gripenberg's original algorithm had overflow problems, depending on the maximal number of evaluations. Each of the four categories of test cases contains 100 instances.*

possible solution is to modify the algorithm by using successive rescalings of the matrices so that the products remain tractable. This is done in the branch-and-bound exploration in the first step of the JSR Toolbox. Table 5.4 shows the number of instances where the original version of Gripenberg's algorithm had overflow problems, for several test sets. Note that in this table, there is no reduction in the number of overflows between sets of two 2×2 matrices and sets of four 2×2 matrices, for 5000 and 50000 evaluations. This can be explained by the fact that in these cases, 5000 evaluations were already enough to produce products that are too long in most affected instances. There is still a reduction of the number of overflows before reaching this number of evaluations.

Optimization methods such as the ellipsoidal norm or the sum-of-squares approximations are able to obtain a much better accuracy, but only for a subset of all test cases. Indeed, these algorithms may directly reach the exact value of the joint spectral radius in a non-negligible number of cases, in contrast to product enumeration algorithms. This can be seen in the bottom right part of the figure as they are indeed the only methods (excluding the JSR Toolbox) that have been able to obtain upper bounds within a tolerance of 10^{-6}. Note also that even though the sum-of-squares approximation is supposed to give better bounds than the simple ellipsoidal norm approximation, this is not always the case in practice because of the fact that increasing the degree of the polynomials also increases the chances for the optimization solver to run into numerical problems.

Comparing the results for the two tolerance values also shows that the LR and MR-procedures are able to obtain upper and lower bounds most of the time, but only with a limited accuracy due to discretization

errors. Indeed, the large number of points required to reach a tolerance of 10^{-6} would imply a correspondingly large computation time. In order to avoid this drawback, algorithms based on successive approximations of (unit balls of) norms would have to restrict themselves to subsets of norms, e.g., polytope norms.

The JSR Toolbox seems to be able to reach the exact value in all cases for this test set. This is due to the fact that most sets of matrices have a short optimal product that can be found by the branch-and-bound method. Although the upper bounds obtained by this first step are still weak, the polytope or conitope algorithm directly tries to prove the optimality of the candidate. This explains the performance in the upper bounds subfigures. Note that in these simple cases, the total computation time is a bit higher than the other algorithms with a high performance rating since with the standard parameters, three different methods are launched successively.

Finally, the genetic algorithm does its job well: very good lower bounds are indeed obtained in a short amount of time. In fact, with this first set of instances, the bounds obtained by the genetic algorithm are all optimal or close to optimal, even though there was no guarantee on the quality of the solution.

Figure 5.4 shows the quality of the upper bounds found by the algorithms when considering different tolerance values. For each algorithm, one representative has been selected. These representatives have been chosen so that their computation times are roughly the same. More precisely, the computation times are all around 7 seconds in this case. This allows most algorithms to be close to their best performance (see Figure 5.3). The results shown in Figure 5.4 support the observations given in the previous paragraphs. This is expected as Figure 5.4 is simply another view of what happens between the two subfigures in the right column of Figure 5.3.

Note that for the lower bounds, Figure 5.3 already shows that except for Kozyakin's geometric algorithms, there is little to no difference in the performance ratings when comparing the two tolerance values. Therefore, a performance profile such as the one in Figure 5.4 would not give much more information in this case: an optimal product has been reached.

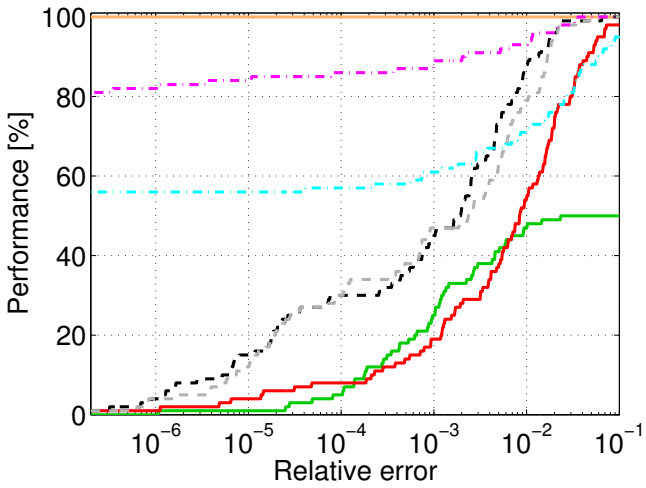


Figure 5.4: *Performance of upper bounds found by the different algorithms on sets of two 2×2 random matrices, with respect to the relative error tolerance. Parameters have been chosen such that the computation times are around 7 seconds. The colors correspond to the legend in Figure 5.2: solid lines are associated to the JSR Toolbox (orange, top), Gripenberg's algorithm (green, bottom) and the naive brute-force method (red, middle); dash-dot lines are the sum-of-squares (magenta, top) and the ellipsoidal norm (cyan, bottom) approximations; dashed lines are the LR-procedure (black) and MR-procedure (gray).*

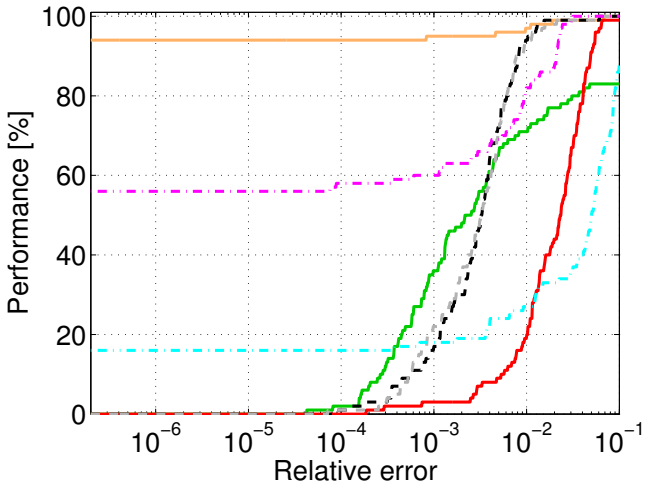


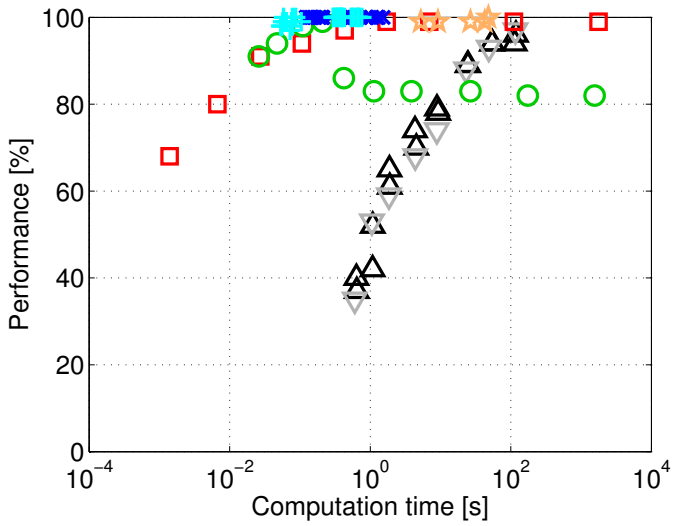
Figure 5.5: *Performance of the different algorithms on sets of four* 4×4 *random matrices, with respect to the relative error tolerance. Parameters have been chosen such that the computation times are around 50 seconds. The colors follow the legend in Figure 5.2: solid lines are, from the left to the right, the JSR Toolbox (orange), Gripenberg's algorithm (green) and the naive brute-force method (red); dash-dot lines correspond to the sum-of-squares (magenta, top) and the ellipsoidal norm (cyan, bottom) approaches; dashed lines are the LR- (black) and MR-procedures (gray).*

Sets containing four 4×4 random matrices

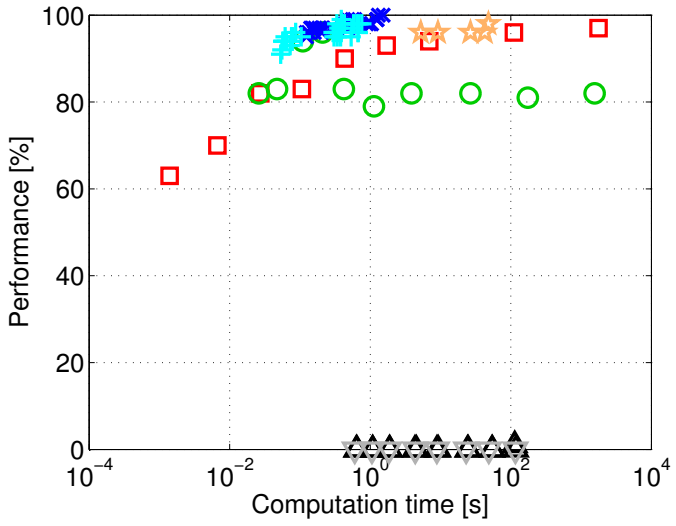
Figure 5.6 summarizes the results for a set of larger problems, which allows us to observe the influence of the size of the problem on the results. The counterpart of Figure 5.4 is also presented as Figure 5.5, with an average computation time of about 50 seconds.

As expected, the general performance tends to be weaker, while the computation time is higher. Still, several interesting observations can be made. First, the improvements of Gripenberg's algorithm with respect to a naive brute-force approach begin to appear, at least when there are no overflow issues. The fact that its performance increases when comparing Figures 5.4 and 5.5 is due to the fact that overflows only happened in a small number of instances in this case.

The upper bounds are still quite weak, but this also holds for most of the other algorithms: for example, the performance of the ellipsoidal

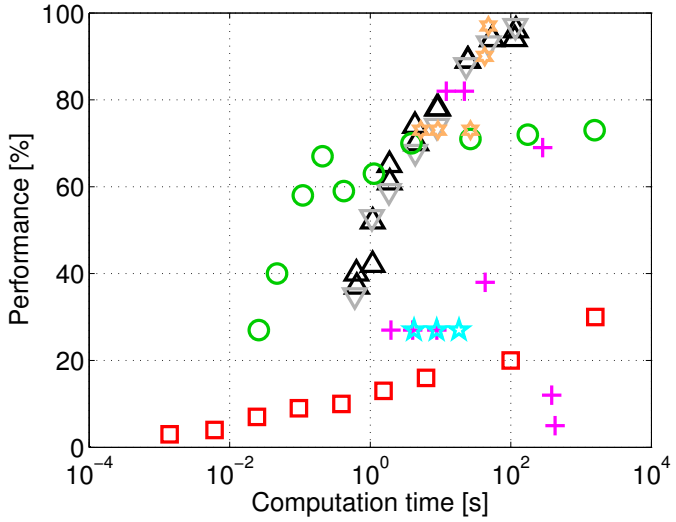


(a) Lower bounds, tolerance of 10^{-2}

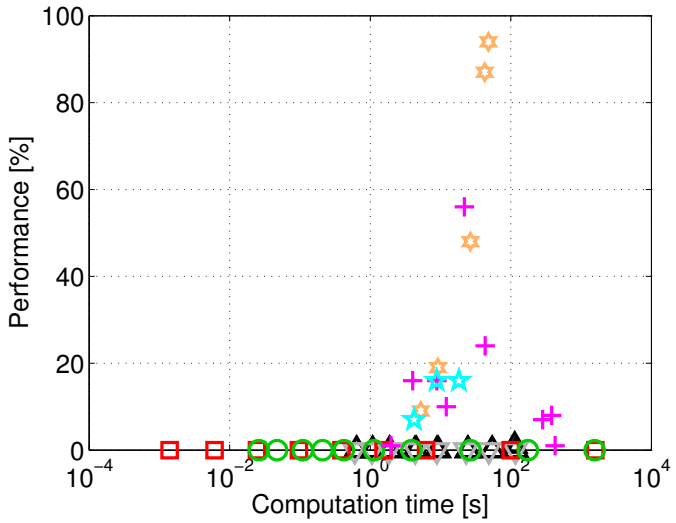


(b) Lower bounds, tolerance of 10^{-6}

Figure 5.6: *Results for sets of four* 4×4 *random matrices. The performance measure is the same as in Figure 5.3.*



(c) Upper bounds, tolerance of 10^{-2}



(d) Upper bounds, tolerance of 10^{-6}

Figure 5.6: *(continued)*.

norm approximation method is also dropping by a large factor compared to the 2×2 case. In fact, the only method that is still able to obtain tight upper bounds around the exact value with nearly all instances is the algorithm of the JSR Toolbox. Note that the success rate shown in the figure is a bit less than 100%. In fact, it is possible to reach a performance rating of exactly 100%, by increasing a little bit the threshold on the computation time. Indeed, optimal products for all 600 test sets with $n \in \{2, 4\}$ are known thanks to results obtained with this algorithm.

The implementations of the LR and MR-procedures are also close to their limits as a large number of discretization points are required even for a tolerance of 10^{-2}. Of course, the threshold at 10^{-6} is unreachable in a reasonable time. Note also that in practice, for an arbitrary set of matrices, it is even possible to obtain an interval that does *not* contain the actual value of the joint spectral radius due to discretization errors and thus depending on the quality of the discretization. As we need an exponentially growing number of points when the size of the problem increases, it is clear that this approach does really not scale well to high-dimensional problems.

This last remark is also valid for optimization-based methods. Indeed, Figure 5.6 shows that the ellipsoidal norm approximation and thus the sum-of-squares approximation of degree 2 are now inadequate in most of the test cases and even increasing the degree of the polynomial does not always yield much better results in the general case.

If we concentrate on the lower bounds, Figure 5.6 also shows that both the genetic algorithm and the combined algorithm of the JSR Toolbox are able to find an optimal product in nearly all cases. Of course, the genetic algorithm is faster as expected, but the JSR Toolbox is able to provide a certificate of the optimality of the products. As the dimension of the sets of matrices is still reasonably small in this case, the time required for this certification is rather low, which allows a performance of about 95% or more. Note that in the same amount of time, a naive brute-force approach would have given lower bounds of comparable quality, but without guarantees.

Sets containing eight 8×8 random matrices

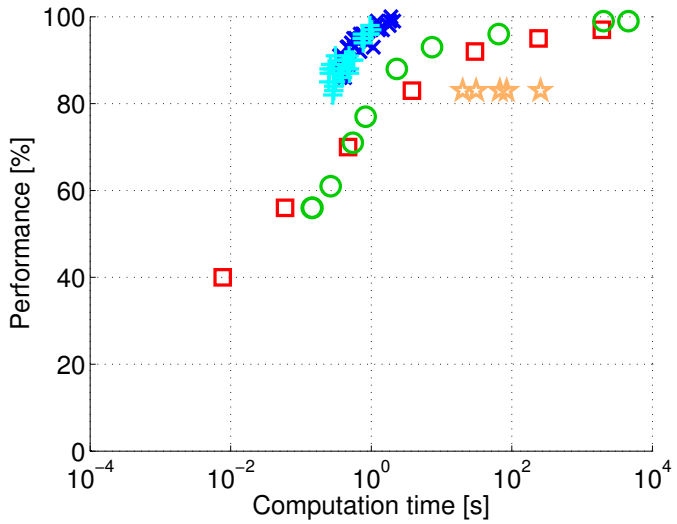
In Figure 5.7, a partial comparison between four algorithms is shown. The other algorithms are not represented as the size of the problem is

now too large to be able to obtain relevant results in reasonable time. Moreover, in some cases, operations such as the computation of a convex hull may become problematic due to the high dimension of the problem. In these cases, allowing more computation time does not always improve the situation. The performance is measured with respect to the best known bound because the exact value of the joint spectral radius is not available in general. Thus, a performance of 100% does *not* mean that the exact value is reached in all cases, but only that the other algorithms have not found better bounds.

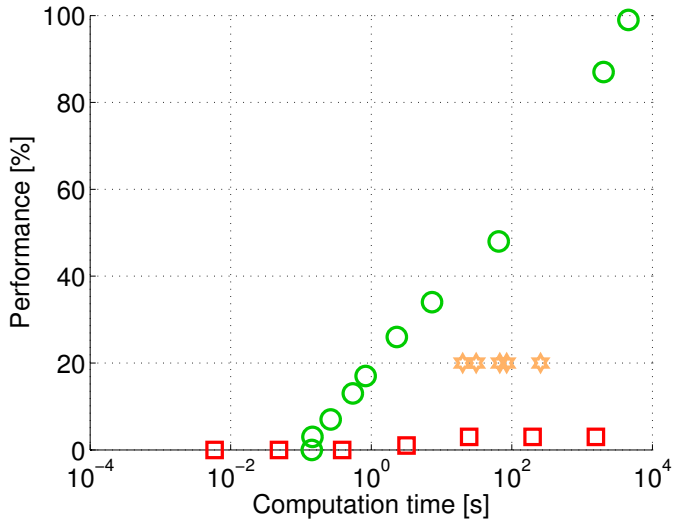
Due to the large number of possible products for a set of eight matrices, Gripenberg's algorithm does not run into overflow problems and is clearly ahead of the other two methods. In this case, the JSR Toolbox did not find very good bounds. This is due to the fact that the first step is only a limited version of a variant of Gripenberg's algorithm, i.e., only short products are tested. The main part of the algorithm is the conitope step, that tries to certify the optimality of the candidate found in the first step if it is possible, while the candidate is replaced if the algorithm finds a better one. This main step is thus time-consuming, and the maximum time limit of 5 minutes is generally too low for sets of eight 8×8 matrices.

Indeed, the typical behavior of the performance of this algorithm corresponds to a lower threshold where only "particularly easy" instances are completely solved, a transition region where the performance rating improves rapidly, and an upper threshold associated to "particularly hard" instances. The three situations shown in Figures 5.3, 5.6 and 5.7 correspond to points in the upper threshold, the transition region, and the lower threshold, respectively. Usually, the algorithm of the JSR Toolbox gives much more accurate results than Gripenberg's method starting from this transition region, especially when we consider the upper bounds. This is even more apparent in Figures 5.4 and 5.5.

Results obtained by the genetic algorithm are still of better quality and/or obtained faster than with other algorithms. Indeed, only Gripenberg's algorithm was able to find better products but a much larger amount of computation time was needed. This supports the fact that our approach may give bounds of good quality in a short amount of time, even for larger sizes of problems. Although the products are not necessarily optimal, they can still be useful as initial points for other algorithms, such as the conitope algorithm used by the JSR Toolbox.



(a) Lower bounds, tolerance of 10^{-2}



(b) Upper bounds, tolerance of 10^{-2}

Figure 5.7: *Results for large sets of eight* 8×8 *random matrices. The exact value of the joint spectral radius is not known for all cases. The performance is therefore measured with respect to the best lower or upper bound available.*

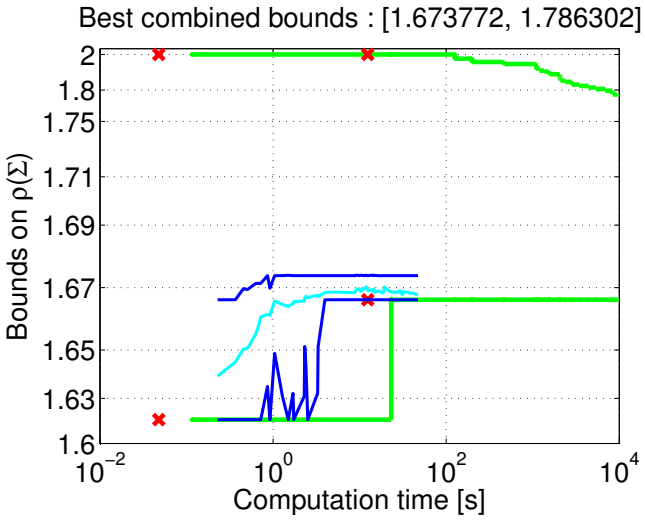


Figure 5.8: *Evolution of the bounds on the joint spectral radius* $\rho(\Sigma)$ *for the set associated to the difference pattern* $+0-00$. *The values of the parameters used for the genetic algorithm are* $(T_1, T_2) = (15, 100)$, *with a population size growing from* $S = 10$ *to* $S = 3250$.

Capacity of codes avoiding difference patterns

The joint spectral radius appears in several applications. One of them is the computation of the capacity of codes avoiding difference patterns. Let us consider the binary alphabet $\mathcal{B} = \{0, 1\}$. Let $x, y \in \mathcal{B}$, then the difference alphabet $\mathcal{D}$ is defined to be the possible values of $x - y$. Hence, $\mathcal{D} = \{-1, 0, +1\}$, which is usually written as $\mathcal{D} = \{-, 0, +\}$, omitting the character "1".

A *binary code* of length n is defined to be a subset of $\mathcal{B}^n$, i.e., a set of words of length n containing zeroes and ones. Let p be a difference pattern of length k, i.e., an element of $\mathcal{D}^k$. A code $\mathcal{C}$ is said to *avoid* the difference pattern p if for all $x, y \in \mathcal{C}$, $x - y$ does *not* contain p as a substring. For example, the code $\{11010, 11100, 01001\}$ does not avoid the difference pattern $+00+$ since $11010 - 01001 = +00+-$, which contains $+00+$ as a substring.

Given a difference pattern p and a length n, we are interested in the maximal size $\delta_n(p)$ of a binary code of length n avoiding p. The *capacity*

$\mathrm{cap}(p)$ is then defined as follows:

$$\mathrm{cap}(p) = \log_2 \lim_{n\to\infty} \delta_n(p)^{\frac{1}{n}}.$$

In [MOS01], the authors show that the value $2^{\mathrm{cap}(p)}$ can be computed as the joint spectral radius of a set Σ of binary matrices. In particular, this set contains 2^{2^z} matrices of size $2^{k-1} \times 2^{k-1}$, where z is the number of zeroes in p. In fact, more complex sets Σ can be obtained by considering codes that avoid a given set of difference patterns instead of a single one.

Figure 5.8 shows the behavior of three algorithms on a set Σ of 256 matrices with size 16×16 and with binary entries, associated with the difference pattern $p = +0-00$.

Due to the large size of the problem, only three algorithms have been tested. The green lines correspond to the upper and lower bounds obtained by Gripenberg's branch-and-bound algorithm with respect to the computation time. The red crosses correspond to the bounds obtained by evaluating all products of length one and two. Going to depth 3 is already too expensive: there are more than 16 million products of length 3. The pruning algorithm could also be used with this set of matrices as all matrices have nonnegative entries, but it does not give better results.

The third algorithm is our genetic algorithm, which only computes lower bounds. The two external lines in blue corresponds to the best and worst results obtained when running the algorithm 50 times for one set of parameters, whereas the average result is represented in cyan. The evolution is not monotone because of the randomness inherent to our method: increasing the population size tends to give better results but this is not guaranteed for all runs. It is interesting to remark that the default algorithm of the JSR Toolbox is unable to handle this set of matrices; one has to modify the different parameters in order to obtain bounds in a reasonable amount of time.

By comparing the bounds obtained by Gripenberg's algorithm and the genetic algorithm, one can see that the latter has indeed been able to obtain good lower bounds in a short amount of time, despite the size of the problem. Indeed, the computation time of the genetic algorithm is much lower and the results are never worse than the results obtained with the branch-and-bound algorithm. Moreover, the genetic algorithm has been able to find a better bound in most of the cases.

As the best lower bound seems to stabilize at about 1.6738 with a product of length 6, one could conjecture that this is the exact value of

the joint spectral radius. Unfortunately, none of the algorithms was able to prove this in a reasonable amount of time.

Tables 5.5 and 5.6 show the results obtained by both Gripenberg's algorithm and our method, on several sets related to the capacity of codes. We first present results for all difference patterns of length at most 4, then several problems with larger sizes.

Note that several rows have been merged in these tables. For example, it can be proved that $\mathrm{cap}(0+) = \mathrm{cap}(0-) = \mathrm{cap}(+0) = \mathrm{cap}(-0)$. Hence, we only present the results for the pattern $0+$ in the tables. This is due to the following properties:

- Reversing the string does not change the capacity:

$$\mathrm{cap}(p_k p_{k-1} \dots p_1) = \mathrm{cap}(p_1 p_2 \dots p_k)$$

  for all difference patterns $p_1 p_2 \dots p_k \in \mathcal{D}^k$. For instance, we have $\mathrm{cap}(0++0-) = \mathrm{cap}(-0++0)$.

- Taking the complement of the string does not change the capacity:

$$\mathrm{cap}(\overline{p_1 p_2} \dots \overline{p_k}) = \mathrm{cap}(p_1 p_2 \dots p_k),$$

  where $\overline{+} = -$, $\overline{0} = 0$ and $\overline{-} = +$. For instance, we have $\mathrm{cap}(0+ +0-) = \mathrm{cap}(0--0+)$.

- Replacing the symbols at odd positions by their complements does not change the capacity. For instance, we have $\mathrm{cap}(+++++) = \mathrm{cap}(-+-+-)$.

- Replacing the symbols at even positions by their complements does not change the capacity. For instance, we have $\mathrm{cap}(+++++) = \mathrm{cap}(+-+-+)$.

These results confirm the behavior observed in Figure 5.8. Our genetic algorithm was able to find a lower bound at least as good as the result obtained by Gripenberg's algorithm and is still able to produce results with large size problems.

Difference pattern	$\|\Sigma\|$	Size of the matrices	Genetic algorithm	Gripenberg's algorithm
$0+$	4	2×2	$1.000 \leqslant \rho$	$1.000 \leqslant \rho \leqslant 1.040$
$++$	2	2×2	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.618$
$00+$	16	4×4	$1.000 \leqslant \rho$	$1.000 \leqslant \rho \leqslant 1.190$
$0+0$	16	4×4	$1.414 \leqslant \rho$	$1.414 \leqslant \rho \leqslant 1.414$
$0++$	4	4×4	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.622$
$+0+$	4	4×4	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.618$
$+0-$	4	4×4	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.618$
$+++$	2	4×4	$1.839 \leqslant \rho$	$1.839 \leqslant \rho \leqslant 1.839$
$++-$	2	4×4	$1.755 \leqslant \rho$	$1.755 \leqslant \rho \leqslant 1.755$
$000+$	256	8×8	$1.000 \leqslant \rho$	$1.000 \leqslant \rho \leqslant 1.414$
$00+0$	256	8×8	$1.414 \leqslant \rho$	$1.356 \leqslant \rho \leqslant 1.606$
$00++$	16	8×8	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.668$
$0+0+$	16	8×8	$1.658 \leqslant \rho$	$1.658 \leqslant \rho \leqslant 1.675$
$0+0-$	16	8×8	$1.658 \leqslant \rho$	$1.658 \leqslant \rho \leqslant 1.675$
$0++0$	16	8×8	$1.735 \leqslant \rho$	$1.735 \leqslant \rho \leqslant 1.736$
$+00+$	16	8×8	$1.618 \leqslant \rho$	$1.618 \leqslant \rho \leqslant 1.618$
$0+++$	4	8×8	$1.839 \leqslant \rho$	$1.839 \leqslant \rho \leqslant 1.840$
$0++-$	4	8×8	$1.755 \leqslant \rho$	$1.755 \leqslant \rho \leqslant 1.767$
$+0++$	4	8×8	$1.839 \leqslant \rho$	$1.839 \leqslant \rho \leqslant 1.839$
$+0-+$	4	8×8	$1.797 \leqslant \rho$	$1.797 \leqslant \rho \leqslant 1.800$
$++++$	2	8×8	$1.928 \leqslant \rho$	$1.928 \leqslant \rho \leqslant 1.928$
$+++-$	2	8×8	$1.867 \leqslant \rho$	$1.867 \leqslant \rho \leqslant 1.871$
$++--$	2	8×8	$1.891 \leqslant \rho$	$1.891 \leqslant \rho \leqslant 1.891$

Table 5.5: *Results obtained with sets of matrices associated to difference patterns of lengths* 2, 3 *and* 4. *All non-equivalent patterns are represented in the table. The computation time has been limited to about 30 seconds. The parameters for the genetic algorithm are* $(T_1, T_2) = (15, 100)$, *with a maximal population size of* $S = 3000$.

Difference pattern	$\lvert\Sigma\rvert$	Size of the matrices	Genetic algorithm	Gripenberg's algorithm
0000+	65536	16×16	$1.000 \leqslant \rho$	Timeout
000+0	65536	16×16	$1.414 \leqslant \rho$	Timeout
00+00	65536	16×16	$1.498 \leqslant \rho$	Timeout
000++	256	16×16	$1.618 \leqslant \rho$	$1.611 \leqslant \rho \leqslant 1.871$
00++0	256	16×16	$1.741 \leqslant \rho$	$1.722 \leqslant \rho \leqslant 2.000$
00+0+	256	16×16	$1.674 \leqslant \rho$	$1.666 \leqslant \rho \leqslant 2.000$
00+0−	256	16×16	$1.674 \leqslant \rho$	$1.666 \leqslant \rho \leqslant 2.000$
0+0+0	256	16×16	$1.717 \leqslant \rho$	$1.705 \leqslant \rho \leqslant 2.000$
0+0−0	256	16×16	$1.709 \leqslant \rho$	$1.705 \leqslant \rho \leqslant 2.000$
0000++	65536	32×32	$1.618 \leqslant \rho$	Timeout
000++0	65536	32×32	$1.759 \leqslant \rho$	Timeout
000+0+	65536	32×32	$1.674 \leqslant \rho$	Timeout
000+0−	65536	32×32	$1.666 \leqslant \rho$	Timeout
00++00	65536	32×32	$1.762 \leqslant \rho$	Timeout
00+00+	65536	32×32	$1.685 \leqslant \rho$	Timeout
00+0+0	65536	32×32	$1.717 \leqslant \rho$	Timeout
00+0−0	65536	32×32	$1.705 \leqslant \rho$	Timeout
0+00+0	65536	32×32	$1.705 \leqslant \rho$	Timeout

Table 5.6: *Results obtained with sets of matrices associated to several difference patterns of lengths* 5 *and* 6 *The computation time has been limited to about 60 seconds (length* 5*) or 300 seconds (length* 6*). The parameters for the genetic algorithm are* $(T_1, T_2) = (15, 100)$*, with a maximal population size of* $S = 3000$*. In several cases ("timeout"), Gripenberg's algorithm was unable to return any bounds, even after a much longer period of time, e.g., more than 20 minutes for the pattern* 0000+*.*

5.7 Conclusion

The approximation of the joint spectral radius is a difficult computational problem. Different methods had previously been proposed to approximate the joint spectral radius, but most of them have only been studied from a theoretical point of view. In practice, most of them require a large amount of computation time to obtain an accurate approximation.

In this chapter, a large set of approximation methods for the joint spectral radius has been presented. These algorithms have been implemented in MATLAB® and released as part of the *JSR Toolbox*, a project available to the public via MATLAB® Central. These implementations can be used to compare the performance of the different algorithms on a set of benchmarks. A default algorithm combining several approaches can also be found in the toolbox, which may be useful if one wants to approximate the joint spectral radius of a set without having to decide between the different available algorithms.

We also propose a new heuristic method that computes lower bounds on the joint spectral radius at a low computational cost. This is done using a genetic algorithm that tries to investigate promising products of matrices. Although there is no a priori guarantee on the quality of the bounds, numerical experiments done on a large number of sets of matrices show that this approach may give good bounds in practice. Furthermore, it is able to handle problems of large size, whereas other methods simply fail or are unable to terminate in a reasonable amount of time. The products associated to the lower bounds may be used as an initial point in several other algorithms, e.g., methods that check if a given product is optimal and return upper and lower bounds on the joint spectral radius.

The results in Section 5.6 show that although all the usual methods are able to derive bounds that theoretically converge to the exact value of the joint spectral radius, this is often not observed in practice due to limitations such as numerical problems, sometimes even if a large computation time is allowed. Using a combination of several methods seems to be a good approach. For example, one can use a branch-and-bound technique or our genetic algorithm to quickly find a good lower bound, and then use the corresponding product as starting point of a slower geometric algorithm. Indeed, the numerical experiments also confirm that the exact value of the joint spectral radius is often reached by a finite product, and that most of the time, such a finite product can be found

in a reasonable amount of time. Of course, the main difficulty is to certify that this product is indeed optimal, but this shows the difference in difficulty between deriving good lower and upper bounds.

The technique presented in this article can also become the basis for further work. A better integration of the optional post-processing step can be done, e.g., if the second algorithm detects that the candidate product is not optimal, it may be wise to insert the new candidate in the population and restart the genetic algorithm. One can also consider investigating other spectral quantities, such as the joint spectral subradius which represents the minimum (instead of the maximum) asymptotic growth rate associated to the set of matrices.

CHAPTER 6
Conclusion

Optimization is a key aspect in many domains. Indeed, solving a problem typically involves finding a solution that is the *best* one with respect to some criteria. However, despite all the progress in algorithms, computing technology and hardware, finding the optimal solution may be computationally intractable. Indeed, some problems may be time critical while we are expected to handle larger and larger problem sizes. Other problems may be inherently difficult in the sense of complexity theory. In practical applications, one may be satisfied with suboptimal solutions, provided that they are "good enough". For example, it may happen that the considered problem possesses a large number of very good solutions that are close to optimality. Using such nearly optimal solutions may be sufficient in some cases.

Heuristic strategies consist in algorithmic approaches that aim at finding a good solution, even if it is not optimal. As the framework is very general, it can be — and has successfully been — applied to many difficult problems. Of course, for the same reason, heuristics can be adapted in a suitable manner to the particular problem we are interested in if we want to improve the performance. These methods present two main features: on the one hand, algorithms such as simulated annealing, tabu search or simple greedy approaches are designed to improve a single candidate solution. On the other hand, population-based methods such as genetic algorithms or particle swarm optimization algorithms use a large set of individuals that exchange information in order to produce new candidate solutions.

Different heuristics may also be combined or hybridized in order to improve their performance. For example, a population-based algorithm can be used for the exploration of the search space while a local improvement method is applied on the population elements in order to find locally optimal solutions in the neighborhood of the current candidates.

This is the strategy we have taken in this thesis. Indeed, we investigate the performance of combinations of heuristics for three specific applications belonging to different domains.

In Chapter 3, we have studied the problem of the *minimum-volume arbitrarily oriented bounding box* in the three-dimensional real space where one wants to find the smallest rectangular parallelepiped enclosing a set of points. Even though the exact optimal solution can be obtained in $\mathcal{O}(n^3)$ time for a set of n points, the corresponding algorithm is much too expensive as computation time is critical in many relevant applications, mostly in the field of computer graphics. Hence, in practice, different heuristics are used to solve the problem. Although these approaches may be very fast, the solutions are sometimes very far from the optimal one. We have designed a hybrid algorithm combining the genetic and the Nelder-Mead algorithms on the manifold $SO(3, \mathbb{R})$, which has been able to find very good solutions at a low computational cost. The different parameters can be used to adjust a trade-off between accuracy and speed. In our experiments in particular, optimal bounding boxes have been obtained much faster than with the $\mathcal{O}(n^3)$ exact algorithm.

Chapter 4 was devoted to the *column subset selection* problem. Here, we are interested in selecting k columns from a given $m \times n$ matrix A such that they are the "most independent" possible. This problem appears in fields such as data mining as it can be interpreted as an approximation of the information encoded in the matrix A, with a much smaller matrix. Dimensionality reduction techniques are indeed very important as they allow the analysis of large databases by decreasing the complexity of the problem. Our specific version of the column subset selection problem minimizes the volume of the subset and is known to be NP-hard. Here, we propose a heuristic approach consisting of a windowed algorithm at the core, with possible combinations with simulated annealing, stochastic hill climbing or tabu search variants. These heuristic approaches are designed with the intent of being faster than the non-windowed algorithm, in particular if $k \ll m, n$. In our numerical experiments, these algorithms have been able to find a subset of similar or better quality than the non-windowed algorithm while being significantly faster.

Finally, Chapter 5 described our research about the *joint spectral radius*. This quantity is a generalization of the usual spectral radius to sets of matrices, and appears in many applications in system and control theory and also in fields related to discrete mathematics. The problem of approximating the joint spectral radius is also known to be NP-hard.

Many different approaches have been proposed for its computation, but at the same time, many algorithms have only been studied from a theoretical point of view. Hence, our goal here was twofold. Firstly, we have implemented the different algorithms in order to compare their practical performances, both in accuracy and in computational cost. These algorithms are now available as part of a MATLAB® toolbox. Secondly, we have proposed a heuristic approach to derive lower bounds on the joint spectral radius at a low computational cost. This algorithm is based on a genetic algorithm combined with several local improvement rules. Our experiments show that this heuristic strategy has been able to find very good solutions that are often optimal for small-size problems, and that is able to manage problems with larger dimensions, where other algorithms are simply too expensive.

Perspectives and future research

We have proposed heuristic approaches tackling three problems in very different fields. The common goal was to develop algorithms which quickly yield good solutions with respect to some fitness function. As a by-product, they also have more tolerance for increasing problem sizes. Another common characteristic is the presence of tunable parameters that affect the accuracy of the result or the computation time.

Although it may be possible to infer the effect of a small modification of the parameters (for example, increasing the population size in a genetic algorithm will probably produce more accurate results while possibly increasing the computation time), one problem remains: what "initial value" should we use *a priori*, without performing a full experimental analysis? For instance, if one wants to approximate the joint spectral *sub*radius using the same approach as in Chapter 5, can one keep the same range of parameters?

In the last decades, research has been done on developing methods that try to tune the parameters of an optimization method in an automatic fashion. This strategy is sometimes known as *meta-optimization* or *hyper-heuristic* approaches. Investigating this area of research should allow us to better understand the behavior of the heuristics applied to our problems. Furthermore, this would also facilitate the usage of algorithms for non-experts of the field. Such an algorithm would then be considered as a black box since no user input would be required, except for the problem data itself.

Independently, there is also room for improvement for experts of the problems. Indeed, even though the algorithms we have presented in this thesis are heuristics adapted to the problems we consider, they do not use all the available information. For example, the optimal bounding box of a set of points cannot have an arbitrary orientation: possible candidates are related to the edges of the convex hull. Taking into account such information about the structure of the problem may lead to better performance, if one finds a way to exploit it wisely. It is thus clear that there are still many remaining things to investigate, whether one moves up to general meta-optimization approaches, or concentrates on the exploitation of particular features of the problems.

If we consider particular problems, then there is still another important point: as we have seen in Chapter 5, "In theory there is no difference between theory and practice. In practice there is." The practical behavior of an algorithm may be very implementation dependent. Without going into low-level implementation details, the performance of numerical algorithms depends on choices such as the data structures used to store some relevant information, or the methods used to approximate complex objects. Such design choices have to be investigated thoroughly if we want to produce optimized implementations of our algorithms for practical applications.

Finally, there is also the fact that our heuristic algorithms usually involve randomness. This is a key component in the exploration of the search space and one may want to analyze how this randomness affects the behavior of the algorithm. For example, it may be possible to design deterministic algorithms with similar performances to our randomized algorithms. This may be useful in several applications where the process has to be deterministic and would also facilitate any theoretical analysis of the algorithms. Such analysis would also allow us to even better understand the behavior of our algorithms.

Bibliography

[AHPV04] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

[AJPR11] Amir Ali Ahmadi, Raphaël M. Jungers, Pablo A. Parrilo, and Mardavij Roozbehani. Analysis of the joint spectral radius via lyapunov functions on path-complete graphs. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 13–22. ACM, 2011.

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of graphics, GPU, and game tools*, 5(1):9–22, 2000.

[AS98] Tsuyoshi Ando and Mau-Hsiang Shih. Simultaneous contractibility. *SIAM Journal on Matrix Analysis and Applications*, 19(2):487–498, 1998.

[BA10] Pierre B. Borckmans and Pierre-Antoine Absil. Oriented bounding box computation using particle swarm optimization. In *Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN2010)*, pages 345–350, Bruges, Belgium, 2010. D-side publications.

[Bar88] Nikita Barabanov. Lyapunov indicators of discrete inclusions i-iii. *Automation and Remote Control*, 49:152–157, 283–287, 558–565, 1988.

[BBP10] Mary E. Broadbent, Martin Brown, and Kevin Penner. Subset selection algorithms: Randomized vs. deterministic. *SIAM Undergraduate Research Online*, 3, 2010.

[BCG+96] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S.B. Mitchell, and Ayellet Tal. Boxtree: A hierarchical representation for surfaces in 3d. In *Computer Graphics*

Forum, volume 15, pages 387–396. Wiley Online Library, 1996.

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[BGD12] Christopher G. Baker, Kyle A. Gallivan, and Paul Van Dooren. Low-rank incremental methods for computing dominant singular subspaces. *Linear Algebra and its Applications*, 436(8):2866–2888, 2012.

[BHP01] Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91–109, 2001.

[BJP06] Vincent D. Blondel, Raphaël M. Jungers, and Vladimir Y. Protasov. On the complexity of computing the capacity of codes that avoid forbidden difference patterns. *IEEE Transactions on Information Theory*, 52(11):5122–5127, 2006.

[BM02] Thierry Bousch and Jean Mairesse. Asymptotic height optimization for topical ifs, tetris heaps, and the finiteness conjecture. *Journal of the AMS*, 15(1):77–111, 2002.

[BMD09] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

[BN05] Vincent D. Blondel and Yurii Nesterov. Computationally efficient approximations of the joint spectral radius. *SIAM Journal on Matrix Analysis and Applications*, 27(1):256–272, 2005.

[BNT05] Vincent D. Blondel, Yurii Nesterov, and Jacques Theys. On the accuracy of the ellipsoid norm approximation of the joint spectral radius. *Linear Algebra and its Applications*, 394(1):91–107, 2005.

[Bro70] Charles G. Broyden. The convergence of a class of double-rank minimization algorithms. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[BS09] Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, 2009.

[BT00] Vincent D. Blondel and John N. Tsitsiklis. The boundedness of all products of a pair of matrices is undecidable. *Systems & Control Letters*, 41(2):135–140, 2000.

[BTV03] Vincent D. Blondel, Jacques Theys, and Alexander A. Vladimirov. An elementary counterexample to the finiteness conjecture. *SIAM Journal on Matrix Analysis and Applications*, 24(4):963–970, 2003.

[BW92] Marc A. Berger and Yang Wang. Bounded semi-groups of matrices. *Linear Algebra and its Applications*, 166:21–27, 1992.

[CB00] Christophe Couvreur and Yoram Bresler. On the optimality of the backward greedy algorithm for the subset selection problem. *SIAM Journal on Matrix Analysis and Applications*, 21(3):797–808, 2000.

[CB10] Chia-Tche Chang and Vincent D. Blondel. A comparison of approximation algorithms for the joint spectral radius. In *Book of Abstracts of the 29th Benelux Meeting on Systems and Control (BMSC10)*, page 85, Heeze, The Netherlands, Mar. 2010.

[CB11a] Chia-Tche Chang and Vincent D. Blondel. Approximating the joint spectral radius using a genetic algorithm framework. In *Proceedings of the 18th IFAC World Congress (IFAC WC2011)*, pages 8681–8686, Milano, Italy, Aug. 2011.

[CB11b] Chia-Tche Chang and Vincent D. Blondel. A genetic algorithm approach for the approximation of the joint spectral radius. In *Book of Abstracts of the 30th Benelux Meeting on Systems and Control (BMSC11)*, page 105, Lommel, Belgium, Mar. 2011.

[CB12a] Chia-Tche Chang and Vincent D. Blondel. An experimental study of approximation algorithms for the joint spectral radius. *Numerical Algorithms*, 2012. Accepted for publication.

[CB12b] Chia-Tche Chang and Vincent D. Blondel. A genetic based algorithm for fast approximations of the joint spectral radius. Submitted to Systems & Control Letters, 2012.

[CGM11] Chia-Tche Chang, Bastien Gorissen, and Samuel Melchior. Fast oriented bounding box optimization on the rotation group $SO(3, \mathbb{R})$. *ACM Transactions on Graphics*, 30(5):122:1–122:16, Oct. 2011.

[CGSCZ10] Antonio Cicone, Nicola Guglielmi, Stefano Serra-Capizzano, and Marino Zennaro. Finiteness property of pairs of 2×2 sign-matrices via real extremal polytope norms. *Linear Algebra and its Applications*, 432(2-3):796–816, 2010.

[CGT00] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Cha06] Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry: Theory and Applications*, 35(1-2):20–35, 2006.

[CI94] Shivkumar Chandrasekaran and Ilse C.F. Ipsen. On rank-revealing factorisations. *SIAM Journal on Matrix Analysis and Applications*, 15:592, 1994.

[CIBD12] Chia-Tche Chang, Ilse Ipsen, Vincent D. Blondel, and Paul Van Dooren. Polynomial-time subset selection via updating. In preparation, 2012.

[CJB09] Chia-Tche Chang, Raphaël M. Jungers, and Vincent D. Blondel. On the growth rate of matrices with row uncertainties. In *Book of Abstracts of the 14th Belgian-French-German Conference on Optimization (BFG09)*, page 86, Leuven, Belgium, Sept. 2009.

[cMI09] Ali Çivril and Malik Magdon-Ismail. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science*, 410(47):4801–4811, 2009.

[CS03] Rachid Chelouah and Patrick Siarry. Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research*, 148(2):335–348, 2003.

[CSV09] Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[DA99] Nicolas Durand and Jean-Marc Alliot. A combined nelder-mead simplex and genetic algorithm. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO99)*, pages 1–7, Orlando, FL, USA, 1999. Morgan Kaufmann.

[Dav91] Lawrence D. Davis. *Handbook Of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[dB98] Gino Van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1998.

[DB01] Jamal Daafouz and Jacques Bernussou. Parameter dependent lyapunov functions for discrete time systems with time varying parametric uncertainties. *Systems & Control Letters*, 43(5):355–359, 2001.

[DHKK09] Darko Dimitrov, Mathias Holst, Christian Knauer, and Klaus Kriegel. Closed-form solutions for continuous pca and bounding box algorithms. *Computer Vision and Computer Graphics. Theory and Applications*, 24:26–40, 2009.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DKKR09] Darko Dimitrov, Christian Knauer, Klaus Kriegel, and Günter Rote. Bounds on the quality of the pca bounding boxes. *Computational Geometry: Theory and Applications*,

42(8):772–789, 2009. Special Issue on the 23rd European Workshop on Computational Geometry.

[DL92] Ingrid Daubechies and Jeffrey C. Lagarias. Two-scale difference equations ii. local regularity, infinite products of matrices and fractals. *SIAM Journal on Mathematical Analysis*, 23(4):1031–1079, 1992.

[Dor92] Marco Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, Milano, Italy, 1992.

[Dós07] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leqslant 11/9\ OPT(I) + 6/9$. *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11, 2007.

[DP85] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[Dre06] David W. Dreisigmeyer. Direct search algorithms over riemannian manifolds. Technical report, Los Alamos National Laboratory, USA, 2006.

[DRVW06] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. *Theory of Computing*, 2:225–247, 2006.

[DV06] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 4110:292–303, 2006.

[EM85] Herbert Edelsbrunner and Hermann A. Maurer. Finding extreme points in three dimensions and solving the post-office problem in the plane. *Information processing letters*, 21(1):39–47, 1985.

[Eri04] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. Morgan Kaufmann, Dec. 2004.

[Fek23] Michael Fekete. über die verteilung der wurzeln bei gewissen algebraischen gleichungen mit ganzzahligen koeffizienten. *Mathematische Zeitschrift*, 17(1):228–249, 1923.

[Fle70] Roger Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[FM93] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, 1993.

[FR64] Roger Fletcher and Colin M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[FS75] Herbert Freeman and Ruth Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413, 1975.

[GASF94] Alejandro Garcia-Alonso, Nicolás Serrano, and Juan Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 14(3):36–43, 1994.

[GE96] Ming Gu and Stanley C. Eisenstat. Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[GJ79] Michael R. Gary and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.

[GJF09] Christophe Geuzaine and cois Remacle Jean-Fran˙Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[GKS76] Gene H. Golub, Virginia Klema, and Gilbert W. Stewart. Rank degeneracy and least squares problems. Technical Report TR-456, Dept. of Computer Science, University of Maryland, College Park, MD, 1976.

[GL98] Fred W. Glover and Manuel Laguna. *Tabu search*, volume 1. Springer, 1998.

[GLM96] Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH96)*, pages 171–180, New York, NY, USA, 1996. ACM.

[Glo89] Fred W. Glover. Tabu search — part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glo90] Fred W. Glover. Tabu search — part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[Gol70] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[GP11] Nicola Guglielmi and Vladimir Y. Protasov. Exact computation of joint spectral characteristics of linear operators. *Foundations of Computational Mathematics*, pages 1–61, 2011.

[Gri96] Gustaf Gripenberg. Computing the joint spectral radius. *Linear Algebra and its Applications*, 234:43–60, 1996.

[Gro08] GAMMA Group. 3D meshes research database of the group Génération Automatique de Maillages et Méthodes d'Adaptation (GAMMA), INRIA, France, 2008. Website. Available at `http://www-roc.inria.fr/gamma/gamma/download/download.php`.

[Gur95] Leonid Gurvits. Stability of discrete linear inclusion. *Linear Algebra and its Applications*, 231:47–85, 1995.

[GvL96] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[GZ07] Nicola Guglielmi and Marino Zennaro. Balanced complex polytopes and related vector and matrix norms. *Journal of Convex Analysis*, 14:729–766, 2007.

[GZ09] Nicola Guglielmi and Marino Zennaro. Finding extremal complex polytope norms for families of real matrices. *SIAM Journal on Matrix Analysis and Applications*, 31(2):602–620, 2009.

[HMST11] Kevin G. Hare, Ian D. Morris, Nikita Sidorov, and Jacques Theys. An explicit counterexample to the lagarias-wang finiteness conjecture. *Advances in Mathematics*, 226(6):4667–4701, 2011.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. Second edition, 1992.

[IZK98] Andrei Iones, Sergey Zhukov, and Anton Krupkin. On optimality of obbs for visibility tests for frustum culling, ray shooting and collision detection. In *Proceedings of the Computer Graphics International Conference (CGI98)*, pages 256–263, Los Alamitos, CA, USA, 1998. IEEE Computer Society.

[JB08] Raphaël M. Jungers and Vincent D. Blondel. On the finiteness property for rational matrices. *Linear Algebra and its Applications*, 428(10):2283–2295, 2008.

[JFGCM10] cois Remacle Jean-Fran Christophe Geuzaine, Gaëtan Compère, and Emilie Marchandise. High-quality surface remeshing using harmonic maps. *International Journal for Numerical Methods in Engineering*, 83(4):403–425, 2010.

[Jol02] Ian T. Jolliffe. *Principal Component Analysis*. Springer, New York, NY, USA, 2002.

[JPB09] Raphaël M. Jungers, Vladimir Y. Protasov, and Vincent D. Blondel. Overlap-free words and spectra of matrices. *Theoretical Computer Science*, 410(38):3670–3684, 2009.

[JTT00] Pablo Jiménez, Federico Thomas, and Carme Torras. 3d collision detection: A survey. *Computers and Graphics*, 25:269–285, 2000.

[Jun09] Raphaël M. Jungers. *The Joint Spectral Radius: Theory and Applications*. Springer, Berlin, Germany, 2009.

[Kah66] William Kahan. Numerical linear algebra. *Canadian Mathematical Bulletin*, 9:757–801, 1966.

[KB07] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[KE95] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[Kel99] Carl T. Kelley. *Iterative Methods for Optimization*. Number 18 in Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[KGV83] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KHM07] Krzysztof A. Krakowski, Knut Hüper, and Jonathan H. Manton. On the computation of the karcher mean on spheres and special orthogonal groups. In *RoboMat 2007, Workshop on Robotics and Mathematics*, pages 119–124, Coimbra, Portugal, Sep. 2007.

[KLT03] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1997.

[Kor08] Johannes Korsawe. Minimal bounding box. MATLAB® Central File Exchange, January 2008. Software, MATLAB® function. Available at `http://www.mathworks.com/matlabcentral/fileexchange/18264`.

[Koz90] Victor S. Kozyakin. Algebraic unsolvability of problem of absolute stability of desynchronized systems. *Automation and Remote Control*, 51(6):754–759, 1990.

[Koz05] Victor S. Kozyakin. A dynamical systems construction of a counterexample to the finiteness conjecture. In *Proceedings of the 44th IEEE Conference on Decision and Control (CDC05)*, pages 2338–2343, Seville, Spain, 2005. IEEE.

[Koz07] Victor S. Kozyakin. Structure of extremal trajectories of discrete linear systems and the finiteness conjecture. *Automation and Remote Control*, 68(1):174–209, 2007.

[Koz09] Victor S. Kozyakin. On accuracy of approximation of the spectral radius by the gelfand formula. *Linear Algebra and its Applications*, 431(11):2134–2141, 2009.

[Koz10] Victor S. Kozyakin. Iterative building of barabanov norms and computation of the joint spectral radius for matrix sets. *Discrete and Continuous Dynamical Systems - Series B*, 14(1):143–158, 2010.

[Koz11] Victor S. Kozyakin. A relaxation scheme for computation of the joint spectral radius of matrix sets. *Journal of Difference Equations and Appllications*, 17(2):185–201, 2011.

[Lib03] Daniel Liberzon. *Switching in systems and control*. Birkhäuser, 2003.

[LKM+00] Michael Lahanas, Thorsten Kemmerer, Natasa Milickovic, Kostas Karouzakis, Dimos Baltas, and Nikolaos Zamboglou. Optimized bounding boxes for three-dimensional treatment planning in brachytherapy. *Medical Physics*, 27(10):2333–2342, 2000.

[LW95] Jeffrey C. Lagarias and Yang Wang. The finiteness conjecture for the generalized spectral radius of a set of matrices. *Linear Algebra and its Applications*, 214:17–42, 1995.

[Mae96] Mohsen Maesumi. An efficient lower bound for the generalized spectral radius of a set of matrices. *Linear Algebra and its Applications*, 240:1–7, 1996.

[Mae98] Mohsen Maesumi. Calculating joint spectral radius of matrices and hölder exponent of wavelets. *Approximation theory IX*, 2:1–8, 1998.

[McK99] Ken I.M. McKinnon. Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1999.

[Moa02] Maher Moakher. Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications*, 24(1):1–16, 2002.

[MOS01] Bruce E. Moision, Alon Orlitsky, and Paul H. Siegel. On codes that avoid specified differences. *IEEE Transactions on Information Theory*, 47(1):433–442, 2001.

[MOS07] Bruce E. Moision, Alon Orlitsky, and Paul H. Siegel. On codes with local joint constraints. *Linear Algebra and its Applications*, 422(2–3):442–454, 2007.

[MS70] Bruce A. Murtagh and Roger W.H. Sargent. Computational experience with quadratically convergent minimisation methods. *The Computer Journal*, 13(2):185–194, 1970.

[NCM12] Ferrante Neri, Carlos Cotta, and Pablo Moscato. *Handbook of memetic algorithms*, volume 379. Springer, 2012.

[NM65] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer-Verlag, 2nd edition, 2006.

[O'R85] Joseph O'Rourke. Finding minimal enclosing boxes. *International Journal of Computer & Information Sciences*, 14:183–199, Jun. 1985.

[PJ08] Pablo A. Parrilo and Ali Jadbabaie. Approximation of the joint spectral radius using sum of squares. *Linear Algebra and its Applications*, 428(10):2385–2402, May 2008.

[PJB10] Vladimir Y. Protasov, Raphaël M. Jungers, and Vincent D. Blondel. Joint spectral characteristics of matrices: a conic programming approach. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2146–2162, 2010.

[PML97] Madhav K. Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64, 1997.

[PR69] Elijah Polak and Gérard Ribière. Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, 3(16):35–43, 1969.

[Pro96] Vladimir Y. Protasov. The joint spectral radius and invariant sets of linear operators. *Fundamentalnaya i prikladnaya matematika*, 2(1):205–231, 1996.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational geometry: an introduction*. Springer, New York, NY, USA, 1985.

[PTVF92] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[Rez00] Bruce Reznick. *Some concrete aspects of Hilbert's 17th Problem*, volume 253 of *Contemporary Mathematics*, pages 251–272. American Mathematical Society, 2000.

[RS60] Gian-Carlo Rota and Gilbert Strang. A note on the joint spectral radius. *Indagatione Mathematicae*, 22:379–381, 1960.

[Sha70] David F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[Sha78] Michael Ian Shamos. *Computational geometry*. PhD thesis, Yale University, New Haven, CT, USA, 1978.

[Ski98] Steven S. Skiena. *The algorithm design manual*. Springer-Verlag, 1998.

[SS09] Alain Sarlette and Rodolphe Sepulchre. Consensus optimization on manifolds. *SIAM Journal on Control and Optimization*, 48(1):56–76, 2009.

[SWM+07] Robert Shorten, Fabian Wirth, Oliver Mason, Kai Wulff, and Christopher King. Stability criteria for switched and hybrid systems. *SIAM Review*, 49(4):545–592, 2007.

[TB97] John N. Tsitsiklis and Vincent D. Blondel. The lyapunov exponent and joint spectral radius of pairs of matrices are hard — when not impossible — to compute and to approximate. *Mathematics of Control, Signals, and Systems*, 10(1):31–40, 1997.

[Tou83] Godfried Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of the IEEE 1983 Mediterranean Electrotechnical Conference (MELECON83)*, volume 9, pages 1–8, Athens, Greece, May 1983.

[VHJ+11] Guilllaume Vankeerbergen, Julien Hendrickx, Raphaël Jungers, Chia-Tche Chang, and Vincent Blondel. The JSR Toolbox. MATLAB® Central File Exchange, 2011. Software, MATLAB® toolbox. Available at `http://www.mathworks.com/matlabcentral/fileexchange/33202-the-jsr-toolbox`.

[vLA87] Peter J.M. van Laarhoven and Emile H.L. Aarts. *Simulated annealing: theory and applications*, volume 37. Springer, 1987.

[Wir02] Fabian Wirth. The generalized spectral radius and extremal norms. *Linear Algebra and its Applications*, 342(1):17–40, 2002.

[Wol69] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

[Yan09] Xin-She Yang. Firefly algorithms for multimodal optimization. *Stochastic algorithms: foundations and applications*, pages 169–178, 2009.

[YD09] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC2009)*, pages 210–214. IEEE, 2009.