UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Some open problems on geometric separability

Project Memory

Nicolau Oliver Burwitz

Bachelor Degree in Informatics Engineering
Major in Computing
Factultat d'Informàtica de Barcelona
Universitat Politècninca de Catalunya

Director: Carlos Seara
Department of Mathematics

23 June 2022

Abstract

Català

En aquest projecte hem considerat dos problemes oberts de separació de punts vermells i blaus en el pla, contextualitzats en el camp de la geometria computacional. Partint de resultats ja coneguts, hem estès i millorat els algoritmes, concretament per la separabilitat emprant 4 rectes paral·leles en tires monocromàtiques. I hem analitzat condicions suficients per a aquests criteris de separabilitat.

Castellano

En este trabajo hemos considerado dos problemas abiertos de separación de puntos rojos y azules en el plano, contextualizados en el campo de la geometría computacional. Partiendo de resultados ya conocidos, hemos extendido y mejorado los algoritmos, concretamente para la separabilidad por 4 rectas paralelas en tiras monocromáticas. I hemos analizado condiciones suficientes para estos criterios de separabilidad.

English

In this project, we have tackled two open questions regarding the separability of red and blue points in the plane, from the framework of computational geometry. Building on existing results, we have extended and improved algorithms, specifically for the separability using 4 parallel lines that define monochromatic strips. Also, sufficient conditions to meet these separability criteria have been studied.

Contents

List of Figures

List of Tables

1 Introduction

This project is a Bachelor Thesis written by Nicolau Oliver, student of the FIB at UPC Barcelona. It belongs to the GEI degree, within the computation branch. This section will describe the context for the project, why it has been developed, and the roles of the different actors involved.

1.1 Context

The project is inscribed in the broader field of Theoretical Informatics, but because this field is vast, a further taxonomy of the project is needed.

1.1.1 Academic context

The main objective of this project is to answer some open problems in the topic of geometrical separability, within the field of Computational Geometry. This will entail the modification of an existing algorithm and demonstrations about the properties these modifications achieve. The problem was initially proposed in the PhD thesis of my director Carlos Seara, published as part of the DCCG research group. Some brief introductions of the thesis and the field are needed in order to understand the context of this project.

Belonging to the department of mathematics and also the FIB, the UPC *Research Group on Discrete, Combinatorial and Computational Geometry* (DCCG) has active research in the field of computational geometry (as the name suggests). As a member of DCCG, my director's PhD [11] presents a solid foundation, and is a necessary element, for the contribution of this project to be possible. As an undergraduate student at the FIB, this project is a small addition to the work already published.

1.1.2 Computational Geometry & Geometric separability

The field that the project is framed in is computational geometry. In a narrow sense, computational geometry is concerned with computing geometric properties of sets of geometric objects. In a broader sense, it is concerned with the design and analysis of algorithms for solving geometric problems. In a deeper sense, it is the study of the inherent computational complexity of geometric problems under various models of computation.

As the core element of this project, the PhD thesis "On Geometric Separability" [11] is a compendium of problems and algorithms with general goal of separating a set of objects with colours into monochromatic subsets. A more formal definition is that geometrical separability is a topic that centres in finding "geometric separators": a finite set S of surfaces is a separator for coloured object sets in $\mathbb{R}^d$ if the connected components in $\mathbb{R}^d - S$ are monochromatic.

Let's consider the following generic problem: we are given two disjoint sets of objects in the space (polygons, points...), classified as red objects and blue objects. Does a surface of some specific type exists that separates the red objects from the blue objects?

This question can be asked for different kinds of objects and separators in spaces of any dimension, and even for more than 2 colours. This family of problems is what is understood as the topic of geometrical separability.

For the open problems that concern us, objects will be points in $\mathbb{R}^2$, and the separator of objects will be a line (hyperplane) also in $\mathbb{R}^2$. Some basic results on this basic notion of separability are that two object sets are line separable if and only if their convex hulls do not intersect [12]. It is also well known that the decision problem of linear separability (i.e., the separability by a line) for two disjoint sets of points, segments, polygons, or circles can be solved in linear time [7]. So our contributions need a final expansion of this natural notion of separability.

Our separability criteria will involve a set of separators instead of only one. We will allow more than one line as separators, but they must all be parallel. The number of lines will be denoted by the variable $k \in \mathbb{N}$.

The problems can easily be formulated slightly differently to change the format of the answer. One could ask for the direction that yield the minimum amount of lines, or instead ask for the intervals of orientations in $\mathcal{S}^1$ (the unit circle defining the set of all the orientations in the plane) such that the objects can be separated for a fixed number k of lines. Even transform into a decisional problem by asking if there exist some orientation in $\mathcal{S}^1$ s.t. the objects can be separated with a fixed k number of lines.

1.2 Justification

Because of the close relationship with the source material, justification for the project needs of a deeper review of the already existing results.

1.2.1 Results of the source material

As already mentioned, there are lots of other families of problems if we choose different definitions for our objects sets and our separator sets. And just as numerous are the findings contained in the source material. Because the open problem builds on the findings of the thesis, those must be discussed before enunciating the open problem in its full context.

The first result is a sweep line algorithm, that takes advantage of the dual representation of the problem. Without going into details, the result is that the minimum amount of lines necessary to separate the sets can be determined by an algorithm which uses $O(n^2 \log n)$ time and $O(n^2)$ space, also yielding the corresponding slope intervals (Theorem 5.8 of [11]). So, any further improvements must be at least better than this generic algorithm. This result is used as a complexity to beat, the project won't build upon it. Reading the source material is recommended to understand the motivation of the project in more detail, but the dual algorithm will not be described or discussed.

The second relevant finding aims to prove that faster algorithms exists for particular cases. If $k \leq 4$, then there exists an algorithm that solves the problem in $O(n \log n)$ time and $O(n)$ space. But only under a certain constraint on the points sets.

This algorithm uses the rotating calipers technique and some properties of the points to dynamically rotate across all orientations, checking efficiently if the sets are separable. Because this project builds on this algorithm, it will be described in detail in further sections.

1.2.2 Open problem & goals

Now that we have a brief overview of the topic, we can finally specify the open problem:

Open problem 1. *Let B (blue) and R (red) be two disjoint point sets in the plane. Can it be decided whether the sets are red/blue/red/blue/red separable by four (non-oriented) parallel lines in $O(n \log n)$ time and $O(n)$ space, if the convex hull of the blue points does not contain any red point?*

As the open problem reveals, the original algorithm described in the PhD achieves the stated complexity, but at the cost of needing the stated constraint over the points.

Our main goal is to expand the capabilities of the already existing algorithm for the $k = 4$ problem, attempting to generalize it so that it can solve the problem without the constraint. As this hasn't been possible, we would at least desire to relax as much as possible the constraint. But always remain below the $O(n^2 \log n)$ time complexity, as any slower results are irrelevant.

This expansion of the algorithm is performed by some natural extensions to the original work. The main focus of the project is demonstrating that those extensions are worthwhile because they preserve the correctness and other properties of the original algorithm, plus relaxing the constraint over the input[1].

1.2.3 Research implications

To finally conclude the enunciation of the open problem, it is adequate to consider the implications that justify it as a worthy project.

A second open problem is posed in the PhD following the one enunciated:

Open problem 2. *Let B and R be two disjoint point sets in the plane. Can it be decided in $O(kn \log n)$ time whether the sets B and R are separable with at most k parallel lines for $k \geq 5$?*

As it is revealed in this second open problem, the aim is to step by step generalize the algorithm. If a general algorithm for $k = 4$ is found, maybe the task of generalizing it for any k is more feasible. This could be further work for other projects, and in general a positive impact to the research stakeholders.

[1] The constraint is that there must exist a red point inside the blue convex hull

1.2.4 Why are those problems important?

As for why those two open problems are relevant, the main reason is that they belong to already existing literature. In general geometric separability is a very theoretical topic, but as always, industrial applications are endless. Separating objects might mean separating data points of all kinds. From segmenting maps into districts or terrains into parcels, to the separation of the smallest chemical compounds, including all kind of optimization problems involving classification.

A second reason is the intertwined nature of both questions. Advances in relaxing the input constraint for the first problem could be very useful in attempting to build recursive algorithms to solve the second question. Any advances on the first question could be compounded as breakthroughs in the second question. This alone is enough to motivate this project.

A last but very persuasive reason is that having the chance of working with the author of the source material is a great opportunity. Not embarking in this project would be missing a very valuable learning experience.

1.3 Scope & Risks

My vision is that this project attempts to produce results similar in form to an article by a member of a research group, although with much more humble results. The publication of the results is not the end goal of this project, due to the uncertain nature of obtaining relevant results. But as a bachelor thesis in research, it is imperative that non-trivial results are formalized up to the expected quality and elegance. In order to provide a more adequate focus to the project, and make it possible to follow the evaluation guidelines, this project could be understood to be contextualized as theoretical research within a fictional institution/company.

1.3.1 Stakeholders & scope

As an official thesis from a student of the FIB, the UPC "acts" as the journal the article will be published in. So a primary stakeholder is the several actors within the FIB that will be supervising the correctness and formality of my proofs. This includes my director, who is evidently interested in the resolution and study of the open questions posed. It also includes the rest of the research group, who could use these advancements as stepping stones in further research.

Other actors are the rest of the research community, who would be the main beneficiaries of this project. No direct tangible and quantifiable benefits to specific corporations, state institutions or social communities are expected to arise from this project. But other future articles or more efficient implementations of the algorithm might build upon my contributions, thus having a secondary impact.

This environment of actors opens the opportunity for this to be a pedagogical environment, expanding the scope of the project to include the development of my formal writing and research skills. Skills involving problem-solving have manifested and developed, but also skills used in the adequate presentation of the results. To accomplish this second half of the pedagogical scope, the project has also consisted of external work on building the necessary background on several adjacent topics.

1.3.2 Risks

All projects have risks. Some threatening this one are:

Inconclusive results

As already explained, the main risk of this project is not finding relevant results.

Incompatible evaluation

This second risk has appeared during the execution of the Project Planning family of tasks. It is possible that the evaluation method of the bachelor thesis is incompatible with the format of the project. This problem points to a general complication that arises from the evaluation of the general competences of the TFG. Due to the non-industrial nature of the project, it is very difficult and time-consuming to adapt it to fit the evaluation rubrics.

Deadline

Another risk is the deadline of the project. This is a double edge risk because the speed at which breakthroughs are expected to be done is always uncertain. This means that it is very difficult to select an exposition turn for the project. If all breakthroughs had happened very fast, spare time would have appeared at the end of the project. Conversely, if demonstrations had been much more difficult to formalize than expected, maybe the project would have suffered delays, pushing the deadline dangerously.

Inexperience

As the first project of this scope I perform, most skills involved in the proper formalization of proof remain to be improved. Even if I knew how to execute the project, for some micromanagement decisions I still needed supervision.

1.4 Methodology

The ongoing work in this problem started before it was constituted as a bachelor thesis project. And it is one among several of the problems I was working in parallel, all in the context of building a background in several topics of computational geometry. This means that the planning of the project concerns only the proper writing and formalization, as the "design phase" of the solutions happened outside the scope of the timetable. Unfortunately I am forced to include a speculative list of tasks that act as the ones I performed in the design of the algorithm, take them with a grain of salt.

With this consideration, the project already started from a draft of the demonstrations of the results, that needed to be formalized further. This gave a complete guideline of all the work to do, providing a road-map that was to be completed sequentially. This is because all demonstrations must rely on the previous one being completely proven. And due to the highly specialized requirements of each demonstration, it is most efficient to achieve a full proof instead of gradually formalizing simultaneously several demonstrations.

The methodology consisted of a single sequential list of demonstrations to formalize from intuition (notes), and a posterior work of harmonizing and editing the layout of the document.

This could be understood as a waterfall methodology under some interpretations. Automatic version control was performed by online repositories like Google Drive and Overleaf.

The task management has been autonomous on my side, and it was performed using a paper calendar and checklist. The decision was made to not use online task managers like Trello, because other task managing systems were already in place. Namely, a wall of sticker notes with the tasks, sorted by the sequential order to carry them out.

Completed proofs were shared almost weekly with the director to check the formality and correctness, but only in a need to know basis. This means that even if meetings were performed, they were of a spontaneous nature, to accommodate the schedule of the director and to avoid unnecessary supervision. This was decided on the principle that it would encourage creative problem-solving critically needed to complete the project. It has also been the fruit of the understandably busy schedule of my director, who has nevertheless been very understanding and available.

Regarding the rigour of the methodology of the project, it is very hard to quantify, if not impossible. The veracity of the results of this project rely entirely on the veracity of the lemmas and theorems that will be exposed. They are meant to be scrutinized up to the level required of a bachelor thesis, but just how rigorous that level is, falls totally outside my methodological capabilities.

In summary, even if the strategies I've followed to try and design correct proofs end up yielding the correct result, the statistical veracity of this project is hard to assess. A conclusion will be either true and substantiated by the work done, or ambiguous and found to be insufficiently correct. No statistical analysis of the errors made is really possible.

2 Temporal planning

The project started well before the planning of the project, so it is hard to account for all past tasks already performed. The start date was roughly the 01/01/2022 when I first started reading the PhD, and the selected deadline for evaluation is 17/06/2022.

So the workload distribution is theoretically about 450 hours in roughly 138 total working days, meaning an approximate average workload of 3h and 30 minutes. But this ended up not being the real most frequent time slot per day. Work was more frequently performed in 2 slots of 2h in a pomodoro like fashion. This is because one day a week I was able to work for only 30 min, indeed Sundays.

Other time factors included the Easter holidays and other special occasions that restructured the work around the week. Even with these factors included, the task management was mildly overestimated.

2.1 Definition of tasks

Here follows a brief definition of each task by family. The Project Planing tasks run in parallel to the actual project, that consists of the Study of the source material group of tasks, and the Demonstrations group. The only temporal dependency among task groups is those last two, so it was a requisite to finish the Study group to start the Demonstrations group.

The following list describes the tasks belonging to the project planing family. These were heavily based on the assignments to deliver, that in turn had very well-defined sub objectives by virtue of the rubrics used to evaluate them. The rubrics also provided the material needed to read in order to perform each task, meaning that all tasks incorporated reading the suggested material for the assignment.

2.1.1 Project Planing

1. **Scope of the project** :
   First delivery. It includes scope, objectives, justification, methodology and risks.

2. **Time planning** :
   Second delivery. Task and temporal planing of the project, including the Gantt diagram.

3. **Ecological budget report** :
   Third delivery. Analysis of the economical and costs of the project. Including the materials used to prove the theorems, the ecological impacts of the mathematical research, and the budget of drawing images as support material.

4. **Final document** :
   Formatting of all previous tasks into a single document, integrating the GEP tutor feedback.

The next task family is the Study tasks. This family contains all the tasks that consisted of: building background knowledge about the topic, reading the research done over similar open questions, exploring possible modifications of existing solutions & designing new ways of understanding the problem. In summary, this is the design stage of the project, a design that was build on existing work that needed to be heavily researched.

Just as a small reminder, this creative stage was wildly unpredictable, full of uninspired days followed by frustration and maybe, if lucky, eureka. Attempting to divide in tasks this section was specially difficult, so take them as a guideline to how the creative process unfolded (these tasks were executed before the project management started).

2.1.2 Study of the source material

1. **First reading of the PhD** :
   A first autonomous reading of the paper, done to map the different chapters of the material that are relevant to the open questions. This includes selecting the open questions of interest, but also the surrounding chapters about similar problems.

2. **Correctness of the original algorithm** :
   Further reading on the original algorithm, up to the comprehension level that allows for me to demonstrate its correctness without aid of the original material.

3. **Second read of the PhD** :
   A second read to start expanding from the relevant chapters into the whole body of the document, once again delving into similar problems to see what techniques can be used to solve this kind of problems. With a special focus on dynamic convex hull algorithms.

4. **Investigation into the dual algorithm** :
   Research of the point-line dual algorithm, that provides the open question ceiling complexity, and other material relating to the point-line dual. Informal consultations with other Computational Geometry teachers on the point-line dual (namely Rodrigo Silveira).

5. **Further reading of auxiliary material** :
   Research of other material of undefined theme, characterized by a more inspirational reading of auxiliary material. This tasks also includes the abovementioned days of probing the open question. A healthy dose of the work hours was shared with activities like sport (ping pong) or cultural events (chess, ballet, cinema) to stimulate the creative process.

6. **First session with the project director** :
   First in person session. A general background of the topic is done, with special attention to how to navigate the source material. This acted as supervision of all previous tasks of research. Eureka intuitions were shared in several late evening meetings after the lectures and classes, possibly overextending the already long schedule of my director.

Finally, the Demonstration's family of tasks. These are the main group of tasks that concerned this project management. As explained before, they had a dependency with the design/research phase of the project. The tasks of this family are the most tangible because they constitute the creation of the actual document that results of the project.

From the initial tasks, undistinguishable from the design phase, to the final document, the expected progression was that first ideas are sorted in a very abstract way. The mental formalization of ideas gave birth to an initial, equally abstract, draft proof. From there the usual formalization process materialized the demonstration, each version of the document more precise and elegant than the last one.

This unfolded more or less as described, although some tasks proved to be harder than expected.

2.1.3 Demonstrations

1. **Initial draft of proofs** :
   A first mental survey of the ideas elaborated on the research phase. New ideas might still be considered. No formalization is done.

2. **Initial correctness of the extended algorithm** :
   An intuitive exploration of the new properties the extended algorithm retains, and which other properties it incorporates. The main goal is to find relevant properties that have been added by the extension.

3. **Second draft of proofs** :
   First document that actually contains a written step by step demonstration. Formalization must be enough for it to be read by the director, and remain tight enough correctness wise.

4. **Correctness of the extended algorithm** :
   Written demonstration of the properties that the extended algorithm exhibits, relevant to the previous proofs.

5. **Second session with the project director** :
   In person, overhaul of the second draft. Re-scoping of the project and reframing of the results.

6. **Explanatory material** :
   Creation of the images used to illustrate the demonstrations and correctness of the algorithm.

7. **Definitive document** :
   Elaboration of the final document. Containing all images needed to exemplify the proofs, all theorems properly enumerated, the academic layout of a research article, and all the requisites of the memory. Possibly overhauling the support media of the project (images or even animations of the algorithm).

As it may be obvious, the Demonstration tasks closely followed the draft of proofs to be formalized. The following list is just a summary enumeration to give some perspective on the amount of work that was planned to be performed and in what order. Bigger demonstrations have ended up having more sub-demonstrations, reflecting the complexity found along the way.

Even if changes weren't expected to happen, as mentioned in the risks, this list was designed to be subject to change. It mainly expanded to include more results that arose along the way, some substituting some results that ended up being irrelevant. This list is not immutable and was never meant to be.

List of demonstrations:

1. Extended algorithm
   - 1.1 Correctness
   - 1.2 Complexity
2. Visibility of star polygon's family
3. Upper bound on guards
   - 3.1 Minimizing guards optimal algorithm
   - 3.2 Convex hull to circle reduction
   - 3.3 Point set to circle reduction
   - 3.4 Outline of the two circles problem family
   - 3.5 Two circles problem as upper bound
   - 3.6 Upper bound of the two circles problem
     - i. Covering angle α of a guard is a function of the circle's radius
     - ii. Number of guards if a function of α

As it can be clearly seen, the most fundamental ideas for the project were explored very early on. On the other hand, it is also evident that some re-scoping of the project happened towards the end. This is discussed in later sections.

One task done implicitly during the whole thesis was documentation. This means documenting the progress of the project weekly in order to perform a review of it at the end. Also, a last task added was the preparation of the oral defence. If it is true that the visual material of the oral presentation is included in the Explanatory material task, an extra time slot for the task of preparing the oral defence was awarded as an overestimation.

2.2 Time estimations and dependencies

Project Planing : This first family of tasks were to be performed in accordance with the timetables set by the Atenea calendar. This renders all the tasks sequential in dependence. The span of the whole family was 4 weeks, roughly one per task. Because this family of tasks started after the main family of tasks, the workload was somewhat reduced to accommodate the ongoing execution of the main family of tasks. A total of 40h were assigned, in 5 slots of 2h a week.

Main workload : The second & third family of tasks was scheduled to meet the deadline of the second defence turn, unless a critical eureka was achieved. This means that the deadline for the evaluation of the project by the director was the 17/06/2022, a whole week before the real deadline. And the presentation and defence of the project in the 27/06/2022 - 01/07/2022 range. The draft tasks were and have proven to be impossible to estimate in time, so a very rough minimum time was proposed. But in no way this represents the real time spend in the tasks, that is much more than declared.

Code	Name	Time	Dependencies
A1	Scope of the project	10	-
A2	Time planning	10	A1
A3	Ecological budget report	10	A2
A4	Final document	10	A3
A	**Project Planing**	**40**	-
B1	First reading of the PhD	30	-
B2	Correctness of the original algorithm	45	-
B3	Second read of the PhD	30	B1 B2
B4	Investigation into the dual algorithm	35	B3
B5	Further reading of auxiliary material	30	B4
B6	First session with the project director	40	B5
B	**Study of the source material**	**210**	A
C1	Initial draft of proofs	35	-
C2	Correctness of the extended algorithm	35	C1
C3	Second draft of proofs	35	C2
C4	Second session with the project director	35	C3
C5	Explanatory material	35	C4
C6	Definitive document	35	C5
C	**Demonstrations**	**210**	B
D	**Documentation**	**20**	-
E	**Oral exposition**	**10**	C
T	**Total**	**490**	-

Table 1: Summary of tasks.

This concludes the overview of the initial planing of the tasks. But as the project developed, some minor changes affected the temporal planning of the project.

2.3 Timetable deviations

Concerning the deviations that have affected the timetable of this project, they only amount to minor expected changes. The project ended up being awarded the 01/07/2022 date to perform the presentation. This added an extra 4 days to the project that were not accounted for. They were spent refining the quality of the project overall, and are just an extension of task C6.

2.4 Resources

Although very few, some resources have been needed to carry out the project. They are classified in the following groups.

2.4.1 Human resources

The main resource of the project, it constituted most of the (fictitious) expenses. Both the director and author of the project are fundamental to the development.

Other human resources include a lot of roles, most of them auxiliary or secondary actors like teachers, friends and support & other professionals at the university (PDI/PAS). Although they are (appropriately) not included into the monetary budget, they are an invaluable resource that is mostly free at the point of use.

2.4.2 Intellectual resources

Because the intellectual resource, the PhD thesis, on which this project is based is open access, no cost will be accounted by direct use of intellectual rights. But lots of the auxiliary research material used in the design of the algorithm is not free. All this non-free material that has been accessed has been free at point of use. This is because most have been books and intellectual material, accessible to me through the different institutions of the UPC. They will be included as costs, but heavily amortized.

2.4.3 Software

Other resources have been the software used. This includes free office suites like Overleaf for LaTeXcompilation. And supporting latex software like Ipe, used in for the creation of images. And finally, but not less important, the UPC suit of software, from its webmail, to Atenea or Raco. These costs will not be included, as they are free in relation to the scope of the project.

2.4.4 Hardware

Finally, hardware resources have been the less consumed in this project. Only basic office material consisting of:

- pens
- pencils
- erasers
- official UPC paper
- official UPC folder

2.4.5 Infrastructure

Infrastructures used include UPC facilities for the digital access to Overleaf and all the software resources mentioned above. No personal computer was used in this project, instead using the ones belonging to the UPC facilities. These facilities only include the ones freely accessible to the students. All the above-mentioned material is also at the disposal of students free of charge by the UPC.

2.5 Gantt

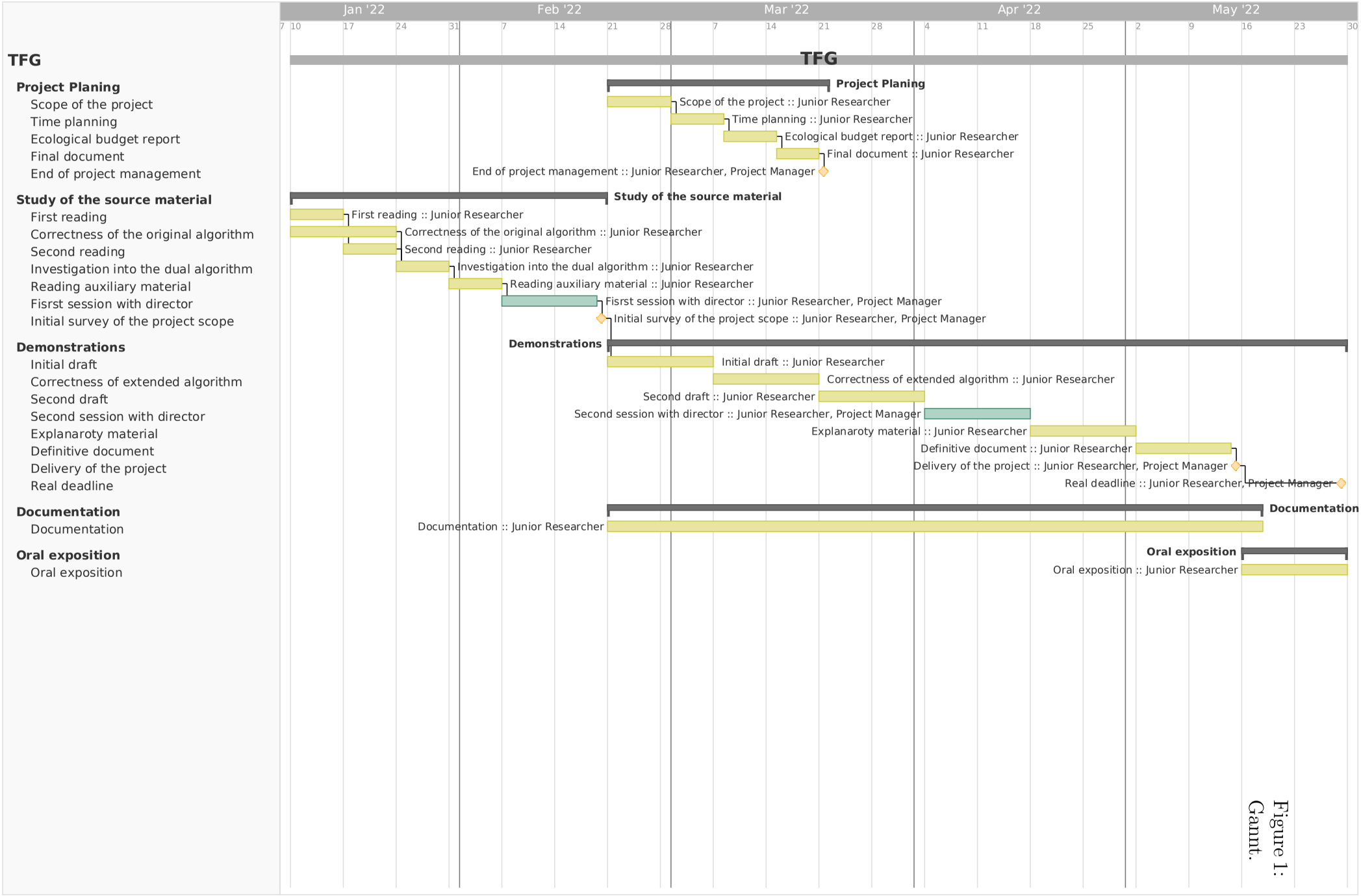

Figure 1: Gannt.

2.6 Risk management

As already mentioned, there have been potential risks threatening the completion of this project. A detailed analysis of these is presented below.

2.6.1 Inconclusive results

Impact : Low
Counter-measures :
This has been already partially mitigated by the measures explained in previous chapters, namely performing an intuitive exploration of what demonstrations could yield interesting results. This mitigation measures were effectuated as part of the first meeting task and the first draft task. As the project advanced and fewer demonstrations remained to be proved, the space for critical failure of the project decreased. Fortunately, this risk was totally mitigated

2.6.2 Incompatible evaluation

Impact : High
Counter-measures :
The only possible mitigation was to consult with the GEP director about the evaluation rubric for the First Event, and reformat adequately the documentation. But as mentioned, the research format of the project was proving very difficult to comply with the rubrics. This risk is unassailable and will continue to exist until the final grade of the project is awarded.

2.6.3 Deadline

Impact : Low
Counter-measures :
Mitigation measures consisted in overestimating the hardness of the tasks. A second measure has been proposing follow-up work, and the creation of support material for the presentation, like animations of the algorithm. Animations and similar content can easily take an undetermined amount of time that in case the project finishes early. A secondary alternate objective also discussed, is the coding of the algorithm for demonstration purposes, although this has been considered very undesirable in comparison to the previous extra work, it ended up being the best option.

2.6.4 Inexperience

Impact : Medium
Counter-measures :
This risk was to be expected, and is part of the learning process. It is important to work on gaining experience, but as the first project of this kind, no mitigation measures could have been proposed beyond more supervision by the director. But too much supervision is also not a good measure, threatening to stifle the learning process.

3 Budget

This section describes the elements of the expected budget, including staff costs, generic costs & indirect costs. The control measures enacted, and unexpected expenses are listed at the end.

3.1 Staff costs

The salaries are defined as cost per hour multiplied by the number of hours dedicated to tasks. The roles needed for the project are:
Project Manager(P.M.): Carlos Seara & Paola Lorenza.
Junior Researcher(J.R.): Nicolau Oliver.
The tasks of the Project Manager are to supervise the progress and soundness of the results generated by the Junior Researcher. Both roles assist in the meeting and review tasks, but only the Junior Researcher participates in all the other tasks.

The salaries have been calculated and rounded with a SS of 130%. The annual salaries have been extracted from the web Glassdoor [3, 4]. A slightly lower salary than the median is due to the inexperience of the Junior Researcher, and the low stakes of the Project Manager. The average hours per year used has been 1850.

Role	Annual Salary(€)	Including SS(€)	**Cost(€)/h**
P.M.	35570 [4]	46250	25
J.R.	21350 [3]	27750	15

Table 2: Hourly cost of each Role.

The P.M. must assist to both sessions with the J.R., meaning that for group B and C of tasks he is awarded 35h of work, a little less than the J.R. And he is also responsible for group A of tasks that total another 40h. The Junior Researcher must perform all tasks except the group A tasks. See Table 3 for a complete breakdown of the working hours.

3.2 Generic costs

Three basic sections of resources are described in the budget. Intellectual material provided by the UPC, the office material also provided by the UPC, and the use made of the UPC facilities.

3.2.1 Intellectual material

Several textbooks about computational geometry have been used as study material to design the algorithm. Because no particular data is available on these book's half-life at the BRGF (Biblioteca Rector Gabriel Ferraté), an average of 30 years of useful life-span for the books is assumed. This is because the books consulted belong to a very low circulation category. For simplicity, assume that the use made of them was of 4 weeks for all of them. See Table 4.

$$\text{Amortization} = \text{Capital expenditure} \times \frac{1}{\text{life expectancy}} \times \frac{1}{\text{days per year}} \times \text{days used}$$

ID	Task	h	P.M.	J.R.	Cost(€)
A	**Project Planning**	**40**	**40**	**0**	**1000**
A1	Scope of the project	10	10	0	250
A2	Time planning	10	10	0	250
A3	Ecological budget report	10	10	0	250
A4	Final document	10	10	0	250
B	**Study of the source material**	**210**	**35**	**210**	**4025**
B1	First reading of the PhD	30	0	30	450
B2	Correctness of the original algorithm	45	0	45	675
B3	Second read of the PhD	30	0	30	450
B4	Investigation into the dual algorithm	35	0	35	525
B5	Further reading of auxiliary material	30	0	30	450
B6	First session with the project director	40	35	40	1475
C	**Demonstrations**	**210**	**35**	**210**	**4025**
C1	Initial draft of proofs	35	0	35	525
C2	Correctness of the extended algorithm	35	0	35	525
C3	Second draft of proofs	35	0	35	525
C4	Second session with the project director	35	35	35	1400
C5	Explanatory material	35	0	35	525
C6	Definitive document	35	0	35	525
D	**Documentation**	**20**	**0**	**20**	**300**
E	**Oral exposition**	**10**	**0**	**10**	**150**
T	**Total**	**490**	**110h**	**450h**	**9500 €**

Table 3: Estimated hours per task.

Books	C (€)	L (years)	A (€)
Computational Geometry (Springer)	51.99	30	0.14
Computational Geometry: An introduction	75.10	30	0.20
Introduction to Algorithms	66.86	30	0.18
Total			**0.52 €**

Table 4: Amortization of books.
C=Cost, **L**= Life expectancy, **A**= Amortization.

3.2.2 Office material

The office material was entirely provided for free by the UPC, but it will be included as consumables for this project.

Item	Cost(€)/unit	Units	Cost(€)
Paper	0.04	200	8
Pen	0.29	2	0.58
Pencil	0.39	2	0.78
Rubber	0.34	2	0.68
Total			**10.04 €**

Table 5: Expenses in consumables.

3.2.3 Facilities & Indirect costs

In order to approximate better the budget of the project, we will include some costs indirectly externalized to the UPC facilities. The BRGF is used as the main building, although the meetings took place in Carlos office, at the Omega building. The most recent and available official report of 2020, states that the final figure allocated to running costs of all the libraries under the UPC is of, 118500€ [1]. The total amount of potential users is 52.726 [1]. Because the use made of internet resources might be much more than the average, the average cost per user per year will be multiplied by 10. This will yield the share of the library budged consumed in the year 2020 due to the project's use of the facilities.

$$118500 \text{ euro/year} \times \frac{1}{52726 \text{ users}} \times 10 = 22.48 \text{ euro}$$

This again shows how making use of shared, semi-public, community resources can greatly reduce the running costs of a project.

Other indirect costs like travel costs have not been included, as they are accounted as expenses externalized on the employees. So, they are paid indirectly though the salary of both roles.

3.2.4 Summary

Because no service or commodity is produced in the project, no tax needs to be applied as sale tax. All the costs included have been calculated after tax.

Description	Cost(€)
Books Amortization	0.52
Consumables	10.04
Facilities Amortization	22.48
Total	**33.04**

Table 6: CG summary.

Because of the high amount of risk present in the project, it is necessary to do a provision of some extra costs that might appear during the project.

Incidental costs

The impact of most of the risks assessed translate into an increase of hours needed to complete the project. To account for this, an emergency budged sized at

15% of the original PCA cost is included. This will be allocated in case that extra hours are needed to finish the project.

The general costs of the project are very amortized, so it is not expected that unforeseen expenses will appear. But due to the increasing energy prices that are having a severe impact across all infrastructure, an extra 40% of CG costs will be included as contingency.

Final budget
Finally, aggregating all the previous sections:

Activity	**Cost(€)**
PCA	9500
Contingency	1425
CG	33.04
Contingency	13.22
Total	**10971.07€**

Table 7: Total cost summary.

3.3 Management Control

Following the motorization of changes on the costs of the project, several indicators are computed.

- Estimated cost by task: the estimation provided above.
- Real cost by task: the real amount of time spend per task. This was to be used to recalculate the cost of staff. Also monitoring the specific kind of task that generated the deviation, forecasting future deviations of similar tasks.
- Deviation: the dynamic deviation as the project advances, monitoring the impact of the risks as it manifests as expenses, or lack of expenses.

The deviation is the one used to review the management of the project. As all the expenses incorporated into the budged have never materialized, no deviation has been detected.

In summary, the **project hasn't had any deviations with respect to the budged** presented above.

4 Sustainability report

This section consists of two subsections. One pertinent to my assessment on the value of generating the sustainability report. A second one reviewing the sustainability of the execution of the project.

4.1 Self assessment

It's easier to imagine the end of the world, than a modest but irreversible reform to the fundamentals of our extractive based economies. Sustainability and ecology concern our present, and the apparent impossibility to change it. The impossibility of the continuation of our current understanding of things. The impossibility of our current way of living, remarkable as it is that we have achieved such an apparent impossibility.

This might inspire lots of fear, anguish and horror. Mental illness, fruit of the "slow cancellation of the future". Because it is easier to imagine that the utopian film of "our future" has been cancelled, than to imagine the cancellation of the next TCU blockbuster.

So it is critical that our reaction, anger, and answers to those fears is appropriate. With this in mind, assessing the sustainability of my Bachelor Thesis is the perfect place to start learning how to tackle these environmental responsibilities. And it is of vital importance to take part in activities like the environmental questioner, to **inspire us**, the future generations, with vitality and tools to fight for their (now in question) future.

The self assessment consisted of several discrete quantitative answers that abstracted and captured in a numerical slider the environmental dimension of my project. This reflects perfectly the **systematic, rigid, and well-organized** way in which the ecological report must be generated and **submitted**. It is imperative for the advancement of our common needs, as one of the species on earth, to elaborate with no hesitation these itemized and rubric based environmental evaluations.

Practising these environmental analysis exercises has greatly improved my ability to perform environmental reports, giving me a guideline to follow step by step and in an analytical optic.

It has also improved my critical skills as much as they are mentioned.

4.2 Environmental impact

¿Have you quantified the environmental impact of the execution of the project? ¿What measures have you taken to reduce the impact? ¿Have you quantified the reduction? The environmental impact of the project has been absolutely minimal. The only secondary impacts one could argue can even be attempted to be measured are the footprints of the workers. But no product or service has been produced that consumes resources in a primary manner. So no quantifications, reductions, and obviously quantifications of the reductions have been performed.

If you executed the project again, ¿Could you perform it with less resources? No, all resources were strictly necessary.

¿What resources do you estimate the project will use over its life span? ¿What will be the environmental impact of these resources? It could be argued that, in the unlikely case that this bachelor thesis is stored in a digital archive, the lifetime impact would be the one produced by this storage. Even with this in consideration, no meaningful resources will be used, so the impact is null.

¿Will this project allow reducing the consumption of other resources? ¿In a global assessment, does the project improve or deteriorate the environmental footprint? No impact on resource consumption is expected. If the algorithms were to be implemented, they would be more efficient. But as there is no current implementation as far as I know, there is no improvement to be done. In any case, implementation is totally outside the scope of the project, and it shouldn't be accounted. So it neither helps nor aggravates the current footprint of any relevant processes.

¿What scenarios could increase the ecological footprint of the project? None specific to this project.

4.3 Economic impact

¿Have you quantified the cost of executing the project? ¿What decisions have you taken to reduce the cost? ¿Have you quantified the reduction? This first two questions are specifically answered in Section 3. No quantification on the reduction has been performed, as none of the expenses has materialized, as exposed in the pertinent section.

¿Has the cost been the same as expected? ¿Are the deviations justified? This is also specifically answered in Section 3. No deviations have happened.

¿What cost will the project generate over its lifespan? ¿Could these be reduced to increase viability? None that are relevant to the scope of the project. No reductions are possible.

¿Have the cost of updates/repairs been included? Fortunately, the cost of "updating/repairing" ideas is still quantified by the effort invested in thinking, not an economic indicator.

¿What scenarios could alter the economic viability of the project? None specific to this project.

4.4 Social impact

¿Has the execution of the project included introspection at a personal, professional, or ethical level for the people implicated? Yes, on all three levels for all the people implicated.

¿Who benefits from the use of the project? ¿Is there a collective that could be affected by the project? ¿In what capacity?This is discussed in Section 1.3.1.

¿Does the project solve the problem initially identified? To the fullest capacity it can.

¿What scenarios could entail a negative social impact for the project with respect to a specific collective? None specific to this project.

¿Could the project generate addictions or over-reliance on the users that could leave them at a disadvantage? Unfortunately, unless a passionate addiction predisposition for geometry is present, most users are safe from this project.

5 The original algorithms

In Chapter 5 of the PhD thesis [11], the geometric separability problem using several strips is discussed, amongst other versions of the problem. Our focus will be in Sections 5.1 and 5.2 (pp. 150–165), where the algorithm is introduced and described. The following summary of the original text is meant to serve as a rephrasing, it is recommended to follow the source material for a better more accurate explanation.

5.1 Notation & preliminaries

Let B and R be two finite disjoint sets of n points in $\mathbb{R}^2$, namely B the blue points and R red ones. Let the convex hulls be noted as $CH(B)$ and $CH(R)$ respectively. As R and B are the inputs to our algorithms $|R| = |B| = n$. The solutions that the algorithms report are a list of angle intervals in the range $[0, \pi]$, representing (angles) orientations that are separable[2]. Finally, the variable $k \in \mathbb{N}$ is the number of separators (parallel lines) used to separate R and B.

Before constructing the algorithm for separability with $k = 4$ (using 4 parallel lines), a simple algorithm for solving the $k = 3$ problems in $O(n \log n)$ time and $O(n)$ space is described. This in turn also needs of describing and executing a simpler algorithm for $k = 2$ with the same time and space complexity. This is because before attempting to find separability using more lines, it simplifies the algorithm to somehow discard orientations that are separable using fewer lines, as those will obviously be separable using more lines.

For all k, separability will place separators between subsets of R and B, '/' will notate those parallel lines. When referring to a $red/blue/red/\ldots$ the notation for the subsets will be $R_1/B_1/R_2/B_2/\ldots$ And the separators will also be named in this order, being s_1 the first parallel line from left to right, followed by the rest of the separators $s_2, s_3 \ldots$

Some basic equations over the subsets are:

– They are disjoint. A point can only belong to one of them at any given time (i.e., orientation).

$$\forall i\ \forall j,\ i \neq j, \quad R_i \cap R_j = \emptyset \qquad\qquad \forall i\ \forall j,\ i \neq j, \quad B_i \cap B_j = \emptyset. \tag{1}$$

– The union of all red subsets equals R, and the union of the blue subsets equals B, i.e.,

$$\bigcup_{i=1}^{\lceil \frac{k+1}{2} \rceil} R_i = R \qquad\qquad \bigcup_{i=1}^{\lfloor \frac{k+1}{2} \rfloor} B_i = B. \tag{2}$$

[2]The angle intervals can also be given in $[0, 2\pi]$. If the usual range $[0, \pi]$ is used, it implies that the $[\pi, 2\pi]$ orientations have the same separability as their equivalent in $[0, \pi]$.

Now focusing on consolidating the *red/blue/red/...* the notation, we show how it simplifies the separability problem.

Assuming a minimal k, for any direction $d \in \mathcal{S}^1$ only two possible orderings of the subsets are possible; *red/blue/red/...* the first subset from left to right is red, or *blue/red/blue/...* the first subset is blue.

By a strip we denote the region in the plane between two parallel lines or the (infinite) half-plane defined by a line. We can use either strips or lines to define monochromatic regions in the plane, i.e., the separability by k lines or k-line separability corresponds to the separability by $(k+1)$ strips or $(k+1)$-strip separability

Lemma 1. *The k-line separability (or the $(k+1)$-strip separability) has the same asymptotic computational time complexity for the red/blue/red/... separability as for the blue/red/blue/... separability.*

Proof. If an algorithm A computes the *red/blue/red/...* separability for a given R and B with time complexity $O(\tau)$, then by renaming and swapping R and B, it can also compute in $O(\tau)$ the *blue/red/blue/...* separability. Merging the two results using the union of angle intervals, it yields the separability of the sets R and B independent of the order of the subsets.

The cost incurred has been: executing the algorithm twice, renaming the inputs (linear time) and merging the angle intervals. Because the angle lists are sorted, they can be merged in linear time. So the total cost is $O(\tau + n)$, and assuming that the algorithm has $\tau \geq n$, this reduces to $O(\tau)$ time. □

With this reduction in mind, only the *red/blue/red/...* separability will be considered, as to simplify the problem.

A very useful technique will be the use of rotating calipers. As the caliper lines will often be used as separators, it is convenient to address the caliper by the convex polygon or a convex hull (of a set of points) it rotates over, or the separators it constitutes. In this last case, notation such as $s_i - s_j$ is used.

Another recurring use of the calipers is the orientations in which a red point is inside a caliper, so some notation is needed. All calipers used will rotate over convex polygons clockwise from 0 radians to π radians[3]. So, all points outside the polygon enter and leave the caliper only once[4]. And all points inside the polygon are always inside.

After computing the supporting lines from a red point with respect to a convex polygon (or the convex hull of a set of points), the slope of those lines indicates an interval of directions for the caliper such that the red point lays inside it. When a red point is inside the caliper, it will be said to be "alive", entering being its "birth" and leaving its "death". One of the supporting lines corresponds to the birth, the other to its death. See Figure 2.

The slopes of these supporting lines are calculated with respect to the horizontal, also clockwise. As if following the orientations of the caliper.

[3]The rotation can also be over $[0, 2\pi]$ respectively.

[4]Or twice in a $[0, 2\pi]$ turn, in summary a constant number.

This means that if the angle of a slope was bigger or smaller than the $[0, \pi]$ interval, it can be transposed it back into it. We can capture this with the following equation:

$$\forall j \in \mathbb{Z},\ \forall \alpha \in [0, \pi] \quad \alpha \equiv \alpha + 180j \tag{3}$$

Defining these slopes is important, as we will use these angles to compare support lines. Further clarification is done in following sections, specially if when using the $[0, 2\pi]$ interval instead of the usual one.

5.2 Single strip separability, $k = 2$

The Lemma 1 applied to the specific case of $k = 2$ yields a *red/blue/red* separation. Using, Equation (2) it is clear that because B_1 is the only blue subset, it must be equal to B. With this in mind, the separator lines s_1 and s_2 must separate the $CH(B)$ form R_1 and R_2. In order to do so, the rotating caliper on $CH(B)$ will be used as s_1 and s_2. The caliper is formed by two parallel lines tangent to $CH(B)$ (or parallel supporting lines of $CH(B)$) with a given direction. These lines are minimal in width in each direction, because if they were moved slightly closer to the $CH(B)$ at least a blue point would lay on a red strip.

The following proposition is straightforward.

Proposition 1. *The sets R and B are 2-line separable (or 3-strip separable) for a given a direction d (i.e., $k = 2$) if and only if they are separable using a strip of minimum width.*

So, by rotating the caliper over $CH(B)$, which corresponds to checking for the minimum width strip, the algorithm finds all the directions in which the sets are separable.

While rotating the direction in which to calculate the caliper lines, the tangent vertex to the $CH(B)$ (or supporting vertex of $CH(B)$) changes. These pair of vertexes are known as antipodal vertices of $CH(B)$, and it is a well known result that there are a linear number (in the number of vertices of $CH(B)$) of such antipodal pairs. The article by Toussaint [15] also explains in more detail how the rotating caliper technique is used to solve this geometric problem.

Expanding the tool of the rotating caliper, the next step is to check if for a given direction the strip is empty of red points, i.e., the strip is monochromatic. For this, we use the support lines of the red points as introduced in the Section 5.1. The lines that are tangent to $CH(B)$ and contains the red point r_i are called *supporting lines of r_i with respect to $CH(B)$*. It is a well known result that computing the supporting line of a point exterior to $CH(B)$ can be done in $O(\log n)$ time [10].

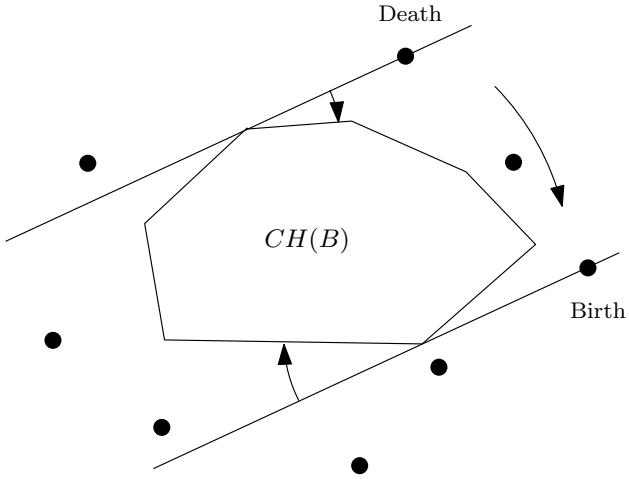

Figure 2: Rotating caliper around $CH(B)$.

Recall, there are two of such lines, a birth one and a death one. See Figure 2. And each of this line has a slope, α and β respectively, clockwise with respect to the horizontal. Using Equation (3), we find the equivalent slope angles of these lines belonging to $[0, \pi]$. These angles define an interval $[\alpha, \beta]$ of directions in which the point r_i lays inside the caliper. But because of the cyclical nature of the directions in S^1 , some red points have $\beta < \alpha$. In this case, the interval $[\alpha, \beta]$ will be a shorthand of $[0, \beta] \cup [\alpha, \pi]$. All these intervals will be referred to as *the living angle interval of* r_i.

As there are n red points in R, the complexity of computing all supporting lines will be $O(n \log n)$ time. From the support lines, the living angle intervals are build, resulting in all the intervals where there will be a red point in the caliper.

After computing the union of these intervals, the complementary set of the union of these intervals with respect to $[0, \pi]$ (also a set of intervals), will indicate the set of directions such that no point is inside the caliper. If no point is inside the caliper they will all be either in R_1 or R_2, so R and B are 2-line separable.

The algorithm exposed has to deal with some extra caveats:

– The first detail hidden in the code is that the order of the supporting lines obtained at line 9. They must be ordered by slope such that the interval $[\alpha, \beta]$ truly represents the interval of directions in which the red point r is inside the caliper. For simplicity, it is assumed that the function $SupportingLines()$ uses the same format as the one exposed at the end of Section 5.1. So the supporting lines are ordered first the birth one as l_1, then l_2 as the death supporting line.

– A second detail omitted has been how to compute the union of intervals at line 10. Supposing that the variable *intervals* maintains an ordered list of the intervals already merged, this is trivial. More sophisticated data structures for intervals can be considered, like segment trees, but are not necessary for this algorithm.

Algorithm 1 Separability $red/blue/red$, $k = 2$

Input: R, B
Output: $intervals$

$CH(B) \leftarrow ConvexHull(B)$
$L \leftarrow \{\ \}$ ▷ Empty list
for $r \in R$ **do**
  if r is inside $CH(B)$ **then**
    return $\emptyset$
  else
    $(l_1, l_2) \leftarrow SupportingLines(r, CH(B))$
    $\alpha_r \leftarrow slope(l_1)$
    $\beta_r \leftarrow slope(l_2)$
    if $\beta < \alpha$ **then**
      $L.append([0, \beta_r])$
      $L.append([\alpha_r, \pi])$
    else
      $L.append([\alpha_r, \beta_r])$
    end if
  end if
end for
$interval \leftarrow \bigcup_{i=1}^{|L|}[\alpha_i, \beta_i]$ ▷ Union of all elements of the list L
$intervals \leftarrow [0, \pi] \setminus intervals$ ▷ Complementary interval
return $intervals$ ▷ List of intervals that are 2-line separable

In a more detailed inspection of the code, it follows that the algorithm correctly separates the sets R and B even in strange edge cases, such as $R_1 = \emptyset$ and/or $R_2 = \emptyset$. The only edge case not covered is $B = \emptyset$, that we consider a degenerate case not worthy of consideration.

As shown previously, the total cost of the algorithm is $O(n \log n)$ time. This concludes the algorithm for $k = 2$.

5.3 Multiple strip separability

Let's now consider separability for higher values of k, i.e., $k \geq 3$, so using multiple parallel strips.

5.3.1 $k = 3$

Let's increment k to 3 and explain the approach for the odd values of k. In this case the separation will be $red/blue/red/blue$, so the subsets are $R_1/B_1/R_2/B_2$, and the separators s_1, s_2 and s_3 by virtue of Lemma 1.

The first change is to notice that our first and last subsets are now of different colour (R_1, B_2). Recall that it does not matter the colour we assume the left-most set to be, in this case red. By virtue of Lemma 1, we will check the case with $blue/red/blue/red$ separability by re-executing the algorithm with swapped colours.

The caliper now used to define s_1 and s_3 will rotate over $CH(R)$ and $CH(B)$, instead of just $CH(B)$. Thus, s_1 must separate R_1 to its left and B_1 to its right, so it must be rotating over $CH(B)$.

Similarly, s_3 must be rotating over $CH(R)$. This rotation over different polygons is also described in more detail in the article by Toussaint [15]. See Figure 3.

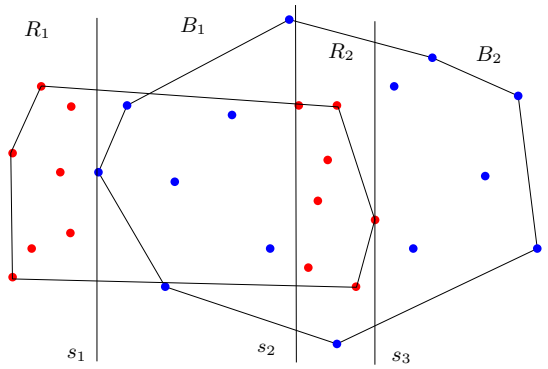

Figure 3: Rotating calipers around $CH(R)$ and $CH(B)$.

This difference has implications over what ranges must be considered for the rotation. Before, the caliper was confined to the range $[0, \pi]$, because the other half of the complete rotation was a symmetric repetition of the first. As this is no longer the case, a complete range $[0, 2\pi]$ for the directions must be traversed. This means that now the solution will be a list of intervals, all belonging to $[0, 2\pi]$. The direction of the supporting lines will also be considered from $[0, 2\pi]$, so we don't apply the Equation (3) for this algorithm. Instead, the angles of a supporting line will again be the angle defined by the horizontal clockwise, but now allowing the full range.

The first step of are now defined also the algorithm is to include as solution all directions that are separable using only 2 lines ($k = 2$) executing Algorithm 1. This simplifies the cases to be treated, as now only consider orientations that need at least 3 lines to be separated. Because the strategy of calling the algorithm for $k - 1$ will be used again, some considerations are worth explaining. The strategy used to merge the solutions of Algorithm 1 with Algorithm 2 has been to perform the union of the two outputs. Because the Algorithm 1 is guaranteed to find all directions that are separable using $k \leq 2$, the only directions that are left to be found are the ones separable by exactly 3 lines. When searching for those, the algorithm is allowed to mislabel the orientations separable by less than 3 lines as non-separable. It does not matter, as any mistakes will be "overwritten" when the union with the Algorithm 1 output is performed.

Now that we have this strategy in place, a new property is introduced to simplify the algorithm. The following proposition states that some separators are "worthless". A separator is referred to as worthless, precisely if in a direction d it can be removed and R and B remain separable. If some separator is worthless when attempting to k-separate R and B in the direction d, it means that R and B are at least $(k - 1)$-separable. And as just shown, those directions can be ignored and assumed to not exist, as the algorithm isn't required to be correct in those directions.

Proposition 2. *In attempting to separate R and B in a direction d using k lines, we can have the following cases:*

1. *If k is odd then:*

   If $R_1 = \emptyset$, s_1 is worthless.

   If $B_{\frac{k+1}{2}} = \emptyset$, s_k is worthless.

2. *If k is even:*

   If $R_1 = \emptyset$, s_1 is worthless.

   If $R_{\frac{k+2}{2}} = \emptyset$, s_k is worthless.

3. *If any s_i is worthless, R and B are $(k-1)$-separable at direction d.*

Where s_k is the right-most separator, and $B_{\frac{k+1}{2}}$ is the rightmost blue subset. The same for the $R_{\frac{k+2}{2}}$ subset.

Here we show a very graphical example of Proposition 2: One can clearly observe that $CH(R)$ contains $CH(B)$, so if we are attempting to find orientations with *red/blue/red/blue* separation, we shall never succeed, or B_2 must be empty. See Figure 4. If B_2 is empty then s_3 is worthless, and what was really found is a $k = 2$ separation, not really an interesting $k = 3$ separation. This is why we don't bother with orientations detected as $k = 2$ separable, and reduce the case analysis.

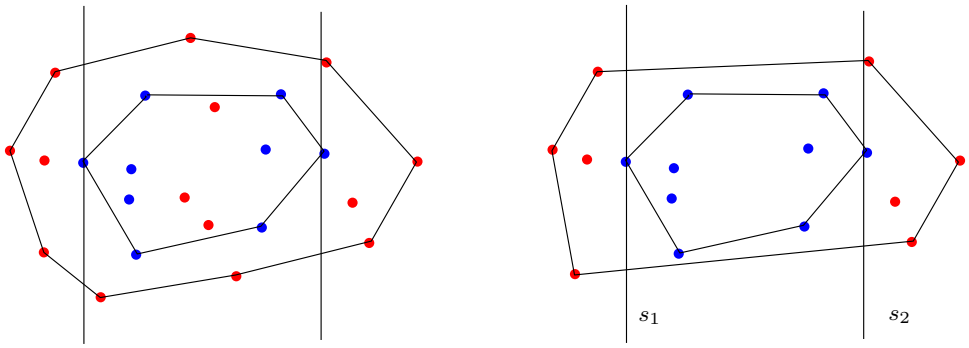

Figure 4: $CH(R)$ containing $CH(B)$.

Use the caliper described above as s_1 and s_3, again, computing the "alive" intervals of the relevant points. This time, there are red and blue points inside the caliper $s_1 - s_3$. What is needed is a way of partitioning the red (blue) points that belong to R_1 (and B_1 respectively) from the ones in R_2 (B_2). Recall that by Equation (1) they are disjoint, so if one of the two subsets is known, the other is immediately follows. In this case, being "alive" for a red point will mean belonging to R_2, and for a blue point belonging to B_1. "Dead" meaning they belong to the other subsets, the ones outside the caliper. See Figure 3.

This again can be computed by obtaining the angle of the supporting lines. As expected, the supporting lines from the blue points are calculated with respect of $CH(R)$, vice versa for the red points as in the Algorithm 1. Again, obtain a "birth" and "death" events associated to a supporting line as the caliper rotates clockwise. Sorting all these events by the angle they happen, yields a "chronology" of a linear number of insertions or births and deletions or deaths of red and blue points into R_2 and B_1. The insertions and deletions, referred to as events, use the notation e_i. The list of these events is denoted as E.

This list E defines the insertions and deletions of the points into the sets R_1 and B_2. A deletion from one of the previous two sets, corresponds with an insertion into the corresponding sets (R_2, B_1), another result stemming from Equation (1) and Equation (2). This sequence of insertions is the C_{cso} (Counterclockwise Sweeping Order) list in the original thesis by Carlos, the only substantial difference being the clockwise order.

From this list of insertion/deletion events, using the results from Brodal and Jacob [2, 5] the planar dynamic convex hull of all subsets can be maintained. The dynamic convex hull of a set X is the several convex hulls $CH(X)$ of the points as the insertions/deletions are applied sequentially in the given order. The notation used for the dynamic convex hull of a set X will be $DCH(X)$.

Recall, an event is a "birth" or "death" with an angle associated to it. Given two consecutive events e_i and e_{i+1} that correspond to a certain angle α and β, the dynamic convex hulls don't change in the range of directions $[\alpha, \beta]$. The state of the dynamic convex hulls are denoted by $DCH(B_1)_{[e_i,e_{i+1}]}$ and $DCH(R_2)_{[e_i,e_{i+1}]}$.

The interesting result of Brodal and Jacob [2, 5] is that this dynamic convex hull can be computed in $O(n \log n)$ time: the initial convex hull is computed in $O(n \log n)$, and updating it is performed in optimal $O(\log n)$ time. This optimal updating complexity is crucial to remain in the target $O(n \log n)$ time. As the event list has order $O(n)$ elements, so computing the convex hull after each event would result in $O(n^2 \log n)$ complexity.

Maintaining the $CH(B_1)$ and $CH(R_2)$ using this dynamic convex hull will immediately allow solving the problem. The only separator left to place is s_2, that must separate $DCH(B_1)_{[e_i,e_{i+1}]}$ and $DCH(R_2)_{[e_i,e_{i+1}]}$ for any i. Compute the directions in which the two dynamic convex hulls are separable. To obtain this, compute the supporting lines between the two convex hull in $O(\log n)$ time, yielding an interval of directions in which it is possible to separate them [9].

Because the $DCH(B_1)_{[e_i,e_{i+1}]}$ and $DCH(R_2)_{[e_i,e_{i+1}]}$ are guaranteed to exist only in the interval of directions $[\alpha, \beta]$, we must intersect the interval of all directions that $DCH(B_1)_{[e_i,e_{i+1}]}$ and $DCH(R_2)_{[e_i,e_{i+1}]}$ are separable with $[\alpha, \beta]$. It could happen that $DCH(B_1)_{[e_i,e_{i+1}]}$ and $DCH(R_2)_{[e_i,e_{i+1}]}$ were not separable in any direction, correctly meaning that the intersection with $[\alpha, \beta]$ would be null.

For each consecutive pair of events, it has yielded an interval in which B_1 and R_2 are separable. Because of how the separators s_1 and s_3 are defined by the caliper, it is guaranteed that R_1 and B_1 are separated. The same is true for R_2 and B_2. So the union of the separability interval of B_1 and R_2 for each pair of consecutive events yields the solution.

The following is a pseudocode implementation of the code for $k = 3$.

Algorithm 2 Separability *red/blue/red/blue* $k = 3$

Input: $R,\ B$
Output: *intervals*

$CH(R) \leftarrow ConvexHull(R)$
$CH(B) \leftarrow ConvexHull(B)$
$events \leftarrow [\,]$ ▷ Creation of the event list
for $r \in R$ **do**
    $(l_1, l_2) \leftarrow SupportingLines(r, CH(B))$
    $\alpha \leftarrow slope(l_1)$
    $event \leftarrow \{angle : \alpha,\ point : r,\ type : birth\}$
    $events.append(event)$
    $\beta \leftarrow slope(l_2)$
    $event \leftarrow \{angle : \beta,\ point : r,\ type : death\}$
    $events.append(event)$
end for
for $b \in B$ **do**
    $(l_1, l_2) \leftarrow SupportingLines(b, CH(R))$
    $\alpha \leftarrow slope(l_1)$
    $event \leftarrow \{angle : \alpha,\ point : b,\ type : birth\}$
    $events.append(event)$
    $\beta \leftarrow slope(l_2)$
    $event \leftarrow \{angle : \beta,\ point : b,\ type : death\}$
    $events.append(event)$
end for
$d \leftarrow 0$ ▷ The initial direction is arbitrary
$events \leftarrow SortByAngle(events, d)$ ▷ Sorting from initial direction
$(r_{left}, r_{right}) \leftarrow ExtremalPoints(CH(R), d)$
$(b_{left}, b_{right}) \leftarrow ExtremalPoints(CH(B), d)$
$s_1 \leftarrow LineFromPointSlope(b_{left}, d)$ ▷ Initial s_1 & s_3
$s_3 \leftarrow LineFromPointSlope(r_{right}, d)$ ▷ Used to initialize R_1 & B_2
$R_1 \leftarrow \{r \in R \mid r$ is to the left of $s_1\}$ ▷ Initalization of all subsets
$R_2 \leftarrow R \setminus R_1$
$B_2 \leftarrow \{b \in B \mid b$ is to the right of $s_3\}$
$B_1 \leftarrow B \setminus B_2$
$DCH(R_1) \leftarrow DynamicConvexHull(R_1)$
$DCH(B_1) \leftarrow DynamicConvexHull(B_1)$
$DCH(R_2) \leftarrow DynamicConvexHull(R_2)$
$DCH(B_2) \leftarrow DynamicConvexHull(B_2)$

$intervals \leftarrow \emptyset$
$presentEvent \leftarrow events[0]$
for $i = 1$ to $events.size()$ **do**
    $nextEvent \leftarrow events[i]$
    if $presentEvent.point \in R$ **then** ▷ Update dynamic convex hull
        $(DCH(R_1)_{[e_i,e_{i+1}]}, DCH(R_2)_{[e_i,e_{i+1}]}) \leftarrow UpdateDCH(presentEvent)$
    else if $presentEvent.point \in B$ **then**
        $(DCH(B_1)_{[e_i,e_{i+1}]}, DCH(B_2)_{[e_i,e_{i+1}]}) \leftarrow UpdateDCH(presentEvent)$
    end if ▷ Births add it to B_2 and remove it from B_1, viceversa for death
    $(l_1, l_2) \leftarrow SupportingLines(DCH(B_1), DCH(R_2))$
    $\alpha \leftarrow slope(l_1)$
    $\beta \leftarrow slope(l_2)$ ▷ Separability angle interval
    $\Lambda \leftarrow presentEvent.angle$
    $\Omega \leftarrow nextEvent.angle$ ▷ Consecutive events angle interval
    $intervals \leftarrow intervals \cup ([\Lambda, \Omega] \cap [\alpha, \beta])$
    $presentEvent \leftarrow nextEvent$
end for
return $intervals$

5.3.2 $k = 4$

Finally, we reach the main star of the original work. This is the last algorithm from the original work left to explain, and the one Section 6 builds upon. The approach will be similar to the one used for $k = 2$ in Section 5.2, as we are again studying an even value for k, but will incorporate the dynamic convex hulls used for $k = 3$ in Section 5.3.1.

As it is already customary, before the algorithm starts, it quickly checks what directions are separable using 3 or fewer lines, by executing Algorithm 2.

Once computed the angle intervals that at least need 4 lines to separate the sets, the original algorithm starts by constructing a caliper that rotates around $CH(B)$. This caliper represents the separators s_1 and s_4. See Figure 5. Again, it computes the support lines of the red points with respect to the $CH(B)$ and builds the list of birth and death events in $O(n \log n)$ time. The slopes of the support lines here belong again to the $[0, \pi]$ interval, using Equation (3).

With this event list, the caliper can be pivoted as it rotates over $CH(B)$, maintaining the sets R_1,R_2,R_3 with their respective dynamic convex hulls. The difference with the previous algorithms is that inside the caliper there are two blue sets, B_1 and B_2, and it must be determined if they are separable from R_2 using the two remaining separators s_2 and s_3. See Figure 5.

Before proceeding, it must be pointed out that a similar argument to Proposition 2 justifies the 2 separators already used. If either of the lines of the caliper over $CH(B)$, say s_1 and s_4, was not necessary to separate the sets in a given direction, then it would mean that one of the blue points that was supporting the caliper has no red points on the outer side of the caliper in that direction.

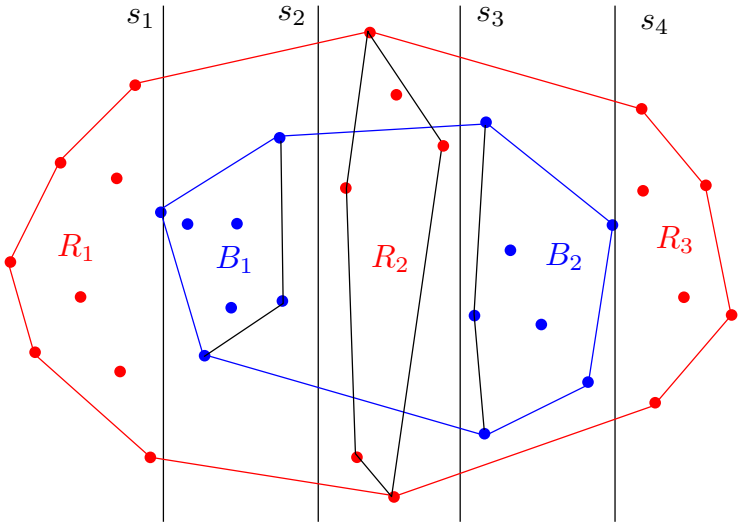

Figure 5: General separability by 4 lines.

This would render the separator indeed unnecessary, but it would mean that in this orientation you wouldn't have a *red/blue/red/blue/red* separation. You would need less than 4 lines. But this is a contradiction with the first step of the algorithm, that guarantees that at least 4 are needed, for the directions we are considering.

So far, we have just restated some steps used in previous algorithms, and the cost of all previous steps has already been discussed. To proceed with the separation of the blue points into B_1 and B_2, a very important lemma is used.

Lemma 2. *The caliper over $CH(B)$ is never empty of red points.*

Proof. By Proposition 2, R_1 and R_3 are never empty. By a similar demonstration R_2 is also never empty.

If for a given orientation R_2 was empty, one could separate the sets using only two lines to separate R_1 from B_1, and B_2 from R_3. This could be notated as $R_1/B_1 \cup B_2/R_3$, where $B_1 \cup B_2 = B$, meaning that less than 4 separators would be needed. Again, a contradiction with the first step of the algorithm. □

This guarantees that there is always some point in R_2 for any considered direction. Here the algorithm takes further advantage of the points inside the $CH(B)$, that by definition must always be inside the caliper and belong to R_2 for all directions. Pick one of them to be the *guard* with nomenclature $g = r \in R$ such that r lays inside $CH(B)$. It must be that the separators s_2 and s_3 at least separate this point g from B_1 and B_2 for any direction.

In a more general way: given an orientation d and a point in R_2 for that orientation, a line with that slope d (same orientation as the caliper separators) passing through the red point must classify the blue points. The ones left of this line will belong to B_1, the rest to B_2. This line can be understood as applying a $k = 1$ separability algorithm to the blue points with respect to the "guard". Or it can also be meant as an arbitrarily thin caliper rotating over the red point. The slopes of the lines here belong again to the $[0, \pi]$ interval, using Equation (3).

Because g is inside $CH(B)$, the classification is always possible if done with respect to g. See Figure 6.

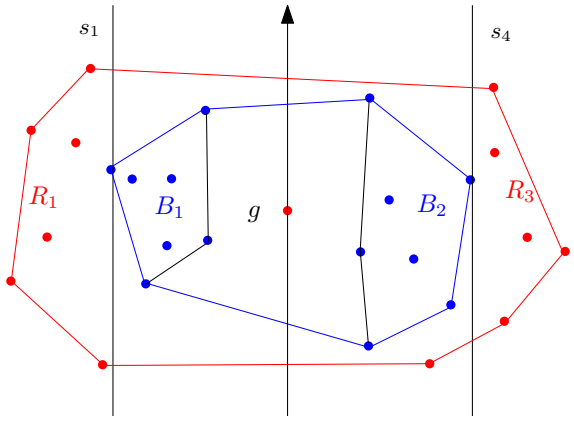

Figure 6: Separability by 4 lines with a guard.

The blue points will change from belonging to B_1 or B_2, as they change from laying to the left or right of the (oriented) line that goes through g with the same direction as the caliper. These will be referred to as jump events, with their associated angle given by the slope of the line. This slope is again inside the $[0, \pi]$ interval, using Equation (3).

Because the caliper $s_1 - s_4$ is always clockwise rotating around and supporting $CH(B)$ then, the (guard) red point g will be always inside the rotating caliper.

To compute the list of jump events, calculate the angle of the line that goes through the red guard and each blue point. After computing all the angles, sort them. Computing the lines, their angles and then sorting them takes $O(n \log n)$ time, yielding a list with $O(n)$ angles. See Figure 7. This list is referred to as the bi-partition list of a "guard" with respect to the points of opposite colour. Or its list of jump events.

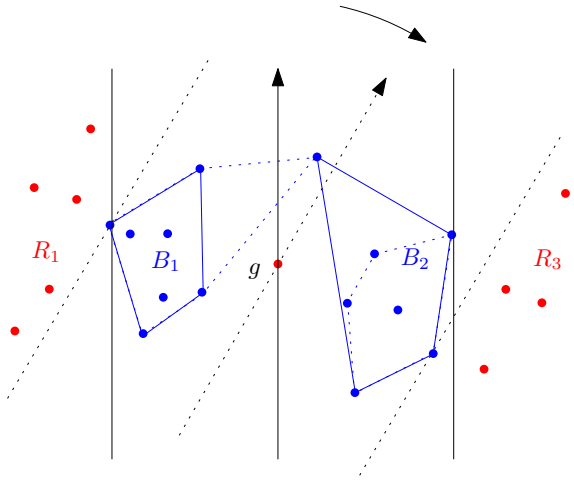

Figure 7: The guard g inside $CH(B)$ given the bipartition of the point set B.

Given this list, it is possible to perform the following query: Given an orientation d, what blue points lay to the right half plane that has orientation d and goes through g? This bi-partitions the set of blue points in B_1 and B_2. The computation of this query is linear the first time performed, but as the algorithm increases d by traversing from one event to the next, calculating the next B_1 and B_2 will be a constant time update.

Merging the events of the bi-partition list, with the birth and death events calculated previously, we can keep track of all the subsets $R_1/B_1/R_2/B_2/R_3$ as we rotate the caliper. The birth and death events tell us when a red point enters R_2 (the caliper) or when it exits, and to what subset (R_1 or R_3). And the jump events tell us if a point belongs to B_1 or B_2. See Figure 7.

Once the subsets can be updated dynamically as we rotate, again compute the dynamic convex hull [5].

During an event interval $[e_i, e_{i+1}]$ the dynamic convex hulls, DCH, don't change. Just as we did for $k = 3$, we must check the separability of adjacent subsets, that now are B_1 with R_2, and R_2 with B_2. Checking the linear separability between $DCH(R_2)_{[e_i,e_{i+1}]}$ and $DCH(B_1)_{[e_i,e_{i+1}]}$ and the linear separability between $DCH(R_2)_{[e_i,e_{i+1}]}$ and $DCH(B_2)_{[e_i,e_{i+1}]}$ for all consecutive events, will yield the solution.

Compute the supporting lines between the adjacent pairs of dynamic convex hulls (B_1/R_2, R_2/B_2). The slopes of the supporting lines define an angular interval where there exists a line that separates the two adjacent convex hulls.

If the hulls overlap, the interval is null. Intersect these intervals with $[e_i, e_{i+1}]$. Repeat for each consecutive pair of events, merge the results by calculating the union. Just as done in Algorithm 2. See Figure 8.

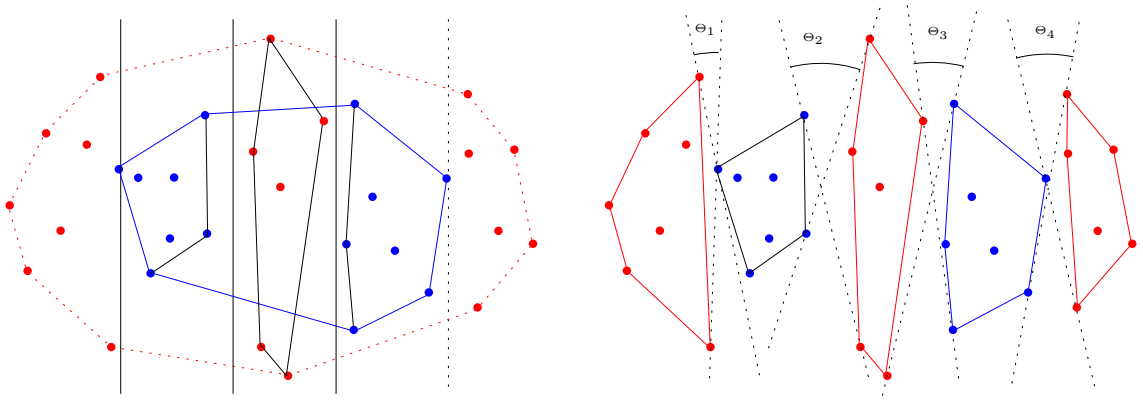

Figure 8: The Angles Θ_1, Θ_2, Θ_3, and Θ_4 defined by the *interior* supporting lines between consecutive monochromatic subsets.

The complexity of these dynamic convex hulls is again $O(n \log n)$ time, and updating them is $O(\log n)$ time. The cost of calculating supporting lines between them is, also again, $O(\log n)$ time. The events here consisted of all births and deaths, and all jumps of blue points between B_1 and B_2. This results in a linear number of events, so there is a linear number of updates to the dynamic convex hulls.

Meaning that the cost of updating the dynamic convex hull is $O(n \log n)$ in total, and the total cost of calculating supporting lines between convex hulls is $O(n \log n)$ time.

Algorithm 3 Separability *red/blue/red/blue/red*, $k = 4$

Input: $R,\ B$ (at least one red point is inside CH(B))
Output: *intervals*

$CH(R) \leftarrow ConvexHull(R)$
$CH(B) \leftarrow ConvexHull(B)$
$events \leftarrow [\,]$ ▷ Creation of the event list
for $r \in R$ **do**
    if r is inside $CH(B)$ **then**
        $g \leftarrow r$ ▷ The guard is the red point inside $CH(B)$
    end if
    $(l_1, l_2) \leftarrow SupportingLines(r, CH(B))$
    $\alpha \leftarrow slope(l_1)$
    $event \leftarrow \{angle : \alpha,\ point : r,\ type : birth\}$
    $events.append(event)$
    $\beta \leftarrow slope(l_2)$
    $event \leftarrow \{angle : \beta,\ point : r,\ type : death\}$
    $events.append(event)$
end for
for $b \in B$ **do**
    $l_r \leftarrow lineFrom2Points(b, g)$
    $\gamma \leftarrow slope(l_r)$ ▷ Jump event from B_1 to B_2
    $jumpEvent \leftarrow \{angle : \gamma, point : b, type : jump\}$
    $events.append(event)$
end for
$d \leftarrow 0$
$events \leftarrow SortByAngle(events, d)$
$(r_{left}, r_{right}) \leftarrow ExtremalPoints(CH(R), d)$
$s_1 \leftarrow LineFromPointSlope(b_{left}, d)$ ▷ Initial s_1 & s_4 caliper
$s_4 \leftarrow LineFromPointSlope(r_{right}, d)$
$R_1 \leftarrow \{r \in R \mid r$ is to the left of $s_1\}$ ▷ Initalization of all subsets
$R_3 \leftarrow \{r \in R \mid r$ is to the right of $s_4\}$
$R_2 \leftarrow R \setminus (R_1 \cup R_3)$
$l_{blue} \leftarrow LineFromPointSlope(g, d)$ ▷ Auxiliary line that partitions B_1 from B_2
$B_1 \leftarrow \{b \in B \mid b$ is to the left of $l_{blue}\}$
$B_2 \leftarrow B \setminus B_1$
$DCH(R_1) \leftarrow DynamicConvexHull(R_1)$
$DCH(B_1) \leftarrow DynamicConvexHull(B_1)$
$DCH(R_2) \leftarrow DynamicConvexHull(R_2)$
$DCH(B_2) \leftarrow DynamicConvexHull(B_2)$
$DCH(R_3) \leftarrow DynamicConvexHull(R_3)$

```
intervals ← ∅
presentEvent ← events[0]
for i = 1 to events.size() do
    nextEvent ← events[i]
    if presentEvent.point ∈ R then                    ▷ Update dynamic convex hull
        (DCH(R_1)_[e_i,e_{i+1}], DCH(R_2)_[e_i,e_{i+1}], DCH(R_3)_[e_i,e_{i+1}]) ← UpdateDCH(presentEvent)
    else if presentEvent.point ∈ B then
        (DCH(B_1)_[e_i,e_{i+1}], DCH(B_2)_[e_i,e_{i+1}]) ← UpdateDCH(presentEvent)
    end if                    ▷ Jump events move points from one set to the other
    (l_1, l_2) ← SupportingLines( DCH_{B_1}, DCH_{R_2})
    α ← slope(l_1)
    β ← slope(l_2)
    sepInterv_1 ← [α, β]                    ▷ Interval where B_1 is separable from R_2
    (l_3, l_4) ← SupportingLines( DCH_{R_2}, DCH_{B_2})
    γ ← slope(l_3)
    δ ← slope(l_4)
    sepInterv_2 ← [γ, δ]                    ▷ Interval where R_2 is separable from B_2
    Λ ← presentEvent.angle
    Ω ← nextEvent.angle                    ▷ Consecutive events angle interval
    intervals ← intervals ∪ ([Λ, Ω] ∩ sepInterv_1 ∩ sepInterv_2)
    presentEvent ← nextEvent
end for
return intervals
```

As all steps have an aggregated time complexity of $O(n \log n)$, that is the final complexity of the algorithm for $k = 4$. This concludes the overview of the original algorithms, and all the original work. All further sections are new contributions.

6 The new algorithm for $k = 4$

This section expands the original algorithm for $k = 4$, building on it. Here start the extensions of the original algorithms. All further sections describe new ideas to answer the open questions.

In order to solve configurations of points that lack any red point inside the $CH(B)$, the new algorithm makes use of several red points as guards, instead of only one. In the original algorithm we had a single guard called g, in this extension of the algorithm we will have a set of guards $G = \{g_1, g_2, \ldots, g_m\}$. From now on, assume that there are no red points (guards) inside $CH(B)$. If there was any, the configuration can be solved using the old algorithm as part of a pre-processing.

For each $g_i \in G$, compute the bi-partition list computed for g in the original algorithm. This entails computing the slopes of the lines going from each guard to each blue point. Each list of angles, that recall all belong to $[0, \pi]$, is then sorted in $O(n \log n)$ time. See Figure 9.

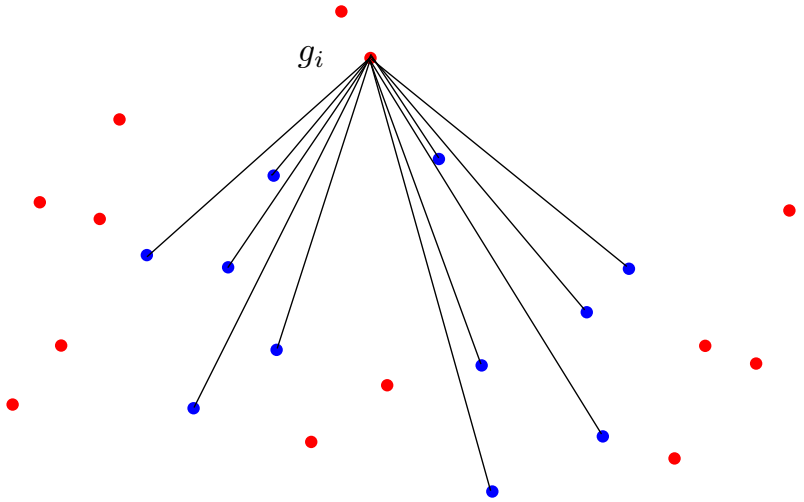

Figure 9: The guard g_i and the orientations of the blue points of B with respect to g_i, defining a vector of orientations to be sorted later.

Because we need a way of dividing the blue points into the two sets, we need at least one guard in G at any direction inside the caliper. This was guaranteed before because the guard was inside the $CH(B)$. As the caliper rotates, every red point is at least outside the caliper for some direction interval, the complement of its living angle interval. But it still is true by Lemma 2 that there is at least one red point inside the caliper for all orientations, so there has to exist the set G, $G \subseteq R$, such that for all orientations there is a guard inside the caliper. In other words, $G = R$ always yields a valid set of guards.

Convert G into the sequence $< g_1, g_2, \ldots, g_m >$. The order in which the guards are used to bi-partition the blue points, i.e., the order they become alive as the caliper rotates, is what determines the order of the sequence. If for a direction d more than one guard is alive (inside the caliper), consider only as "genuinely" alive the one with the smallest birth angle, that is the first to enter the caliper i.e., the first to appear in the sequence.

For each orientation, use the alive guard to partition the blue points into B_1 and B_2, as with the original guard in the old algorithm. And once this guard dies, another takes its place.

The variable m will denote the cardinality of guards ($|G| = m$). If the computational cost was $O(n \log n)$ for the original case with only one guard, now it will be $O(m \cdot n \log n)$. This is because we had to compute the bi-partition event list for each guard. This means that the number of guards must be minimized, and indeed try to make it into a constant factor. So $G = R$ might be a valid set of guards, but it is not an option due to the computational complexity it entails.

As the guards are red points, they have a birth and death event associated, forming the living angle interval. Before the genuinely alive guard dies, the next must already be alive. Let the living angle interval of a guard be referred to as the angle that is "guarded" or "covered" by that guard. See Figure 10. Those intervals must overlap in a cyclical manner, totally covering the $[0, \pi]$ interval.

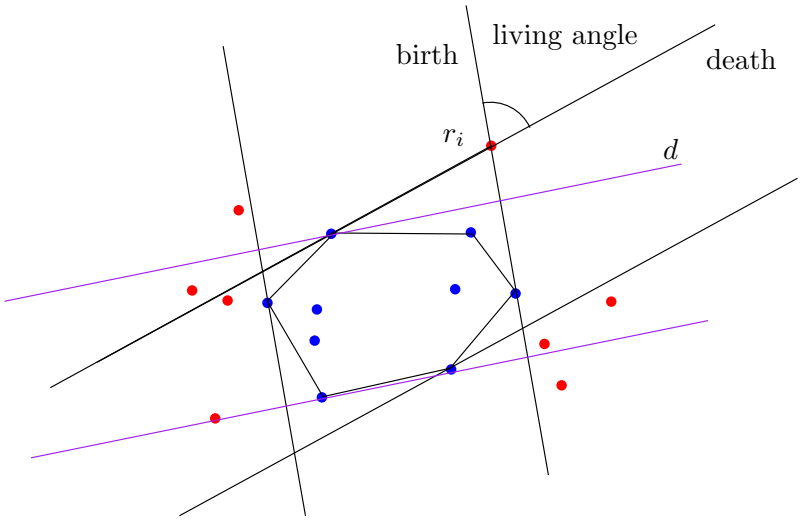

Figure 10: Example of each red point r_i being outside a caliper for some direction $d \in [0, \pi]$.

This lets us reformulate the problem of minimizing guards as minimizing the sets to cover the $[0, \pi]$ interval. See Figure 11.

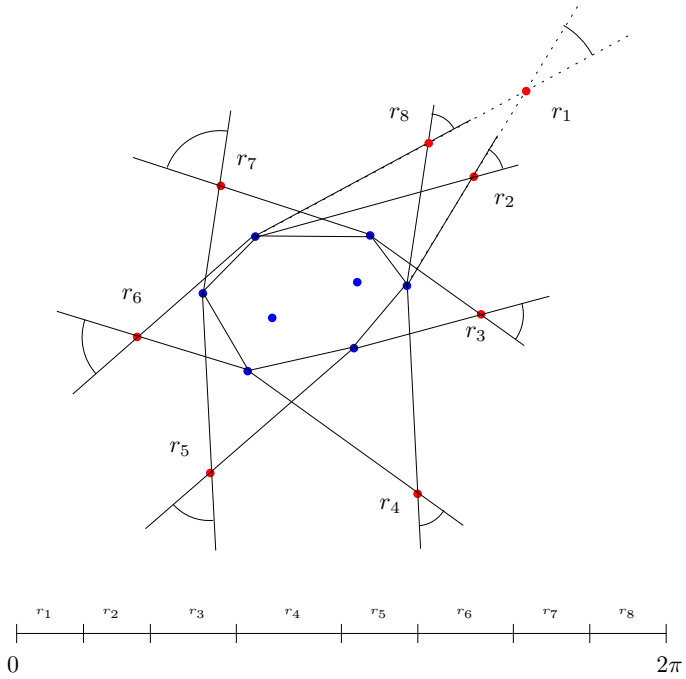

Figure 11: A set of 8 blue points, and a set of 8 red points which are the (minimum number of) guards covering *exactly* the $[0, \pi]$ interval.

6.1 Algorithm pseudocode

For this section, it will be assumed that the set G is provided as input. But in Section 8 it is discussed how to compute the set G given R and B. It's also where this notion of the minimizing of guards as minimizing of intervals to cover $[0, \pi]$ is developed.

Algorithm 4 Separability $red/blue/red/blue/red, \quad k = 4$ New version

Input: $R,\ G,\ B$
Output: $intervals$

$CH(R) \leftarrow ConvexHull(R)$
$CH(B) \leftarrow ConvexHull(B)$
$events \leftarrow [\,]$ ▷ Creation of the event list
for $r \in R$ **do**
    $(l_1, l_2) \leftarrow SupportingLines(r, CH(B))$
    $\alpha \leftarrow slope(l_1)$
    $event \leftarrow \{angle : \alpha,\ point : r,\ type : birth\}$
    $events.append(event)$
    $\beta \leftarrow slope(l_2)$
    $event \leftarrow \{angle : \beta,\ point : r,\ type : death\}$
    $events.append(event)$
end for
$sequence\_G \leftarrow sortByBirthAngle(G)$ ▷ From the set G to a list $sequence$
$(\_,\ prev\_g) \leftarrow SuportingLines(sequence.last, CH(B))$
$prev\_g \leftarrow slope(prev\_g)$ ▷ Death angle of previous guard in $sequence$
for $g \in sequence$ **do**
    $jumpAllBlue \leftarrow \{angle : prev\_g, points : B, type : jump\}$ ▷ All blue points
    $events.append(jumpAllBlue)$ ▷ When guard changes, all blue points jump
    for $b \in B$ **do**
        $l_r \leftarrow lineFrom2Points(b, g)$
        $\gamma \leftarrow max(slope(l_r), prev\_g)$
        $jumpEvent \leftarrow \{angle : \gamma, point : b, type : jump\}$
        $events.append(event)$ ▷ Jump event from B_1 to B_2
    end for
    $(\_,\ prev\_g) \leftarrow SuportingLines(g, CH(B))$
    $prev\_g \leftarrow slope(prev\_g)$
end for
$d \leftarrow 0$
$events \leftarrow SortByAngle(events, d)$
$(r_{left}, r_{right}) \leftarrow ExtremalPoints(CH(R), d)$
$s_1 \leftarrow LineFromPointSlope(b_{left}, d)$ ▷ Initial s_1 & s_4 caliper
$s_4 \leftarrow LineFromPointSlope(r_{right}, d)$
$R_1 \leftarrow \{r \in R \mid r$ is to the left of $s_1\}$ ▷ Initalization of all subsets
$R_3 \leftarrow \{r \in R \mid r$ is to the right of $s_4\}$
$R_2 \leftarrow R \setminus (R_1 \cup R_3)$

36: $l_{blue} \leftarrow LineFromPointSlope(sequence.first, d)$
37: $B_1 \leftarrow \{b \in B \mid b \text{ is to the left of } l_{blue}\}$
38: $B_2 \leftarrow B \setminus B_1$
39: $DCH(R_1) \leftarrow DynamicConvexHull(R_1)$
40: $DCH(B_1) \leftarrow DynamicConvexHull(B_1)$
41: $DCH(R_2) \leftarrow DynamicConvexHull(R_2)$
42: $DCH(B_2) \leftarrow DynamicConvexHull(B_2)$
43: $DCH(R_3) \leftarrow DynamicConvexHull(R_3)$
44: $intervals \leftarrow \emptyset$
45: $presentEvent \leftarrow events[0]$
46: **for** $i = 1$ to $events.size()$ **do**
47: $nextEvent \leftarrow events[i]$
48: **if** $presentEvent.point \in R$ **then** ▷ Update dynamic convex hull
49: $(DCH(R_1)_{[e_i,e_{i+1}]}, DCH(R_2)_{[e_i,e_{i+1}]}, DCH(R_3)_{[e_i,e_{i+1}]}) \leftarrow UpdateDCH(presentEvent)$
50: **else if** $presentEvent.point \in B$ **then**
51: $(DCH(B_1)_{[e_i,e_{i+1}]},\ DCH(B_2)_{[e_i,e_{i+1}]}) \leftarrow UpdateDCH(presentEvent)$
52: **end if** ▷ Jump events move points from one set to the other
53: $(l_1, l_2) \leftarrow SupportingLines(\ DCH_{B_1},\ DCH_{R_2})$
54: $\alpha \leftarrow slope(l_1)$
55: $\beta \leftarrow slope(l_2)$
56: $sepInterv_1 \leftarrow [\alpha, \beta]$ ▷ Interval where B_1 is separable from R_2
57: $(l_3, l_4) \leftarrow SupportingLines(\ DCH_{R_2},\ DCH_{B_2})$
58: $\gamma \leftarrow slope(l_3)$
59: $\delta \leftarrow slope(l_4)$
60: $sepInterv_2 \leftarrow [\gamma, \delta]$ ▷ Interval where R_2 is separable from B_2
61: $\Lambda \leftarrow presentEvent.angle$
62: $\Omega \leftarrow nextEvent.angle$ ▷ Consecutive events angle interval
63: $intervals \leftarrow intervals \cup ([\Lambda, \Omega] \cap sepInterv_1 \cap sepInterv_2)$
64: $presentEvent \leftarrow nextEvent$
65: **end for**
66: **return** $intervals$

The changes to the pseudocode has been the introduction of the new input G, that is traversed in a for loop at line 16. This traversal is done over the variable *sequence*, that represents the $< g_1, g_2, \ldots, g_m >$ mentioned before. A new event *jumpAllBlue* is used to swap all blue points from B_2 to B_1 when one guard dies and the next takes the role of active guard. This bookkeeping with the death of guards is aided by the variable $prev\_g$ that stores the death angle of the last guard. All events of the current guard must happen at a bigger/later angle than this $prev\_g$.

6.2 Linear size counterexamples

As a final remark of this extension, in Figure 11 we illustrate the worst case scenario where m is linear with n, concretely, $m = n$, i.e., each red point is a guard.

As we can see, for some particular configurations of points so that the number of guards is linear ($O(m) = O(n)$). Meaning that the cost would grow to $O(nn \log n) = O(n^2 \log n)$. See again Figure 11. This was the same cost as the algorithm that swept the dual. Meaning that unless we can find a better upper bound for the number of guards, the new algorithm will not be an improvement.

7 Sufficient conditions for a constant size G with $k = 4$

As shown in the previous section, Algorithm 4 has the set G as input, and it has been shown that its size m could be linear with respect to n. This begs the question of finding conditions over R and B that guarantee that m is constant with respect to n. A very similar approach to the one used in the original algorithm.

Other than the configurations that have a red point inside the $CH(B)$ (i.e, the ones treated in the original algorithm), thanks to the new version of the algorithm there is a second family of configurations with a constant number of guards. We will explore it by introducing first a special case of this family.

7.1 Triangle of guards

Condition 1. *There are* 3 *red guards, denoted by* A, B *and* C, *such that all the sides of the triangle* $\widehat{ABC}$ *cross the* $CH(B)$.

Proposition 3. *If* A, B, C *satisfies Condition 1, then* $G = \{A, B, C\}$ *is a guard set of size* 3 *that covers the entire rotation of the caliper.*

Proof. Let's study into what set R_1, R_2, or R_3 these red points $\{A, B, C\}$ would fall into. Assume that there exists a given orientation such that none of these 3 points is inside R_2. Then they must be in R_1 and/or R_3. Without loss of generality, assume that point A is inside R_1. Then it is impossible for points B or C to also be in R_1, because the segment joining them wouldn't cross $CH(B)$. See Figure 12.

So B and C can't be in R_1, and it is assumed that R_2 is empty, so they must both be in R_3. But this is a contradiction, because if both are in the same set, the straight segment $\overline{\text{CB}}$ can't cross the $CH(B)$. See Figure 12. A contradiction, so the assumption was false.

Thus, R_2 can't be empty for any direction with respect to $\{A, B, C\}$, so the set of guards defined by them is valid and has the stated size of 3. □

This means that all configurations that satisfy the Condition 1 always have a set $G = \{A, B, C\}$ such that the whole rotation of the caliper can be guarded. And most importantly that the size of this G is always 3, in other words it is constant. This yields a $O(n \log n)$ time complexity.

This is a first condition over R and B that makes them treatable by the Algorithm 4. In the next section, we relax it more to allow more configurations of points to be treated by the algorithm.

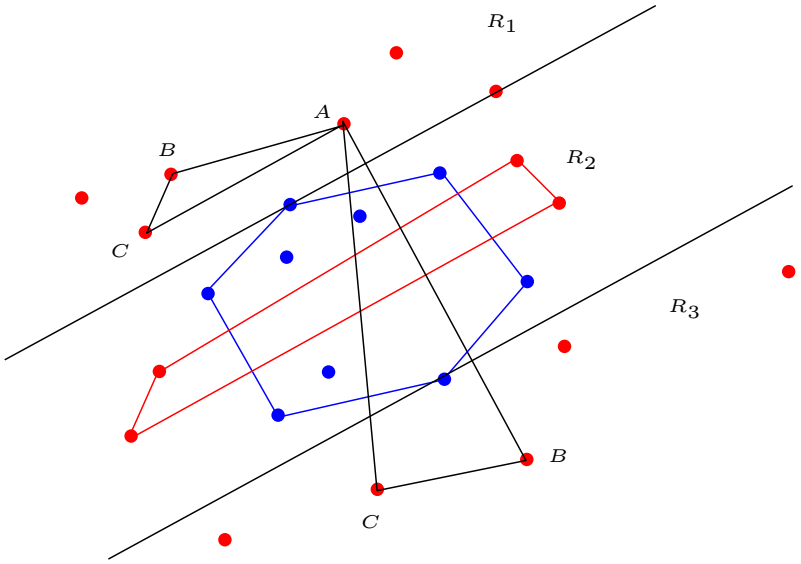

Figure 12: Visual proof that it is impossible for B or C to be in R_1, and analogously in R_3 because the segment joining them wouldn't cross $CH(B)$.

7.2 Star-shaped guards

The specific property of this guard set has a close relationship with the triangle it formed. So, we extend this geometrical analysis to other guard sets. Recall that a sequence of guards was generated from G by sorting the guards according to birth angle. This sequence of guards will be understood to trace a polygonal line, closed by adding an edge from the last to the first guard.

Condition 2. *There is a guard sequence* $< g_1, g_2, \ldots, g_m >$ *such that all the edges of the closed polygonal line traced by the sequence of guards cross the* $CH(B)$*, and* m *is odd.*

Proposition 4. *If* $< g_1, g_2, \ldots, g_m >$ *satisfies Condition 2, then* $G = \{g_1, g_2, \ldots, g_m\}$ *is a size* m *guard set that covers the entire rotation of the caliper.*

Proof. As before, we prove by contradiction that R_2 is not empty for any direction. So assume R_2 is empty, and without loss of generality, g_1 belongs to R_1.

As in Proposition 3, g_2 can't be in R_1 or R_2, so by pigeonhole principle it must be in R_3. Repeat for g_3, that can't be in either R_2 or R_3, so it again belongs to R_1. Alternate until only the last guard remains to be assigned.

Because of Condition 2 there are an odd number of guards, and we started assigning guards to R_1. Therefore, g_{m-1} must belong to R_3. So the last guard g_m can't be in R_3, because its predecessor belong to it. But it can neither be in R_1, because the first guard is there, and the edge from the last guard back to the first one wouldn't intersect the $CH(B)$. See Figure 13. This means that the last guard has to be in R_2, completing the proof by contradiction. □

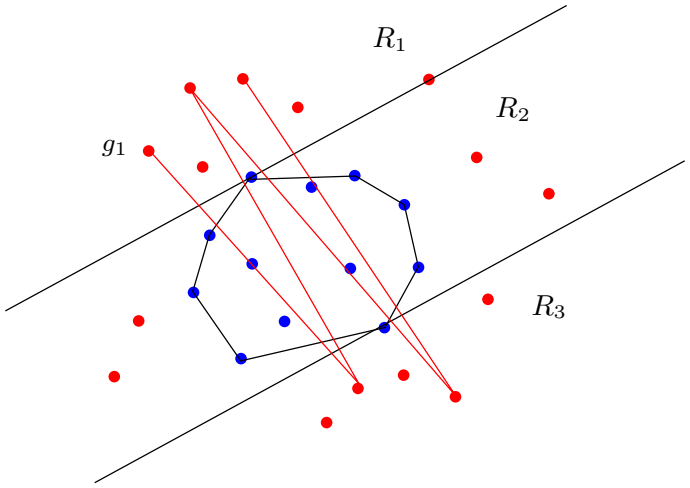

Figure 13: Visual proof that two guards fall in the same subset, the edge doesn't intersect the $CH(B)$

This extended family of closed polygonal lines can be considered for a constant value of m. If $m = 3$ yielded the triangle, what do higher odd values of m yield? A priori, a very large and diverse variety of shapes.

The polygons that result of Condition 2 include from convex to self intersecting. It also includes the particular family of star polygons, with special interest in the ones with maximum density. Such as for $m = 5$ the pentagram $\{5/2\}$, for $m = 7$ the heptagram $\{7/3\}$, ..., in general $\{m/\lfloor m/2 \rfloor\}$. See Figure 14.

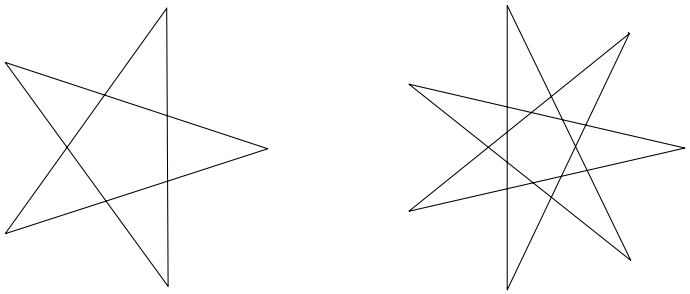

Figure 14: Left: The pentagram $\{5/2\}$. Right: The heptagram $\{7/3\}$.

Those specially starred polygons with high turning number are a very good example of how configurations that might have a very small (diameter) $CH(B)$, or equivalently, whose red points are very distant to the $CH(B)$, are still tractable if an adequately large but constant value for m is chosen. The problem remains that as these configurations could have smaller and smaller (diameter) $CH(B)$, the variable m would need to grow to allow them to be treated by the algorithm. Eventually m would become linear with respect to n.

This allows for a refinement of Proposition 4 to use this more appealing family of star polygons.

Proposition 5. *If* $< g_1, g_2, \ldots, g_m >$ *traces an odd star polygon, whose segments all intersect* $CH(B)$*, then* $G = \{g_1, g_2, \ldots, g_m\}$ *is a size* m *guard set that covers the entire rotation of the caliper.*

As these star polygons satisfy Condition 2, the demonstration is the same.

The beautiful family of star polygons shows that the new additions made in Section 6 do indeed allow for solving much more general configurations. Although it must be stressed, that Proposition 4 is not biconditional. In other words, Condition 2 is a sufficient but not necessary condition for an input R and B to have a constant size set of guards. See Figure 15.

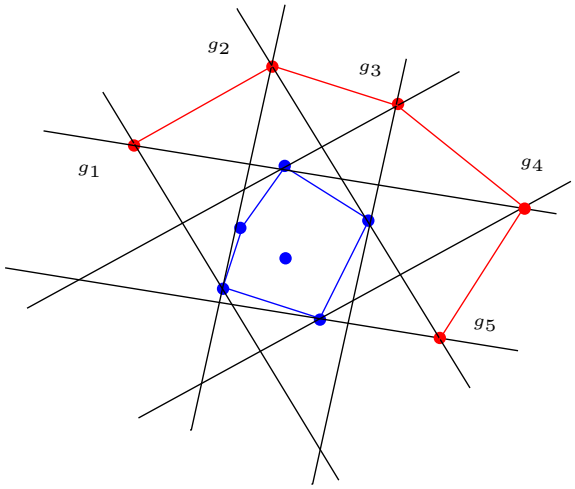

Figure 15: Sequence of guards covering the entire rotation, but not tracing a star polygon.

Further study is needed to find even more relaxed conditions that ensure a constant number of guards, or to prove that no such conditions exist. In the second case, a new better (faster, stronger) version of the algorithm would need to be designed. It would also be interesting in further work to explore more in depth the family of polygons that satisfy Condition 2.

7.3 Further work with guard polygons

Because of the interest arisen in studying the shapes defined by guards, it has been useful to define a new relation of equivalence over the guards. Recall that the use we make of guards is directly tied to the angle interval they cover, in turn defined by their living angle, i.e., their supporting lines with respect to $CH(B)$. If two guards had the same living angle, they would receive identical use by the algorithm. So the equivalence relation among guards follows:

Proposition 6. *Two guards are said to be equivalent if the interval they cover*[5] *is identical, i.e., the support lines of both guards are pairwise*[6] *parallel. This defines an equivalence relation.*

Proof. The proposition describes an equivalence relation denoted by $\equiv$, because it is a binary relation that satisfies reflexivity, symmetry, and transitivity:

- $\forall g, \quad g \equiv g$: because the support lines of a guard are parallel to themselves.
- $\forall g_1, g_2, \quad g_1 \equiv g_2 \Leftrightarrow g_2 \equiv g_1$: because two pairs of parallel lines are pairwise parallel even under commutation. As long as the comparison of the supporting lines is done, birth line against birth line, and death line against death line.
- $\forall g_1, g_2, g_3, \quad g_1 \equiv g_2 \wedge g_2 \equiv g_3 \Rightarrow g_1 \equiv g_3$: because all parallel lines to a given line (e.g., g_2 birth) are parallel amongst themselves.

Recall also the use made of Equation (3) when comparing the covered interval of guards. Meaning that guards in the half turn $[0, \pi]$ have equivalent guards in the remaining turn $[\pi, 2\pi]$, as defined in the equation. □

For each red point exterior to $CH(B)$, there is precisely only one other point that satisfies this equivalence. The construction of the equivalent point can be described from the definition made. First, find the support lines of the original point. Then trace the two parallel lines to both support lines, but that are tangent to $CH(B)$ on the antipodal points respectively. The intersection of these two parallel support lines yields only one equivalent guard. See Figure 16.

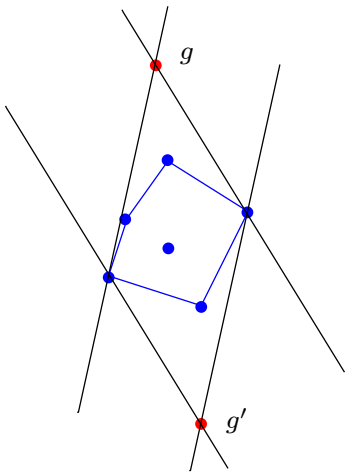

Figure 16: A pair of guards g and g' that are equivalent.

The use for this equivalence relation is that it adequately reflects the indifference our algorithm has when considering points. Both points are indistinguishable from the standpoint of their living angle interval. So it is now possible to swap a guard with its equivalent one, without it affecting the execution of the algorithm.

[5] Their living interval, i.e., the interval in which they lay inside the caliper.
[6] Birth with birth, death with death.

Because it lays outside the scope of this project, and not enough time is available to accommodate such task, it remains as future work to formalize a demonstration for the following open question:

Open problem 3. *For any $G =< g_1, g_2, \ldots, g_m >$ that satisfies Condition 2, there exists a set G' whose guards have an equivalent in G, and its sequence traces a star polygon.*

This G' would be a pruned version of G that had some guards inverted to lay them in a star shape. Some assumptions are that this G' would have an odd number of vertices, although this can be accommodated with some relaxation on the definition of star polygons. In fact, there are similar conditions to Condition 2 defined for even number of guards.

Another interesting open question is finding equivalent reformulations of Condition 2. It is easy to see that if we restrict R and B to be close together, then the proposition is true. Just like shown in Section 6.2, if R and B where not so sparse, then a constant size G could be found. One can formalize this in a lot of interesting drawing conditions, that end up ensuring a constant m. The following is just an example:

Open problem 4. *Assume that R and B lay inside a circle of fixed radius x, and that the $CH(B)$ is bigger than/contains a circle of radius y. Does an upper bound for the number of guards m exist in terms of x and y?*

These conditions could be very useful in real world applications, as in most cases where real data is used for R and B such bounds do normally exist.

8 Finding the constant size G for the new $k = 4$ algorithm

As pointed out before, Algorithm 4 has as input a set G of guards. And because the size of this set determines the computational complexity of the algorithm, it is a central question to find a constant size G.

It is such a central concern that the entirety of the above Section 7 is dedicated to show that those solutions even exist.
A brief run-through of all pertinent properties about the possible sets G:

- There is at least always a valid set $G = R$ that correctly covers the entire rotation, otherwise R and B aren't separable using exactly $k = 4$ lines [7].
- Smaller sets that still cover the entire rotation exist for the most common configurations of points (see example in Section 7).
- For some specially sparse instances, no smaller set exists (see counter-example in Section 6.2).

[7] The non-separable directions can be ignored, and the algorithm executed for the remaining directions.

With this in focus, the goal to reach is:

> *Given a set R and a set B, find a set G of guards that covers the entire rotation, and has minimal size.*

This allows to reframe the finding of guards as a minimization problem. To further reinforce this strategy, different visualizations are proposed.

8.1 Alternative visualizations

The guards can be understood not only as points, but as the interval of the caliper rotation they cover. As shown in Proposition 6, the algorithm only cares about the interval of direction a guard covers. This suggests representing the guards as intervals in the unit circle of directions. Each guard being their covering interval. See Figure 17.

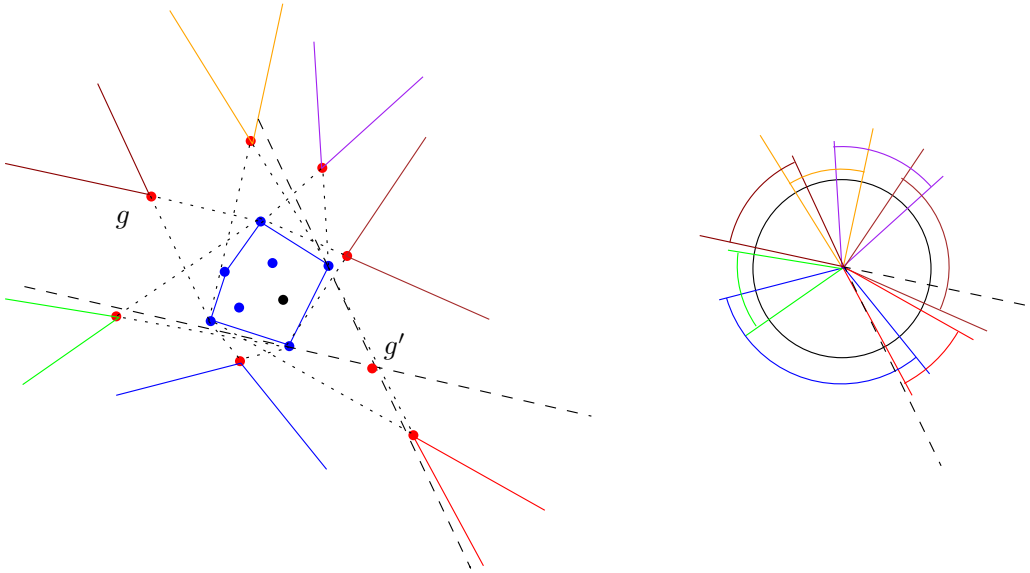

Figure 17: Left: Red guards for the sets R and B. Right: Angular intervals of the red guards on the unit circle. Note that their union dosent cover the interval $[0, 2\pi]$, but using the equivalent guard g' instead of guard g they cover an $[0, \pi]$ interval.

This yields a visualization that makes it easy to draw parallelisms with similar minimization interval cover problems. One could easily imagine that instead of the set of cyclical directions, we just had an interval in $\mathbb{R}$ called $[a, b]$. And the equivalent problem would be to find, amongst a set of intervals, the minimum amount such that the union covers the entirety of $[a, b]$. This is a well known problem [8] that can be solved via a greedy algorithm in $O(n \log n)$ time with respect to the number of intervals in the set. This indeed inspires the proposed solution to find a minimal G.

A second visualization, to further abstract the problem, is to generate the intersection graph from these intervals [6].

The intersection graph has as vertices the intervals, one per guard. Two vertices will have an edge connecting them if the intersection of the corresponding intervals is not empty. This graph then represents pairs of guards that are both alive for some direction, meaning that they are possible consecutive guards in some sequence $< g_1, g_2, \ldots, g_m >$. See Figure 18.

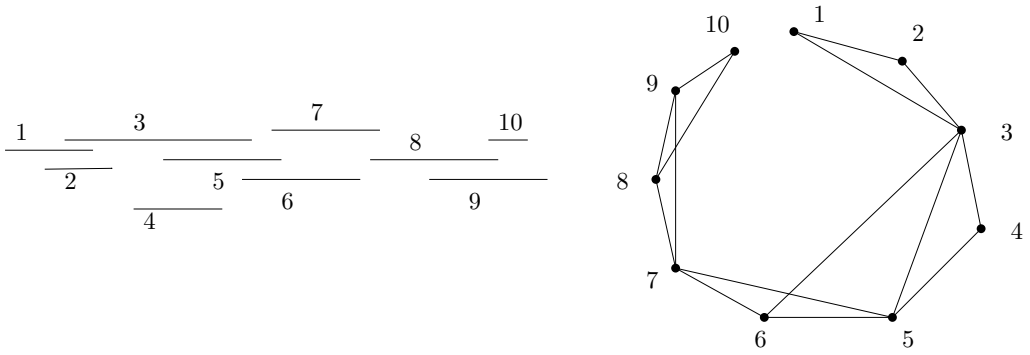

Figure 18: Intersection graph of angular intervals.

Refining further this graph, as the rotation of the caliper was set to be clockwise, direction is given to the edges of the graph. The direction of edges in the intersection graph will satisfy that the origin interval dies before than the destination interval. In this way, the direction of the edge captures how the sequence $< g_1, g_2, \ldots, g_m >$ jumps from one guard to the next, always in clockwise order. See Figure 19.

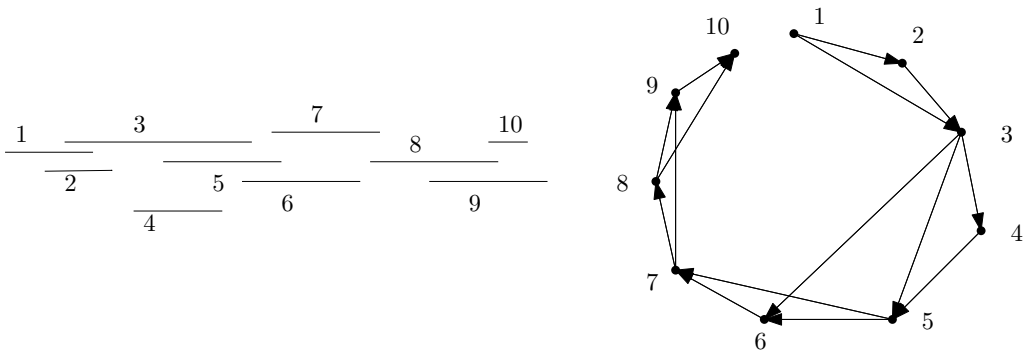

Figure 19: Directed intersection graph of angular intervals.

The computational cost of building this graph is $O(n^2)$ with respect to the vertices, as all intervals could have intersection (complete graph), and because the number of vertices is equal to the number of red points.

Both representations lead to uncovering interesting properties about G and the corresponding $< g_1, g_2, \ldots, g_m >$.

8.2 Domination of guards

The first important property uncovered by the interval representation is domination amongst guards.

Proposition 7. *A guard g_i dominates g_j if all directions covered by g_j are also covered by g_i.*

This is a very visual concept in the interval visualization. If an interval contains another one, then it dominates it. See Figure 20.

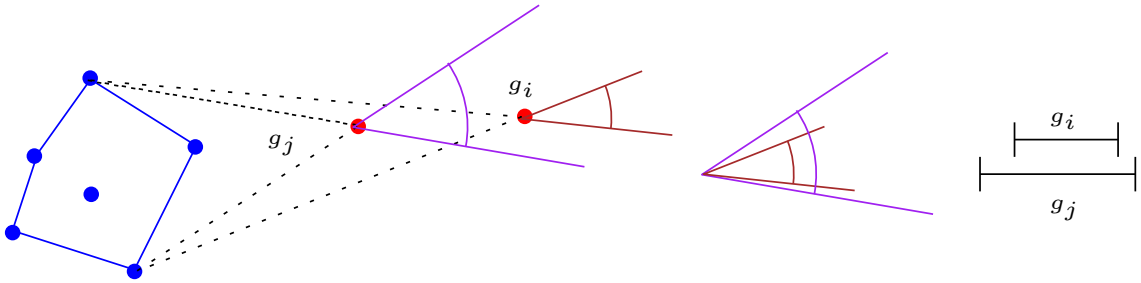

Figure 20: Domination of intervals of guards: g_i dominates g_j.

Tracing back this property to the original visualization of guards as points, an alternative easy way of understanding domination arises. If we draw their support lines of g_i and g_j with respect to $CH(B)$, that remember are what define the covering interval angles, then the property is clear to see. See again Figure 20 Center and Right.

The region contained by the supporting lines of a g_i[8] and sides of the $CH(B)$ is where guards like g_j are located. See Figure 20 Left. If a guard g_j is inside this region, its birth support line has less angle, and the death support line has more angle. In other words, g_j must have an "earlier" birth than g_i, and a "later" death. This means that g_j will cover all the directions covered by g_i, so g_j will dominate g_i.

8.2.1 Pruning algorithm pseudocode

This domination property allows for an immediate pruning of potential guards for the set G. Given all the points in R, only the ones that are not dominated by other red points are considered as candidates[9]. This can easily be done in $O(n \log n)$ time, instead of $O(n^2)$ that would take comparing all potential pairings of guards.

Sort and traverse birth and death of the intervals in increasing angle. This sorting has $O(n \log n)$ time complexity. As birth events are processed, keep an ordered list of the intervals that are being traversed, and drop them as their death events are processed. This list is called "*OpenIntervals*". When processing the death event of an interval named g_i, all intervals that were birthed before g_i birth, will also have a later death than g_i. So g_i is dominated by all the intervals that precede it in the list.

This check can be reformulated as: "When an interval dies, if it is not the first element of the *OpenIntervals* list, then it is dominated by at least one other interval".

[8] Or its inverse, with the corresponding parallel support lines. See Figure 16.
[9] If a guard was inside $CH(B)$, it would dominate all other guards.

So *OpenIntervals* must allow for: appending always at the last position, checking if an element is the first of the list, deletions of arbitrary elements. Recall that the list is sorted by the interval's birth, not by its identifier, and as deleting arbitrary elements requires finding the element, a data structure more complex than a linked-list is required.

A natural choice is to build a BST (balanced search tree) over the linked-list. This allows for finding an interval in the list given its identifier in $O(\log n)$ time. A very common implementation for BST are AVL-trees.

With this two data structures in place, we can support the three operations mentioned:

- Check if an interval identifier corresponds to the first element of the linked-list. With constant time complexity.

- Appending an element to the end of *OpenIntervals*. Now with $O(\log n)$ time complexity as we need to update the balanced tree.

- Deleting an element from *OpenIntervals*. With $O(\log n)$ time complexity, as we need to first find it in $O(\log n)$ time, delete it, and finally update (re-balance) the BST also in $O(\log n)$ time.

All this results in the algorithm having a $O(n \log n)$ time complexity, and $O(n)$ space complexity. The output R_{pruned} still has $O(n)$ size.

Algorithm 5 Domination pruning

Input: R, B
Output: R_{pruned}

$CH(B) \leftarrow ConvexHull(B)$
$CovrIntr \leftarrow coverIntervals(R, CH(B))$ ▷ Computing supporting lines
$TruncatedIntervals \leftarrow \emptyset$ ▷ Intervals that contain the direction 0 or π
for $interval \in CovrIntr$ **do**
    if $interval.birthAngle > interval.deathAngle$ **then**
        $TruncatedIntervals \leftarrow TruncatedIntervals \cup interval$
        $CovrIntr \leftarrow CovrIntr \setminus interval$
    end if
end for
for $interval \in TruncatedIntervals$ **do**
    $CovrIntr \leftarrow CovrIntr \cup [interval.birthAngle - \pi, interval.deathAngle]$
end for
for $interval \in TruncatedIntervals$ **do**
    $CovrIntr \leftarrow CovrIntr \cup [interval.birthAngle, interval.deathAngle + \pi]$
end for
▷ Now all intervals have smaller birthAngle than deathAngle

17: $CovrIntr \leftarrow SortByBithAngle(CovrIntr)$ ▷ From set to event list
18: $OpenIntervals \leftarrow [\,]$ ▷ Empty list
19: **for** $event \in CovrIntr$ **do**
20: **if** $event$ is $birth$ **then**
21: $OpenIntervals.append(event.point)$
22: **else if** $event$ is $death$ **then**
23: **if** $OpenIntervals.isNOTFirstElement(event.point)$ **then**
24: $R_{pruned} \leftarrow R_{pruned} \setminus event.point$ ▷ The point is dominated
25: **end if**
26: $OpenIntervals.delete(event.point)$
27: **end if**
28: **end for**
29: **return** R_{pruned} ▷ Non dominated guards

The only detail to mention on the pseudocode is the $coverIntervals(R, CH(B))$ function in line 2, and the bookkeeping must be done with the truncated intervals.

The function $coverIntervals(R, CH(B))$ is just a warp over the previously used function $SupportingLines(r, CH(B))$. Instead of returning the supporting lines of a single point, it computes them for all red points belonging to the input argument, that is the set R. This has a $O(n \log n)$ time complexity. And then it returns them in interval format, with a birthAngle and deathAngle attribute, and a point attribute with the identifier of the point the interval represents. In summary, $coverIntervals(R, CH(B))$ has the expected behaviour.

The bookkeeping consists in simply "extending" the truncated intervals. This is accomplished by finding the equivalent angle with respect to Equation (3). The extension must be done in both directions, as this truncated sets could dominate intervals that start and end very close to the 0 angle, i.e., in an interval $[\epsilon, \epsilon']$, or to the π angle, i.e., in an interval $[\pi - \epsilon', \pi - \epsilon]$. These extensions correspond to the for loops of declared in lines 10 and 13.

8.3 Greedy heuristic

Now that dominated red points have been pruned, let's proceed to find the smallest size set of guards. This optimal guard set G^{opt} will generate an also optimal sequence, denoted by $< g_1, g_2, \ldots, g_m >^{opt}$. A local optimality can be found among consecutive guards of the sequence.

Given a guard that belongs to the optimal sequence, let it be named g_i^{opt}, there are several guards that cover the angle (direction) of g_i^{opt} death. These guards remain inside the caliper once g_i^{opt} it leaves it, so one of them must be the next element of the optimal sequence. This set of guards that could succeed an g_i^{opt} are the candidates of g_i^{opt}. The candidates of any g_i are denoted as $candidates(g_i)$, being a set of guards.

These candidates can be understood more easily as the adjacent vertices of g_i in the intersection graph. This visualization is very relevant.

Remember that the edges are directed to ensure that they always advance clockwise when jumping from g_i to any of its candidates. See Figure 19.

From these candidates, a greedy heuristic would be to choose the one that dies the last, i.e., with the largest death angle[10]. The candidate selected by this heuristic will be alive at least for all directions covered by any guard in $candidates(g_i^{opt})$, clockwise from g_i^{opt} death. The candidate chosen by the greedy heuristic is labelled g_{i+1}^{greedy}. The corresponding representation in the intersection graph is to select one edge from the outgoing ones.

Lemma 3. *For any* g_i^{opt} *then,* $g_{i+1}^{greedy} = g_{i+1}^{opt}$.

Proof. A proof by contradiction demonstrates that this greedy heuristic is globally optimal. Given g_i^{opt}, assume that $g_{i+1}^{opt} \neq g_{i+1}^{greedy}$. The way g_{i+1}^{greedy} is chosen means that its death angle is greater than g_{i+1}^{opt}. (It can't be the same, as we consider it a degenerate case. If we assume the points in R and B are in general position, this never happens.) So both g_{i+1}^{opt} and g_{i+1}^{greedy} must die at a greater angle than the birth of g_{i+2}^{opt}. This means that even if $g_{i+1}^{opt} \neq g_{i+1}^{greedy}$, the greedy choice can "reach/see" g_{i+2}^{opt}, or an even better candidate.

In terms of the intersection graph, g_i^{opt} is adjacent (has as candidates) g_{i+1}^{greedy} and g_{i+1}^{opt}. Then as g_{i+1}^{opt} is adjacent to g_{i+2}^{opt}, the greedy heuristic ensures that g_{i+1}^{greedy} is adjacent to g_{i+2}^{opt}.

Because this holds for any consecutive guards, there is no point in the sequence $< g_1, g_2, \ldots, g_m >^{opt}$ were not choosing g_{i+1}^{greedy} as the next guard yields an advantage further down the algorithm. In other words, all the decisions made by a greedy algorithm based on the heuristic exposed are at least as optimal as decisions that were globally optimal. □

This is again the same greedy heuristic one would use to solve the classical interval cover problem. The only assumption left is that an initial g_i^{opt} is given.

8.3.1 Revisiting the intersection graph

To work around the fact the heuristic needs to start from a g_i^{opt}, several techniques can be used. First, some observations on how the greedy heuristic changes the intersection graph.

Given this heuristic, instead of representing all the outgoing edges for each g_i in the graph, just draw the ones chosen by the greedy heuristic. Omitting degenerate cases, all vertices now have at most one outgoing degree, i.e., $\forall i,\ deg_{out}(g_i) \leq 1$. If we assume that R and B are separable using $k = 4$ lines, the whole rotation must be covered, so there must always exist at least one candidate for any g_i. This results in a stricter $deg_{out}(g_i) = 1$. This subgraph will be referred to as the greedy intersection graph, noted as $GIG = (V, E)$, where the vertices $V = R_{pruned}$.

Some important complexity differences between the GIG and the intersection graph, is that the GIG can be commutated in $O(n \log n)$ time from the intervals. This was not the case with the intersection graph.

[10]Here we are skipping some problems that arise from the cyclical nature of the angles in the interval. This is solved by the same bookkeeping as in Algorithm 5

Proposition 8. *If R and B are separable using $k = 4$ lines, all the vertices of the greedy intersection graph have $deg_{out}(g_i) = 1$. So, the number of edges of GIG is $\sum_i^n deg_{out}(g_i) = n$.*

With this bound on the number of edges, the kind of graph that GIG can be deduced. First, GIG can't be acyclic, as no valid $< g_1, g_2, \ldots, g_m >$ would exist, meaning that it is not possible to cover the entire rotation. So GIG has at least a directed cycle of length 2 (we omit the case with a guard inside $CH(B)$), the cycle corresponds to $< g_1, g_2, \ldots, g_m >^{opt}$. This is corroborated by the fact that the number of vertices and edges also forces the existence of at least one cycle[11]. Other cycles might also exist. And because of how the edges were directed, any path starting from any vertex eventually ends up in one of those cycles.

Each connected component (or Cc.) of the GIG must-have one and only one cycle, that is located at the end of all paths starting from a vertex in the connected component[12]. A simple proof by contradiction suffices. If a Cc. were to have 2 cycles, a path from one to the other must exist, otherwise they are not in the same Cc. But that means that one vertex of the origin cycle has 2 outgoing edges, one that belongs to its cycle, and one that leads down the path to the second cycle. This is a contradiction with Proposition 8. Because GIG has at least one cycle and n edges, GIG has as many Cc. as cycles.[13].

Now that the kind of graph that is GIG has been uncovered, a way of finding $< g_1, g_2 \ldots g_m >^{opt}$ can be described in terms of executing search algorithms over GIG. Here we can see how multiple algorithms could be used, from topological sorts to other graph search algorithms studied in the undergraduate courses [13].

Returning to the algorithm, if before it assumed that a g_i^{opt} was given, it can instead start from all vertices, ensuring that it must at some point start from a g_i^{opt}. So for all vertices in GIG, follow the directed edges until detecting a cycle. Then choose the cycle of shorter length, that must be $< g_1, g_2 \ldots g_m >^{opt}$.

Because the length of a path starting from a g_i (optimal or not) is $O(n)$, if the algorithm starts from all vertices the computational complexity would jump to $O(n^2)$. This would be obviously undesirable. Fortunately, it is easy to find a $O(n \log n)$ time complexity algorithm by using the properties mentioned above.

8.3.2 Greedy algorithm pseudocode

The trick to achieve linear time is to leave a mark as we traverse nodes of the graph, allowing us to visit each vertex a constant number of times. Again, start from all vertexes but maintain an auxiliary data structure. A Union Find Forest [14] to detect if the next vertex in the path belongs to the Cc. of the current path. This Union Find is where the mark is done.

[11] If there are n vertices and n edges, and all outgoing degrees are 1, it is a tree plus the extra edge, or a forest plus extra edges.

[12] All paths lead to a cycle because there must always be a next candidate and GIG is finite.

[13] The demonstration follows from the fact that all paths end in cycles.

So the first path will find that all the vertices it visits aren't yet in the union find, because they haven't yet been visited. Once the path closes on itself, it will detect that the next guard to visit belongs to the same Cc. as itself. And because the Cc. that the path belongs doesn't have any cycle recorded. Store that a cycle has been found for that Cc., storing its length and the sequence $< g_1, g_2 \dots g_m >$ it generates.

The subsequent paths fall into the same mode if they belong to an "unmarked" Cc. So they do the same, but once the cycle is found, compare its length to the stored cycle, and keep the shortest one. But what happens if a path finds a vertex that has already been visited? let's say g_i. Then it means that all the vertices so far visited by the path belong to the same Cc. as the one of g_i. Because the Cc. of g_i has already been explored, its cycle has already been recorded and there is no need to traverse it again.

This schema allows us to start from all vertices, because a vertex is never visited thrice.

The previous step of building the GIG had complexity $O(n \log n)$, that is the final computational complexity of computing G^{opt} from R and B.

For clarity, the pseudocode has been segmented into building the GIG, and then finding G^{opt}.

Algorithm 6 Building the Greedy Interval Graph

Input: R_{pruned}, B
Output: GIG

$CH(B) \leftarrow ConvexHull(B)$
$CovrIntr \leftarrow coverIntervals(R_{pruned}, CH(B))$ ▷ Computing supporting lines
$TruncatedIntervals \leftarrow \emptyset$ ▷ Intervals that contain the direction 0 or π
for $interval \in CovrIntr$ **do**
    if $interval.birthAngle > interval.deathAngle$ **then**
        $TruncatedIntervals \leftarrow TruncatedIntervals \cup interval$
        $CovrIntr \leftarrow CovrIntr \setminus interval$
    end if
end for
for $interval \in TruncatedIntervals$ **do**
    $CovrIntr \leftarrow CovrIntr \cup [interval.birthAngle - \pi, interval.deathAngle]$
end for
for $interval \in TruncatedIntervals$ **do**
    $CovrIntr \leftarrow CovrIntr \cup [interval.birthAngle, interval.deathAngle + \pi]$
end for
▷ Now all intervals have smaller birthAngle than deathAngle
$CovrIntr \leftarrow SortByBirthAngle(CovrIntr)$ ▷ From set to event list
$OpenIntervals \leftarrow [\,]$ ▷ Empty list
$GIG \leftarrow graphFromVertex(R_{pruned})$ ▷ Graph initalization with no edges
$lastToDie \leftarrow null$ ▷ Interval with latest death

```
21: for event ∈ CovrIntr do
22:     if event is birth then
23:         OpenIntervals.append(event.point)  ▷ event.point = point identificator
24:         lastToDie ← maxDeathAngle(lastToDie, event.point)
25:     else if event is death then
26:         GIG.addDirectedEdge(event.point, lastToDie)          ▷ Greedy choice
27:         OpenIntervals.delete(event.point)
28:     end if
29: end for
30: return GIG
```

The code is basically the same scheme used for Algorithm 5. Now the sweep is done maintaining who is the last interval in *OpenIntervals* to die. Using this, generate the edges of the GIG as intervals drop from *OpenIntervals*. In fact because now the input is $R_{prunned}$, we could omit this entirely as we are guaranteed that the last birthed interval inside *OpenIntervals* will be the last to die. This is because the intervals now can't be dominated.

Algorithm 7 Finding an optimal G

Input: GIG
Output: G

```
1: UnionFind ← DisjointSet()                          ▷ Empty Union Find Forest
2: ShortestCycle ← null
3: for v ∈ GIG.V do
4:     path_v ← [ ]
5:     if not(UnionFind.has(v)) then                  ▷ If not already visited
6:         currentCc ← UnionFind.makeSet(v)   ▷ Creates Cc. in with current v
7:         next_v ← GIG.adjacent(v)  ▷ There is only one adjacent vertex in GIG
8:         while not(UnionFind.has(next_v)) do
9:             path_v.append(next_v)
10:            UnionFind.addToSet(next_v, currentCc)
11:            next_v ← GIG.adjacent(next_v)
12:        end while
13:        last_v_Cc ← UnionFind.find(next_v)          ▷ Cc. identifier of last v
14:        if last_v_Cc = currentCc then
15:            cycle_v ← cutCycle(path_v)          ▷ The path has self intersected
16:            ShortestCycle ← minLength(ShortestCycle, cycle_v)
17:        else      ▷ The path intersected another Cc., so it belongs to that Cc.
18:            UnionFind.union(last_v_Cc, currentCc)   ▷ Merge the Cc. markers
19:        end if
20:    end if
21: end for
22: G ← setFromCycle(ShortestCycle)
23: return G                                          ▷ Optimal guard set
```

The *union()* and *find()* operations over the $UnionFind$ data structure have cost $\alpha(n)$, the inverse Ackermann function [14]. The space complexity is $O(n)$. As we execute a linear amount of them, the time complexity of Algorithm 7 is $O(n\alpha(n))$.

Finally, we have an algorithm that finds the smallest set G of guards. Because the sorting ends up being the dominating factor, the final cost for computing G is $O(n \log n)$. Even if the input R and B are of the counterexample kind (Section 6.2) so that $m = n$, the computational complexity is the same. This means that in optimal time $O(n \log n)$ we can detect if the instance can be solved by Algorithm 4 in optimal time, a very nice collateral result.

8.3.3 Further work with greedy algorithms

Some proofs about the greedy algorithm that could not be formalized in time for this bachelor thesis include the elegant:

Open problem 5. *All cycles in the GIG representation have the same length.*

This can intuitively be understood as: If various cycles exist, they are all "interlinked", forcing them to be of the same size. This is a hard to formalize proof, but the best draft on it uses a convincing proof by contradiction. The problem is that it is not sufficient to use the abstract GIG representation, to achieve the contradiction a return to the interval representation is needed. This return to a less abstract representation is what kept the more fine details of the proof pending of formalization. It remains as further work.

If this was to be solved, then executing the algorithm from any arbitrary starting point is allowed, as any cycle we find is the same length as the optimal one. Unfortunately, this doesn't reduce the complexity of the algorithm. But it makes it more elegant, a noble enough pursuit.

With this, the design of Algorithm 4 is completed. It is now an algorithm that, given R and B, can first generate G in $O(n \log n)$ time, check if the number of guards found is small or large (that can be used to estimate the run time of the algorithm), and solve the $4-$separation in $O(mn \log n)$.

This concludes the advances done in answering the Open problem 1.

9 The second open question

Because the whole of this bachelor thesis has been centred in Open problem 1, it is easy to forget that a second question was posed, Open problem 2.

Recall, the Open problem 2 was concerned with solving separability for higher values of k, but achieving $O(kn \log n)$ time. This means finding some kind of recursive algorithm, or extendable algorithm, to systematically solve the higher order separations. Unfortunately, because of time constraints and the limited scope of this thesis, only a description of the further work ahead to solve Open problem 2 is included.

Comparing the Algorithm 1 with Algorithm 3 and Algorithm 4, a few ideas on how to design a recursive algorithm arise. The main insight is that Algorithm 3 worked because it imposed a similar structure to Algorithm 1. The tool used has been rotating calipers over $CH(B)$. But it can be thought that Algorithm 3 rotated a second caliper, this time over the $CH(R_{rec})$, where R_{rec} are the points inside $CH(B)$. So continue to impose that a few blue points lay inside $CH(R_{rec})$, call them B_{rec}, and rotate a third caliper over $CH(B_{rec})$. These calipers would be used in a very similar manner to how they were used in Algorithm 3. See Figure 21.

This nested convex hulls, each with a nested rotating caliper, is the proposal for the recursive algorithm. The drawbacks are clear to see, we are assuming that such nested convex hulls exist to begin with. And in lots of uniform random configurations it might very well be, but this is only speculation.

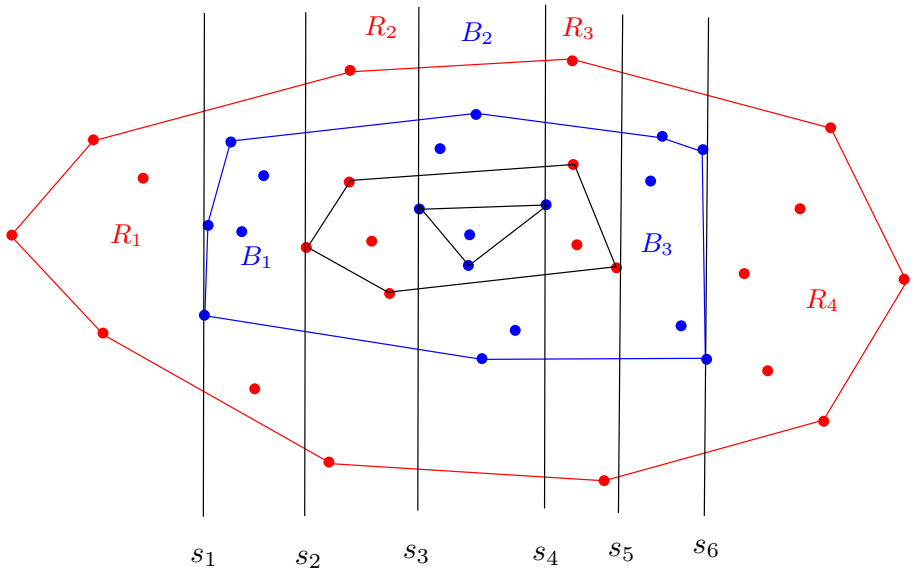

Figure 21: Separability with 6 lines.

So for this recursive algorithm, we are attempting to separate using a number k of lines that is found from the input, not chosen. By similar results to Proposition 2, all the calipers used in this recursive proposal are strictly necessary. In fact, recall that the Proposition 2 was already enunciated for any value of k. Now instead of applying it only once, we apply it recursively.

So the sets will need at least this found k number of lines to be separable. But if the sets are not separable using these k lines, then the algorithm tell us nothing about how many more lines could be needed to achieve separation.

The work ahead in this question is:

- Formalize properly the recursive algorithm.
- Demonstrate that the k lines that the algorithm uses are less or equal than the minimal number k to achieve separation of R and B in all directions.
- Use the relaxed input conditions that the Algorithm 4 achieves over the Algorithm 3, to also relax the input conditions for the nested convex hulls.

This last point is a really important one. Part of the effort of designing Algorithm 4 was to allow for this relaxation. Because the recursive algorithm imposes the condition as many times as it can, until it runs out of nested convex hulls, relaxing this condition would further empower the algorithm by an important factor.

10 Conclusions

In this project, we have considered the problems related to the separability of biromantic point sets, by a set of k parallel lines, classifying the points in alternating monochromatic strips. Extending the results obtained in previous work, specifically the separability using 2 or 3 lines, in order to study the case of 4 or more lines.

We have developed in great depth the 4 lines case, and studied the properties that can be crucial for the design of algorithms to solve the general case of k lines. Trying to understand how complexity emerges as the number of lines used increments.

In more detail, we have developed the use of guards to design the algorithm 4. The strategy has been the use of guards, that have the potential to be a useful approach for values of k greater than 4. The complexity for the 4 lines case has resulted in $O(mn \log n)$ time, where m is the minimal number of guards.

As a general goal, we pretend to find algorithms to solve the general k case, with $O(f(k) \cdot n \log n)$ time complexity, where $f(k)$ is a polynomial function in k.

References

[1] Memòria d'actuacions 2020: pla estratègic ENGINY 2020. *Biblioteques, Universitat Politècnica de Catalunya Servei de Arxius, Publicacions i*, 2021. Accepted: 2022-02-24T11:32:21Z.

[2] G. Brodal and R. Jacob. Dynamic planar convex hull. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 617–626, 2002.

[3] Glassdoor. Sueldo: Junior Researcher en Barcelona, Spain. https://www.glassdoor.es/Sueldos/barcelona-junior-researcher-sueldo-SRCH_IL.0,9_IM1015_KO10,27.htm.

[4] Glassdoor. Sueldo: Project Manager en Barcelona, Spain. https://www.glassdoor.es/Sueldos/barcelona-project-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,25.htm.

[5] R. Jacob and G. S. Brodal. Dynamic Planar Convex Hull. Technical Report arXiv:1902.11169, arXiv, Feb. 2019. arXiv:1902.11169 [cs] type: article.

[6] J. Kratochvil and J. Matousek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62(2):289–315, 1994.

[7] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[8] V. Mäkinen, V. Staneva, A. Tomescu, D. Valenzuela, and S. Wilzbach. Interval scheduling maximizing minimum coverage. *Discrete Applied Mathematics*, 225:130–135, 2017.

[9] F. P. Preparata and M. I. Shamos. Convex hulls: Basic algorithms. In *Computational Geometry*, pages 95–149. Springer, 1985.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Science & Business Media, 2012.

[11] C. Seara. *On geometric separability*. PhD thesis, Universidad Politecnica de Catalunya, 2002.

[12] J. Stoer and C. Witzgall. *Convexity and optimization in finite dimensions I*, volume 163. Springer Science & Business Media, 2012.

[13] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[14] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, apr 1975.

[15] G. Toussaint. Solving geometric problems with the rotating calipers. *Proceedings of IEEE MELECON'83*, page 8, 1983.