1.3 An exact algorithm in the plane

In this section we describe a deterministic linear time algorithm to construct a halving line ℓ for a given set D of n disks in the plane. The line ℓ bisects D perfectly (at most $\lfloor n/2 \rfloor$ centers lie on either side) and it intersects at most $O(n^c)$ disks, where c may be chosen arbitrarily close to $2/3$.

The algorithm follows the prune and search paradigm. And is inspired by the prune and search algorithm to find a ham-sandwich-cut in deterministic linear time [44].

We will not consider all possible halving lines, but restrict our attention to those halving lines with their slope in a certain range. This range becomes smaller and smaller after each iteration. We will discard after each iteration a constant fraction of the disks. Namely, those disks that do not intersect any halving line that we still consider. Each iteration step will take linear time. And thus the overall running time is linear as well.

The section is divided into several subsections. We first repeat some definitions regarding point-line duality in Subsection 1.3.1. ...

Recall, the algorithm follows the prune and search paradigm. In the process we will consider only halving lines with a certain slope. Thus in the dual only the λ-level with certain x-coordinates are considered.

Definition 1 (Slab) *A closed region bounded by two vertical lines is a slab. A slab $S = \{(x, y) \in \mathbb{R}^2 \colon l \leq x \leq r\}$ we denote by $S = [l, r]$. The distance $r - l$ between the two bounding vertical lines is the width of S.*

We will also denote closed intervals with $[l, r]$.

As mentioned before, some disks will be discarded. Namely, those that do not intersect any of the potential halving lines under consideration.

1.3.2. Overview of the Algorithm.

The algorithm works in the dual arrangement and follows the prune and search paradigm. This means we find a point on the $\lfloor n/2 \rfloor$-level of the dual line arrangement which interferes with at most $O(n^\delta)$ lines. We will show that δ can be chosen arbitrarily close to $2/3$.

Recall that we will follow the prune and search algorithm. In our case the algorithm consists of three phases. The algorithm starts with an initialization then goes into some loop and finally returns an appropriate point.

Lemma 12 ... *The main condition is*

$$w \geq c \log n / n. \tag{1}$$

...

The outline of an iteration step is as follows.

1. Divide S in four subslabs $S_1, \ldots, S_4$, such that the width of each subslab equals $w/4$.
2. Let v_i be the number of vertices in slab S_i. Compute the approximate number of vertices p_i in each slab according to Lemma 14 with $c = \epsilon/100$. Subsection 1.3.5 is devoted to the proof of Lemma 14.
3. Define $S' = [l', r']$ as the slab with smallest p_i. We will show that for the number of vertices v' within this slab holds $v' \leq \frac{n^2}{100}$. Subsection 1.3.6 is devoted to proof correctness of this step.
4. Construct a *trapezoid* $T \subseteq S'$ as describe in Lemma 13. By Lemma 13 we know that T contains the λ-level of $\mathcal{A}(L)$ within S' and at most half of the lines from L intersect T. For this step we will use that the number of intersections is low.

   Subsection 1.3.4 is devoted to the proof of this lemma and description of the construction of the trapezoid.
5. We define a 1-*tube* τ of the trapezoid as in the end of Subsection 1.3.1. Further we define the γ-*core* C_γ to be the central $(1-2\gamma)$-section of S', that is, $\mathrm{C}_\gamma = [l' + \gamma w', r' - \gamma w']$. See Figure 7, for an illustration. We continue our search in the next iteration in $\overline{S} = \mathrm{C}_\gamma$. Lemma 15 in Subsection .3.7 shows that at most ϵn lines intersect τ within C_γ. For this step we use Condition 1. Intuitively this means that the width of the slab is not too small.
6. Discard *exactly* $\lfloor (\frac{1}{2} - \epsilon) n \rfloor$ lines from L that do not intersect $\overline{S} \cap (\tau \cup T)$. By Lemma 10 and 11 these lines do not interfere with the λ-level of L within $\overline{S}$ and we can discard them. By Lemma 13 and 15 we know that we can discard at least $(\frac{1}{2} - \epsilon) n$ lines.
7. We adjust λ accordingly: decrease λ by the number of lines discarded that are below τ.

3.1 Introduction

Definition. Given a graph $G = (V, E)$ two players denoted by ALICE and BOB take turns alternatingly in the following manner: in the first turn they choose a start vertex in G (not both the same). Thereafter, in each turn the players pick a vertex that is adjacent to the vertex they have picked in their last turn. It is not allowed to pick a vertex that has been chosen by either of the players before at any stage of the game. Thus the vertices get used and cannot be reused. When both players cannot move the game ends and each player gets a score equal to the number of vertices traversed before they could not move anymore. We denote the score for ALICE and BOB as $\#\mathtt{A}$ and $\#\mathtt{B}$ respectively. We say ALICE *wins* if $\#\mathtt{A} > \#\mathtt{B}$; BOB *wins* if $\#\mathtt{B} > \#\mathtt{A}$; and otherwise we call a play of a game a *tie*. The outcome of a play is defined as $\#\mathtt{B}/\#\mathtt{A}$. We say that the players play *rational*ly if both try to optimize the worst case outcome, i.e., ALICE minimizes and BOB maximizes the worst case outcome.

One could also consider the difference instead of the ratio of $\#\mathtt{A}$ and $\#\mathtt{B}$. We will see that this would lead to less interesting results and strategies.

The extremal question is by how much can ALICE win in comparison to BOB and vice versa. The complexity question asks for the computational complexity class of determining if ALICE has a winning strategy.

The game must end ultimately when all vertices are eaten up. And of course the game is not *loopy*, as it is impossible to return to a previous state of the game. A game is called loopy if it is possible to return to a previous state of the game. We assume that both players have perfect information at all times. The game is defined as *normal play*, that is the players try to move as long as possible, in the sense made precise above. It is also possible to consider the variant of *misère game*, where the players try to be unable to move as soon as possible. We further distinguish whether start vertices are given and whether the graph is directed or undirected.

A problem L is PSPACE-complete if every decision problem, that can be decided with polynomial amount of space can be reduced to L and L can be decided with polynomial amount of space.

The decision problem TRON is the following: the input is a graph and we want to answer the question whether ALICE has a winning strategy. We consider the decision problem under different game modes as described above.

For convenience, we define the length of a path to be the number of vertices.

Franchise. Tron is a 1982 American science fiction movie. It was directed by Steven Lisberger and produced by Disney Studios. After its release a whole cult originated from the movie: several books, ...

4.1 Definitions

The house allocation problem is motivated by the following setup: a set of people is interested to be allocated to a certain set of houses. Each person has a ranking over the set of houses and wants to be assigned to the house with her highest preference. As soon as two people have the same favorite house this is not possible. Motivated by this picture we abstract the set up and start with some definitions.

In an instance of the house allocation problem two sets A and B are given. The set A represents applicants and the set B represents houses. We denote by m and n the size of A and B respectively. In the house allocation problem, we assume that every $a \in A$ has a preference list over the set B. A preference list can be formally defined as a total order of B. We call an injective mapping τ from A to B a matching. A *blocking coalition* of τ is a subset A' of A such that there exists a matching τ' that differs from τ only on elements of A', and every element of A' improves in τ', compared to τ according to its preference list. If there exists no blocking coalition, we call the matching τ a *Pareto optimal matching* (POM).

We represent the preference lists by an $m \times n$ matrix. Every row represents the preference list of one of the applicants in A, i.e., in a given row r corresponding to some applicant $a \in A$, the leftmost house is the one that a prefers most, etc., house b_1 is left to b_2 in r if and only if a prefers b_1 over b_2. Note that no row contains an element from B twice. We usually denote this matrix by M and following this interpretation we usually denote the applicants of A by $r_1, r_2, \dots r_m$ and the houses of B by $1, 2, \dots, n$. Because of this matrix representation, we usually refer to applicants of A only as rows and to houses of B as elements (of the matrix).

To illustrate the notion consider the following matrix and observe that the matching indicated by circles is indeed Pareto optimal.

$$\begin{pmatrix} \textcircled{1} & 5 & 3 & 2 & 4 \\ 3 & 1 & \textcircled{4} & 5 & 2 \\ 1 & \textcircled{3} & 5 & 4 & 2 \end{pmatrix}$$

The image set of τ corresponds to the set of houses of B in these positions. Thus, we say that τ *selects* some position p of M (resp. some element b of B), if p is in τ (resp. b is in the image set of τ). Similarly, we say that a row a *selects* a position P in row a (resp. element b) if this holds for the matching τ under consideration.