On one-round discrete voronoi games

On One-Round Discrete Voronoi Games

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands
m.t.d.berg@tue.nl

Sándor Kisfaludi-Bak
Max Planck Institut für Infromatik, Saarbrücken, Germany
sandor.kisfaludi-bak@mpi-inf.mpg.de

Mehran Mehr
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands
mehran.mehr@gamil.com

Abstract

Let V be a multiset of n points in $\mathbb{R}^d$, which we call voters, and let $k \geqslant 1$ and $\ell \geqslant 1$ be two given constants. We consider the following game, where two players $\mathcal{P}$ and $\mathcal{Q}$ compete over the voters in V: First, player $\mathcal{P}$ selects a set P of k points in $\mathbb{R}^d$, and then player $\mathcal{Q}$ selects a set Q of ℓ points in $\mathbb{R}^d$. Player $\mathcal{P}$ wins a voter $v \in V$ iff $\text{dist}(v, P) \leqslant \text{dist}(v, Q)$, where $\text{dist}(v, P) := \min_{p \in P} \text{dist}(v, p)$ and $\text{dist}(v, Q)$ is defined similarly. Player $\mathcal{P}$ wins the game if he wins at least half the voters. The algorithmic problem we study is the following: given V, k, and ℓ, how efficiently can we decide if player $\mathcal{P}$ has a winning strategy, that is, if $\mathcal{P}$ can select his k points such that he wins the game no matter where $\mathcal{Q}$ places her points.

Banik et al. devised a singly-exponential algorithm for the game in $\mathbb{R}^1$, for the case $k = \ell$. We improve their result by presenting the first polynomial-time algorithm for the game in $\mathbb{R}^1$. Our algorithm can handle arbitrary values of k and ℓ. We also show that if $d \geq 2$, deciding if player $\mathcal{P}$ has a winning strategy is Σ_2^P-hard when k and ℓ are part of the input. Finally, we prove that for any dimension d, the problem is contained in the complexity class $\exists\forall\mathbb{R}$, and we give an algorithm that works in polynomial time for fixed k and ℓ.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Algorithmic game theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases competitive facility location, plurality point

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.37

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.09234`.

Funding MdB, SKB, and MM are supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003, 024.002.003, and 022.005.025, respectively.

1 Introduction

Voronoi games, as introduced by Ahn et al. [1], can be viewed as competitive facility-location problems in which two players $\mathcal{P}$ and $\mathcal{Q}$ want to place their facilities in order to maximize their market area. The Voronoi game of Ahn et al. is played in a bounded region $R \subset \mathbb{R}^2$, and the facilities of the players are modeled as points in this region. Each player gets the same number, k, of facilities, which they have to place alternatingly. The market area of $\mathcal{P}$ (and similarly of $\mathcal{Q}$) is now given by the area of the region of all points $q \in R$ whose closest facility was placed by $\mathcal{P}$, that is, it is the total area of the Voronoi cells of $\mathcal{P}$'s facilities in the Voronoi diagram of the facilities of $\mathcal{P}$ and $\mathcal{Q}$. Ahn et al. proved that for $k > 1$ and when the region R is a circle or a segment, the second player can win the game by a payoff of $1/2 + \varepsilon$, for some $\varepsilon > 0$, where the first player can ensure ε is arbitrarily small.

The one-round Voronoi game introduced by Cheong et al. [8] is similar to the Voronoi game of Ahn et al., except that the first player must first place all his k facilities, after which the second player places all her k facilities. They considered the problem where R is a square, and they showed that when k is large enough the second player can always win a fraction $1/2 + \alpha$ of the area of R for some $\alpha > 0$. Fekete and Meijer [11] considered the problem on a rectangle R of aspect ratio $\rho \leqslant 1$. They showed that the first player wins more than half the area of R, unless $k \geqslant 3$ and $\rho > \sqrt{2}/n$, or $k = 2$ and $\rho > \sqrt{3}/2$. They also showed that if R is a polygon with holes, then computing the locations of the facilities for the second player that maximize the area she wins, against a given set of facilities of the first player is NP-hard.

One-round discrete Voronoi games. In this paper we are interested in *discrete (Euclidean) one-round Voronoi games*, where the players do not compete for area but for a discrete set of points. That is, instead of the region R one is given a set V of n points in a geometric space, and a point $v \in V$ is won by the player owning the facility closest to v. (Another discrete variant of Voronoi games is played on graphs [16, 18] but we restrict our attention to the geometric variant.) Formally, the problem we study is defined as follows.

Let V be a multiset of n points in $\mathbb{R}^d$, which we call *voters* from now on, and let $k \geqslant 1$ and $\ell \geqslant 1$ be two integers. The one-round discrete Voronoi game defined by the triple $\langle V, k, \ell \rangle$ is a single-turn game played between two players $\mathcal{P}$ and $\mathcal{Q}$. First, player $\mathcal{P}$ places a set P of k points in $\mathbb{R}^d$, then player $\mathcal{Q}$ places a set Q of ℓ points in $\mathbb{R}^d$. (These points may coincide with the voters in V.) We call the set P the *strategy of* $\mathcal{P}$ and the set Q the *strategy of* $\mathcal{Q}$. Player $\mathcal{P}$ wins a voter $v \in V$ if $\mathrm{dist}(v, P) \leqslant \mathrm{dist}(v, Q)$, where $\mathrm{dist}(v, P)$ and $\mathrm{dist}(v, Q)$ denote the minimum distance between a voter v and the sets P and Q, respectively. Note that this definition favors player $\mathcal{P}$, since in case of a tie a voter is won by $\mathcal{P}$. We now define $V[P \succeq Q] := \{v \in V : \mathrm{dist}(v, P) \leqslant \mathrm{dist}(v, Q)\}$ to be the multiset of voters won by player $\mathcal{P}$ when he uses strategy P and player $\mathcal{Q}$ uses strategy Q. Player $\mathcal{P}$ wins the game $\langle V, k, \ell \rangle$ if he wins at least half the voters in V, that is, when $\big|V[P \succeq Q]\big| \geqslant n/2$; otherwise $\mathcal{Q}$ wins the game. Here $\big|V[P \succeq Q]\big|$ denotes the size of the multiset $V[P \succeq Q]$ (counting multiplicities). We now define $\Gamma_{k,\ell}(V)$ as the maximum number of voters that can be won by player $\mathcal{P}$ against an optimal opponent:

$$\Gamma_{k,\ell}(V) := \max_{P \subset \mathbb{R}^d,\, |P|=k} \; \min_{Q \subset \mathbb{R}^d,\, |Q|=\ell} \; \big|V[P \succeq Q]\big|.$$

For a given multiset V of voters, we want to decide if[1] $\Gamma_{k,\ell}(V) \geqslant n/2$. In other words, we are interested in determining for a given game $\langle V, k, \ell \rangle$ if $\mathcal{P}$ has a *winning strategy*, which is a set of k points such that $\mathcal{P}$ wins the game no matter where $\mathcal{Q}$ places her points.

An important special case, which has already been studied in spatial voting theory for a long time, is when $k = \ell = 1$ [14]. Here the coordinates of a point in V represent the preference of the voter on certain topics, and the point played by $\mathcal{Q}$ represents a certain proposal. If the point played by $\mathcal{P}$ wins against all possible points played by $\mathcal{Q}$, then the $\mathcal{P}$'s proposal will win the vote against any other proposal. Note that in the problem definition we gave above, voters at equal distance from P and Q are won by $\mathcal{P}$, and $\mathcal{P}$ has to win at least half the voters. This is the definition typically used in papers of Voronoi games [3, 4, 5, 6]. In voting theory other variants are studied as well, for instance where points at equal distance

[1] One can also require that $\Gamma_{k,\ell}(V) > n/2$; with some small modifications, all the results in this paper can be applied to the case with strict inequality as well.

to P and Q are not won by either of them, and $\mathcal{P}$ wins the game if he wins more voters than $\mathcal{Q}$; see the paper by McKelvey and Wendell [14] who use the term *majority points* for the former variant and the term *plurality points* for the latter variant.

Previous work. Besides algorithmic problems concerning the one-round discrete Voronoi game one can also consider combinatorial problems. In particular, one can ask for bounds on $\Gamma_{k,\ell}(V)$ as a function of n, k, and ℓ. It is known that for any set V in $\mathbb{R}^2$ and $k = \ell = 1$ we have $\lfloor n/3 \rfloor \leqslant \Gamma_{1,1}(V) \leqslant \lceil n/2 \rceil$. This result is based on known bounds for maximum Tukey depth, where the lower bound can be proven using Helly's theorem. It is also known [6] that there is a constant c such that $k = c\ell$ points suffice for $\mathcal{P}$ to win, that is, $\Gamma_{c\ell,\ell}(V) \geqslant n/2$ for any V.

In this paper we focus on the algorithmic problem of computing $\Gamma_{k,\ell}(V)$ for given V, k, and ℓ. The problem of deciding if $\Gamma_{k,\ell}(V) \geqslant n/2$ was studied for the case $k = \ell = 1$ by Wu et al. [19] and Lin et al. [13], and later by De Berg et al. [9] who solve this problem in $O(n \log n)$ time in any fixed dimension d. Their algorithms works when V is a set (not a multiset) and for plurality points instead of majority points. Other algorithmic results are for the setting where the players already placed all but one of their points, and one wants to compute the best locations for the last point of $\mathcal{P}$ and of $\mathcal{Q}$. Banik et al. [5] gave algorithms that finds the best location for $\mathcal{P}$ in $O(n^8)$ time and for $\mathcal{Q}$ in $O(n^2)$ time. For the two-round variant of the problem, with $k = \ell = 2$, polynomial algorithms for finding the optimal strategies of both players are also known [4].

Our work is inspired by the paper of Banik et al. [3] on computing $\Gamma_{k,\ell}(V)$ in $\mathbb{R}^1$. They considered the case of arbitrarily large k and ℓ, but where $k = \ell$ (and V is a set instead of a multiset). For this case they showed that depending on the set V either $\mathcal{P}$ or $\mathcal{Q}$ can win the game, and they presented an algorithm to compute $\Gamma_{k,\ell}(V)$ in time $O(n^{k-\lambda_k})$, where $0 < \lambda_k < 1$ is a constant dependent only on k. This raises the question: is the problem NP-hard when k is part of the input?

Our results. We answer the question above negatively, by presenting an algorithm that computes $\Gamma_{k,\ell}(V)$ in $\mathbb{R}^1$ in polynomial time. Our algorithm works when V is a multiset, and it does not require k and ℓ to be equal. Our algorithm computes $\Gamma_{k,\ell}(V)$ and finds a strategy for $\mathcal{P}$ that wins this many voters in time $O(kn^4)$. The algorithm can be extended to the case when the voters are weighted, requiring only a slight increase in running time.

The algorithm by Banik et al. [3] discretizes the problem, by defining a finite set of potential locations for $\mathcal{P}$ to place his points. However, to ensure an optimal strategy for $\mathcal{P}$, the set of potential locations has exponential size. To overcome this problem we need several new ideas. First of all, we partition the possible strategies into various classes – the concept of thresholds introduced later plays this role – such that for each class we can anticipate the behavior of the optimal strategy for $\mathcal{Q}$. To compute the best strategy within a certain class we use dynamic programming, in a non-standard (and, unfortunately, rather complicated) way. The subproblems in our dynamic program are for smaller point sets and smaller values of k and ℓ (actually we will need several other parameters) where the goal of $\mathcal{P}$ will be to push his rightmost point as far to the right as possible to win a certain number of points. One complication in the dynamic program is that it is unclear which small subproblems I' can be used to solve a given subproblem I. The opposite direction – determining for I' which larger I may use I' in their solution – is easier, so we use a sweep approach: when the solution to some I' is determined, we update the solution to larger subproblems I that can use I'.

After establishing that we can compute $\Gamma_{k,\ell}(V)$ in polynomial time in $\mathbb{R}^1$, we turn to the higher-dimensional problem. We show (in the full version) that deciding if $\mathcal{P}$ has a winning strategy is Σ_2^P-hard in $\mathbb{R}^2$. We also show that for fixed k and ℓ this problem can be solved in polynomial time. Our solution combines algebraic methods [7] with a result of Paterson and Zwick [15] that one can construct a polynomial-size boolean circuits that implements the majority function. The latter result in essential to avoid the appearance of n in the exponent. As a byproduct of the algebraic method, we show that the problem is contained in the complexity class $\exists\forall\mathbb{R}$; see [10] for more information on this complexity class.

2 A Polynomial-Time Algorithm for $d = 1$

In this section, we present a polynomial-time algorithm for the 1-dimensional discrete Voronoi game. Our algorithm will employ dynamic programming, and it will be convenient to use n, k, and ℓ as variables in the dynamic program. From now on, we therefore use n^* for the size of the original multiset V, and k^* and ℓ^* for the initial number of points that can be played by $\mathcal{P}$ and $\mathcal{Q}$, respectively.

2.1 Notation and Basic Properties

We denote the given multiset of voters by $V := \{v_1, \ldots, v_{n^*}\}$, where we assume the voters are numbered from left to right. We also always number the points in the strategies $P := \{p_1, \ldots, p_{k^*}\}$ and $Q := \{q_1, \ldots, q_{\ell^*}\}$ from left to right. For brevity we make no distinction between a point and its value (that is, its x-coordinate), so that we can for example write $p_1 < q_1$ to indicate that the leftmost point of P is located to the left of the leftmost point of Q.

For a given game $\langle V, k, \ell\rangle$, we say that a strategy P of player $\mathcal{P}$ *realizes* a gain γ if $\big|V[P \succeq Q]\big| \geqslant \gamma$ for any strategy Q of player $\mathcal{Q}$. Furthermore, we say that a strategy P is *optimal* if it realizes $\Gamma_{k,\ell}(V)$, the maximum possible gain for $\mathcal{P}$, and we say a strategy Q is *optimal* against a given strategy P if $\big|V[P \succeq Q]\big| \leqslant \big|V[P \succeq Q']\big|$ for any strategy Q'.

Trivial, reasonable, and canonical strategies for $\mathcal{P}$. For $0 \leqslant n \leqslant n^*$, define $V_n := \{v_1, \ldots, v_n\}$ to be the leftmost n points in V. Suppose we want to compute $\Gamma_{k,\ell}(V_n)$ for some $1 \leqslant k \leqslant n$ and $0 \leqslant \ell \leqslant n$. The *trivial strategy* of player $\mathcal{P}$ is to place his points at the k points of V_n with the highest multiplicities – here we consider the multiset V_n as a set of distinct points, each with a multiplicity corresponding to the number of times it occurs in V_n – with ties broken arbitrarily. Let $\|V_n\|$ denote the number of distinct points in V_n. Then the trivial strategy is optimal when $k \geqslant \|V_n\|$ and also when $\ell \geqslant 2k$: in the former case $\mathcal{P}$ wins all voters with the trivial strategy, and in the latter case $\mathcal{Q}$ can always win all voters not coinciding with a point in P (namely by surrounding each $p_i \in P$ by two points very close to p_i) so the trivial strategy is optimal for $\mathcal{P}$. Hence, from now on we consider subproblems with $k < \|V_n\|$ and $\ell < 2k$.

We can without loss of generality restrict our attention to strategies for $\mathcal{P}$ that place at most one point in each half-open interval of the form $(v_i, v_{i+1}]$ with $v_i \neq v_{i+1}$, where $0 \leqslant i \leqslant n$, $v_0 := -\infty$, and $v_{n^*+1} := \infty$. Indeed, placing more than two points inside an interval $(v_i, v_{i+1}]$ is clearly not useful, and if two points are placed in some interval $(v_i, v_{i+1}]$ then we can always move the leftmost point onto v_i. (If v_i is already occupied by a point in P, then we can just put the point on any unoccupied voter; under our assumption that $k < \|V_n\|$ an unoccupied voter always exists.) We will call a strategy for $\mathcal{P}$ satisfying the property above *reasonable*.

▶ **Observation 1** (Banik et al. [3]). *Assuming $k < \|V_n\|$ there exist an optimal strategy for $\mathcal{P}$ that is reasonable and has $p_1 \in V$ (that is, p_1 coincides with a voter).*

We can define an ordering on strategies of the same size by sorting them in lexicographical order. More precisely, we say that a strategy $P = \{p_1, \ldots, p_k\}$ is *greater than* a strategy $P' = \{p'_1, \ldots, p'_k\}$, denoted by $P \succ P'$, if $\langle p_1, \ldots, p_k\rangle >_{\text{lex}} \langle p'_1, \ldots, p'_k\rangle$, where $>_{\text{lex}}$ denotes the lexicographical order. Using this ordering, the largest reasonable strategy P that is optimal – namely, that realizes $\Gamma_{k,\ell}(V_n)$ – is called the *canonical strategy* of $\mathcal{P}$.

α-gains, β-gains, and gain sequences. Consider a strategy $P := \{p_1, \ldots, p_k\}$. It will be convenient to add two extra points to P, namely $p_0 := -\infty$ and $p_{k+1} := \infty$; this clearly does not influence the outcome of the game. The strategy P thus induces $k+1$ open intervals of the form (p_i, p_{i+1}) where player $\mathcal{Q}$ may place her points. It is easy to see that there exists an optimal strategy for $\mathcal{Q}$ with the following property: Q contains at most two points in each interval (p_i, p_{i+1}) with $1 \leqslant i \leqslant k-1$, and at most one point in (p_0, p_1) and at most one point in (p_k, p_{k+1}). From now on we restrict our attention to strategies for $\mathcal{Q}$ with this property.

Now suppose that x and y are consecutive points (with $x < y$) in some strategy P, where x could be $-\infty$ and y could be ∞. As just argued, $\mathcal{Q}$ either places zero, one, or two points inside (x, y). When $\mathcal{Q}$ places zero points, then she obviously does not win any of the voters in $V_n \cap (x, y)$. The maximum number of voters $\mathcal{Q}$ can win from $V_n \cap (x, y)$ by placing a single point is the maximum number of voters in (x, y) that can be covered by an open interval of length $(y - x)/2$, as by placing her point in any $q \in (x, y)$, $\mathcal{Q}$ wins all (and only) the voters in the open interval $((x + q)/2, (q + y)/2)$ of length $(y - x)/2$; see Banik et al. [3]. We call this value the *α-gain* of $\mathcal{Q}$ in (x, y) and denote it by $\text{gain}_\alpha(V_n, x, y)$. By placing two points inside (x, y), one immediately to the right of x and one immediately to the left of y, player $\mathcal{Q}$ will win all voters $V_n \cap (x, y)$. Thus the extra number of voters won by the second point in (x, y) as compared to just placing a single point is equal to $|V_n \cap (x, y)| - \text{gain}_\alpha(V_n, x, y)$. We call this quantity the *β-gain* of Q in (x, y) and denote it by $\text{gain}_\beta(V_n, x, y)$. Note that for intervals (x, ∞) we have $\text{gain}_\alpha(x, \infty) = |V_n \cap (x, \infty)|$ and $\text{gain}_\beta(x, \infty) = 0$; a similar statement holds for $(-\infty, y)$.

The following observation follows from the fact that $\text{gain}_\alpha(V_n, x, y)$ equals the maximum number of voters in (x, y) that can be covered by an open interval of length $(y - x)/2$.

▶ **Observation 2** (Banik et al. [3]). *For any x, y we have $gain_\alpha(V_n, x, y) \geqslant gain_\beta(V_n, x, y)$.*

Let $P := \{p_0, p_1, \ldots, p_k, p_{k+1}\}$ be a given strategy for $\mathcal{P}$, where by convention $p_0 = -\infty$ and $p_{k+1} = \infty$. Consider $\{\text{gain}_\alpha(V_n, p_i, p_{i+1}) : 0 \leqslant i \leqslant k\} \cup \{\text{gain}_\beta(V_n, p_i, p_{i+1}) : 0 \leqslant i \leqslant k\}$, the multiset of all α-gains and β-gains defined by the intervals $(p_i, p_i + 1)$. Sort this sequence in non-increasing order, using the following tie-breaking rules if two gains are equal:

- Gains from the interval (p_i, p_{i+1}) have precedence over gains from intervals (p_j, p_{j+1}) when $i < j$.
- if both gains are for the same interval (p_i, p_{i+1}) then the α-gain precedes the β-gain.

We call the resulting sorted sequence the *gain sequence* induced by P on V_n. We denote this sequence by $\Sigma_{\text{gain}}(V_n, P)$ or, when P and V_n are clear from the context, by Σ_{gain}.

The canonical strategy of $\mathcal{Q}$ and sequence representations. Given the multiset V_n, a strategy P and value ℓ, player $\mathcal{Q}$ can compute an optimal strategy as follows. First she computes the gain sequence $\Sigma_{\text{gain}}(V_n, P)$ and chooses the first ℓ gains in $\Sigma_{\text{gain}}(V_n, P)$. Then for each $0 \leqslant i \leqslant k$ she proceeds as follows. When $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ and $\text{gain}_\beta(V_n, p_i, p_{i+1})$ are both chosen, she places two points in (p_i, p_{i+1}) that win all voters in (p_i, p_{i+1}); when only

$\text{gain}_\alpha(V_n, p_i, p_{i+1})$ is chosen, she places one point in (p_i, p_{i+1}) that win $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ voters. (By Observation 2 and the tie-breaking rules, when $\text{gain}_\beta(V_n, p_i, p_{i+1})$ is chosen it is always the case that $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ is also chosen.) The resulting strategy Q is optimal as highest possible gains are chosen and is called the *canonical strategy* of $\mathcal{Q}$ with ℓ points against P on V_n.

From now one we restrict the strategies of player $\mathcal{Q}$ to canonical strategies. In a canonical strategy, player $\mathcal{Q}$ places at most two points in any interval induced by a strategy $P = \{p_0, \ldots, p_{k+1}\}$, and when we know that $\mathcal{Q}$ places a single point (and similarly when she places two points) then we also know where to place the point(s). Hence, we can represent a canonical strategy Q, for given V_n and P, by a sequence $M(V, P, Q) \coloneqq \langle m_0, \ldots, m_k \rangle$ where $m_i \in \{0, 1, 2\}$ indicates how many points $\mathcal{Q}$ plays in the interval (p_i, p_{i+1}). We call $M(V, P, Q)$ the *sequence representation* of the strategy Q against P on V_n. We denote the sequence representation of the canonical strategy of $\mathcal{Q}$ with ℓ points against P on V_n by $M(V, P, \ell)$.

▶ **Observation 3.** *The canonical strategy of $\mathcal{Q}$ with ℓ points against P is the optimal strategy Q with ℓ points against P which has lexicographically maximal sequence representation.*

2.2 The Subproblems for a Dynamic-Programming Solution

For clarity, in the rest of Section 2 we assume the multiset of voters V does not have repetitive entries, i.e we have a set of voters, and not a multiset. While all the results are easily extendible to multisets, dealing with them adds unnecessary complexity to the text.

Our goal is to develop a dynamic-programming algorithm to compute $\Gamma_{k^*,\ell^*}(V)$. Before we can define the subproblems on which the dynamic program is based, we need to introduce the concept of *thresholds*, which is a crucial ingredient in the subproblems.

Strict and loose thresholds. Recall that in an arbitrary gain sequence $\Sigma_{\text{gain}}(V_n, P) = \langle \tau_1, \ldots, \tau_{2k+2} \rangle$, each τ_j is the α-gain or β-gain of some interval (p_i, p_{i+1}), and that these gains are sorted in non-increasing order. We call any integer value $\tau \in [\tau_{\ell+1}, \tau_\ell]$ an *ℓ-threshold* for $\mathcal{Q}$ induced by P on V_n, or simply a *threshold* if ℓ is clear from the context. We implicitly assume $\tau_0 \coloneqq n$ so that talking about 0-threshold is also meaningful. Note that when $\tau_\ell \geqslant \tau > \tau_{\ell+1}$ then the canonical strategy for $\mathcal{Q}$ chooses all gains larger than τ and no gains smaller or equal to τ. Hence, we call τ a *strict* threshold if $\tau_\ell \geqslant \tau > \tau_{\ell+1}$. On the other hand, when $\tau = \tau_{\ell+1}$ then gains of value τ may or may not be chosen by the canonical strategy of $\mathcal{Q}$. (Note that in this case for gains of value τ to be picked, we would actually need $\tau_\ell = \tau = \tau_{\ell+1}$.) In this case we call τ a *loose* threshold.

The idea will be to guess the threshold τ in an optimal solution and then use the fact that fixing the threshold τ helps us to limit the strategies for $\mathcal{P}$ and anticipate the behavior of $\mathcal{Q}$. Let P_{opt} be the canonical strategy realizing $\Gamma_{k^*,\ell^*}(V)$. We call any ℓ^*-threshold of P_{opt} an *optimal threshold*. We devise an algorithm that gets a value τ as input and computes $\Gamma_{k^*,\ell^*}(V)$ correctly if τ is an optimal threshold, and computes a value not greater than $\Gamma_{k^*,\ell^*}(V)$, otherwise.

Clearly we only need to consider values of τ that are at most n^*. In fact, since each α-gain or β-gain in a given gain sequence corresponds to a unique subset of voters, the ℓ^*-th largest gain can be at most n^*/ℓ^*, so we only need to consider τ-values up to $\lfloor n^*/\ell^* \rfloor$. Observe that when there exists an optimal strategy that induces an ℓ^*-threshold equal to zero, then $\mathcal{Q}$ can win all voters not explicitly covered by P. In this case the trivial strategy is optimal for $\mathcal{P}$. Our global algorithm is now as follows.

1. For all thresholds $\tau \in \{1, \ldots, \lfloor n^*/\ell^* \rfloor\}$, compute an upper bound on the number of voters $\mathcal{P}$ can win with a strategy that has an ℓ^*-threshold τ. For the run where τ is an optimal threshold, the algorithm will return $\Gamma_{k^*,\ell^*}(V)$.
2. Compute the number of voters $\mathcal{P}$ wins in the game $\langle V, k^*, \ell^* \rangle$ by the trivial strategy.
3. Report the best of all solutions found.

The subproblems for a fixed threshold τ. From now on we consider a fixed threshold value $\tau \in \{1, \ldots, \lfloor n^*/\ell^* \rfloor\}$. The subproblems in our dynamic-programming algorithm for the game $\langle V, k^*, \ell^* \rangle$ have several parameters.

- A parameter $n \in \{0, \ldots, n^*\}$, specifying that the subproblem is on the voter set V_n.
- Parameters $k, \ell \in \{0, \ldots, n\}$, specifying that $\mathcal{P}$ can use $k+1$ points and $\mathcal{Q}$ can use ℓ points.
- A parameter $\gamma \in \{0, \ldots, n\}$, specifying the number of voters $\mathcal{P}$ must win.
- A parameter $\delta \in \{\text{strict}, \text{loose}\}$, specifying the strictness of the fixed ℓ-threshold τ.

Intuitively, the subproblem specified by a tuple $\langle n, k, \ell, \gamma, \delta \rangle$ asks for a strategy P where $\mathcal{P}$ wins at least γ voters from V_n and such that P that induces an ℓ-threshold of strictness δ, against an opponent $\mathcal{Q}$ using ℓ points. Player $\mathcal{P}$ may use $k+1$ points and his objective will be to push his last point, p_{k+1} as far to the right as possible. The value of the solution to such a subproblem, which we denote by $X_{\max}(n, k, \ell, \gamma, \delta)$, will indicate how far to the right we can push p_{k+1}. Ultimately we will be interested in solutions where $\mathcal{P}$ can push p_{k^*+1} all the way to $+\infty$, which means he can actually win γ voters by placing only k^* points.

To formally define $X_{\max}(n, k, \ell, \gamma, \delta)$, we need two final pieces of notation. Let $x \in \mathbb{R} \cup \{-\infty\}$, let $n \in \{1, \ldots, n^*\}$, and let a, b be integers. For convenience, define $v_{n^*+1} := \infty$. Now we define the *(a,b)-span of x to v_{n+1}*, denoted by $\text{span}(x, n, a, b)$, as

$$\text{span}(x, n, a, b) := \begin{cases} \text{the maximum real value } y \in (v_n, v_{n+1}] \text{ such that} \\ \text{gain}_\alpha(V, x, y) = a \text{ and } \text{gain}_\beta(V, x, y) = b & \text{if } x \neq -\infty \text{ and } y \text{ exists} \\ -\infty & \text{otherwise.} \end{cases}$$

If we let $a := \text{gain}_\alpha(V_n, x, y)$ and $b := \text{gain}_\beta(V_n, x, y)$, then player $\mathcal{Q}$ wins either 0, a, or $a+b$ points depending on whether she plays 0, 1, or 2 points inside the interval. It will therefore be convenient to introduce the notation $\oplus_j$ for $j \in \{0, 1, 2\}$, which is defined as

$$a \oplus_0 b := 0, \qquad a \oplus_1 b := a, \qquad a \oplus_2 b := a + b.$$

We assume the precedence of these operators are higher than addition.

▶ **Definition 4.** *For parameters $n \in \{0, \ldots, n^*\}$, $k, \ell, \gamma \in \{0, \ldots, n\}$, and $\delta \in \{\text{strict}, \text{loose}\}$, we define the value $X_{\max}(n, k, \ell, \gamma, \delta)$ and what it means when a strategy P* realizes *this, as follows.*

- *For $k = 0$, we call it an* elementary subproblem*, and define $X_{\max}(n, k, \ell, \gamma, \delta) = v_{n+1}$ if*
  1. *$\{v_{n+1}\}$ wins at least γ voters from V_n, and*
  2. *$\{v_{n+1}\}$ induces an ℓ-threshold τ with strictness δ on V_n,*

  and we define $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$ otherwise. In the former case we say that $P := \{v_{n+1}\}$ realizes *$X_{\max}(n, k, \ell, \gamma, \delta)$.*
- *For $k > 0$, we call it a* non-elementary subproblem*, and $X_{\max}(n, k, \ell, \gamma, \delta)$ is defined to be equal to the maximum real value $y \in (v_n, v_{n+1}]$ such that there exists a strategy $P := P' \cup \{y\}$ with $P' = \{p_1, \ldots, p_k\}$, integer values n', a, b with $0 \leqslant n' < n$ and $0 \leqslant a, b \leqslant n$, an integer $j \in \{0, 1, 2\}$, and a $\delta' \in \{\text{strict}, \text{loose}\}$ satisfying the following conditions:*

1. *P wins at least γ voters from V_n,*
2. *P induces an ℓ-threshold τ with strictness δ on V_n,*
3. $\operatorname{span}(p_k, n, a, b) = y$,
4. *P' realizes $X_{\max}(n', k-1, \ell-j, \gamma-n+n'+a \oplus_j b, \delta')$,*
5. *Let $M(V_{n'}, P', \ell - j) \coloneqq \langle m'_0, \dots, m'_k \rangle$ and $M(V_n, P, \ell) \coloneqq \langle m_0, \dots, m_{k+1} \rangle$. Then $m'_i = m_i$ for all $0 \leqslant i < k$.*

When a set P satisfying the conditions exists, we say that P realizes *$X_{\max}(n, k, \ell, \gamma, \delta)$. We define $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$ if no such P exists.*

For example, given voters $V = \{1, \dots, 6\}$, $k^* = 3$, $l^* = 3$, and $n^* = 6$, $X_{\max}(3, 0, 1, 2, \text{strict}) = -\infty$ because $\mathcal{P}$ cannot win voters without placing any points and $X_{\max}(3, 2, 1, 2, \text{strict}) = 4$ because player $\mathcal{P}$ can easily win two of the first three voters by placing his first two points on them and then push his third point to the far right to position 4; note that $\tau = 1$ is an strict induced 1-threshold in this case.

Intuitively, each prefix of a canonical strategy of $\mathcal{P}$ is a realizing strategy to some of these subproblems, which is consistent with Definition 4 as realizing strategies to subproblems are defined to be a realizing strategy to a smaller subproblem plus the last point of $\mathcal{P}$ pushed to the rightmost possible position, where $\mathcal{Q}$'s response would be the same except she has the chance to place $j \in \{0, 1, 2\}$ without violating the conditions mentioned in the definition.

By induction we can show that if the parameters n, k, ℓ are not in a certain range, namely if one of the conditions $\ell < 2(k+1)$ or $k \leqslant \|V_n\|$ is violated, then $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$. The next lemma shows we can compute $\Gamma_{k^*,\ell^*}(V)$ from the solutions to our subproblems.

▶ **Lemma 5.** *Let $V = \{v_1, \dots, v_{n^*}\}$ be a set of n^* voters in $\mathbb{R}^1$. Let $0 \leqslant k^* \leqslant n^*$ and $1 \leqslant \ell^* \leqslant n^*$ be two integers such that $\ell^* < 2k^*$ and $k^* < \|V\|$, and let τ be a fixed threshold. Then*

$$\Gamma_{k^*,\ell^*}(V) \;\geqslant\; \text{the maximum value of } \gamma \text{ with } 0 \leqslant \gamma \leqslant n^* \text{ for which there exist a } \delta \in \{\text{loose}, \text{strict}\} \text{ such that } X_{\max}(n^*, k^*, \ell^*, \gamma, \delta) = \infty. \tag{1}$$

Moreover, for an optimal threshold $\tau_{\text{opt}} > 0$, the inequality changes to equality.

▶ Remark. Usually in dynamic programming, subproblems have a clean non-recursive definition – the recursion only comes in when a recursive formula is given to compute the value of an optimal solution. Our approach is more complicated: Definition 4 above gives a recursive subproblem definition (and Lemma 5 shows how to use it), however, using this recursive formula to compute solutions is not feasible and Lemma 7 below will then give a different recursive formula to actually compute the solutions to the subproblems.

2.3 Computing Solutions to Subproblems

The solution to an elementary subproblem follows fairly easily from the definitions, and can be computed in constant time; see the full version.

By definition, in order to obtain a strategy P realizing the solution to a non-elementary subproblem $I = \langle n, k, \ell, \gamma, \delta \rangle$ of size k, we need a solution to a smaller subproblem $I' = \langle n', k-1, \ell', \gamma', \delta' \rangle$ of size $k-1$ and add one point $y \in (v_n, v_{n+1}]$ to the strategy $P' = \{p_1, \dots, p_k\}$ realizing I'. Thus by adding y, we extend the solution to I' to get a solution to I. To find the "right" subproblem I', we guess some values for n', a, b, $j \in \{0, 1, 2\}$, and $\delta' \in \{\text{strict}, \text{loose}\}$; these values are enough to specify I'. We note that there are just a polynomial number of cases and therefore a polynomial number of values for the value $y \in (v_n, v_{n+1}]$ which we want to maximize. Namely, there are $O(n)$ choices for the values n', a, and b, three

choices for j, and two choices for δ'. This makes $O(n^3)$ different cases to be considered for each subproblem I, in total. However, not all those subproblems can be extended to I. In the following definition, we list all the triples (δ', j, δ) that can guarantee the extendibility of I' to I.

Let a and b be the α-gain and β-gain of the interval (p_k, y) in a strategy $P = \{p_1, \ldots, p_k, y\}$ with threshold τ. We define the following sets of triples:

$$\Delta(\tau, a, b) :=$$
$$\begin{cases} \{(\text{loose}, 2, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a > \tau \wedge b > \tau \\ \{(\text{loose}, 1, \text{loose}), (\text{strict}, 1, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a > \tau \wedge b = \tau \\ \{(\text{loose}, 1, \text{loose}), (\text{strict}, 1, \text{strict})\} & \text{if } a > \tau \wedge b < \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{loose}), (\text{strict}, 1, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a = \tau \wedge b = \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{loose}), (\text{strict}, 1, \text{strict})\} & \text{if } a = \tau \wedge b < \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{strict})\} & \text{if } a < \tau \wedge b < \tau. \end{cases}$$

▶ **Lemma 6.** *Let $P' = \{p_1, \ldots, p_k\}$ and $P := P' \cup \{y\}$, be two reasonable strategies on $V_{n'}$ and V_n, where $n' = \operatorname{argmax}_{1 \leqslant i \leqslant n^*} v_i < p_k$, $n = \operatorname{argmax}_{1 \leqslant i \leqslant n^*} v_i < y$, and $y \in (v_n, v_{n+1}]$. Let $a = gain_\alpha(V_n, p_k, y)$ and $b = gain_\beta(V_n, p_k, y)$, and assume $\tau > 0$ is an $(\ell - j)$-threshold of strictness δ' for $\mathcal{Q}$ induced by P' on $V_{n'}$, where $j \in \{0, 1, 2\}$. Then, there exists a triple $(\delta', j, \delta) \in \Delta(\tau, a, b)$ if and only if*

1. *P induces an ℓ-threshold τ with strictness δ on V_n.*
2. *Let $M(V_{n'}, P', \ell - j) := \langle m'_0, \ldots, m'_k \rangle$ and $M(V_n, P, \ell) := \langle m_0, \ldots, m_{k+1} \rangle$. Then $m'_i = m_i$ for all $0 \leqslant i < k$.*

Moreover, this triple is unique if it exists.

▶ **Lemma 7.** *For a non-elementary subproblem $I = \langle n, k, \ell, \gamma, \delta \rangle$, we have*

$$X_{\max}(n, k, \ell, \gamma, \delta) = \max_{0 \leqslant n' < n} \max_{0 \leqslant a \leqslant n} \max_{0 \leqslant b \leqslant n} \max_{(\delta', j, \delta) \in \Delta(\tau, a, b)} \operatorname{span}\big(X_{\max}(n', k-1, \ell - j, \gamma - n + n' + a \oplus_j b, \delta'), n, a, b\big).$$

If we can compute the span function efficiently, we can compute all the solutions by dynamic programming and solve the problem. However, a solution based on a trivial dynamic programming will have running time $\lfloor n^*/\ell^* \rfloor \cdot O(k^*\ell^*(n^*)^2) \cdot O((n^*)^3 f(n^*)) = O(k^*(n^*)^6 f(n^*))$, where $\lfloor n^*/\ell^* \rfloor$ is the total number of choices for the threshold, $O(k^*\ell^*(n^*)^2)$ is the number of subproblems for each threshold, and $O((n^*)^3 f(n^*))$ is the time needed to solve each subproblem where $f(n)$ is the time needed to compute the $\operatorname{span}(x, n, a, b)$ function. This algorithm is quite slow. More importantly it is not easy to compute the span function. In the following, we introduce some new concepts to compute the span function and also get a better running time.

2.4 Computing the span Function Using Gain Maps

Before we give the algorithm we introduce the *gain map*, which we need to compute the span function. Consider an arbitrary strategy P of $\mathcal{P}$ on V, and recall that such a strategy induces open intervals of the form (p_i, p_{i+1}) where $\mathcal{Q}$ can place her points. We can represent any interval (x, y) that may arise in this manner as a point (x, y) in the plane. Thus the locus of all possible intervals is the region $R := \{(x, y) : x < y\}$. We will define two subdivisions of this region, the A-map and the B-map, and the gain map will then be the overlay of the A-map and the B-map.

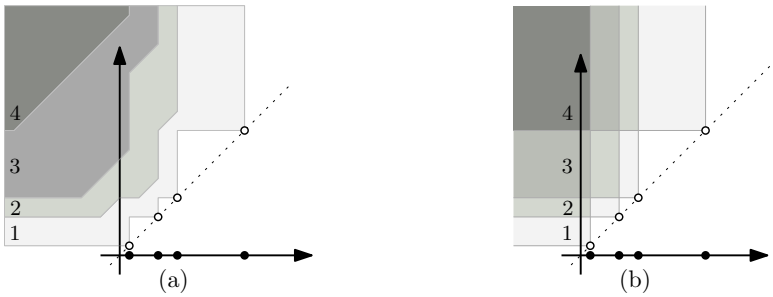

Figure 1 a) A-map of $V = \{1, 4, 6, 13\}$ with the corresponding α-gain for each region. **b)** B-map of V with the corresponding β-gain for each region.

The A-map is the subdivision of R into regions A^t and B^t, for $0 \leqslant t \leqslant n^*$, defined as $A^t := \{(x, y) : \text{gain}_\alpha(V, x, y) = t\}$ and $B^t := \{(x, y) : \text{gain}_\alpha(V, x, y) + \text{gain}_\beta(V, x, y) = t\}$. In other words, A^t is the locus of all intervals (x, y) such that, if (x, y) is an interval induced by P, then $\mathcal{Q}$ can win t voters (but no more than t) from $V \cap (x, y)$ by placing a single point in (x, y). To construct the A-map, let $A^{\geqslant t}$ denote the locus of all intervals (x, y) such that $\text{gain}_\alpha(V, x, y) \geqslant t$. Note that $A^t = A^{\geqslant t} \setminus A^{\geqslant t+1}$. For $1 \leqslant i \leqslant n^* - t + 1$, let $V_i^t := \{v_i, \ldots, v_{i+t-1}\}$ and define

$$A_i^{\geqslant t} := \{(x, y) : V_i^t \subset (x, y) \text{ and } \mathcal{Q} \text{ can win } V_i^t \text{ by placing a single point in } (x, y)\}.$$

Thus $A_i^{\geqslant t} = \{(x, y) : x < v_i \text{ and } y > v_{i+t-1} \text{ and } y > x + 2(v_{i+t-1} - v_i)\}$. Here the conditions $x < v_i$ and $y > v_{i+t-1}$ are needed to guarantee that $V_i^t \subset (x, y)$. The condition $y > x + 2(v_{i+t-1} - v_i)$ implies that V_i^t can be covered with an interval of length $(y - x)/2$, which is necessary and sufficient for $\mathcal{Q}$ to be able to win all these voters. Note that each region $A_i^{\geqslant t}$ is the intersection of three halfplanes, bounded by a vertical, a horizontal and a diagonal line, respectively.

Since $\mathcal{Q}$ can win at least t voters in inside (x, y) with a single point if she can win at least t consecutive voters with a single point, we have $A^{\geqslant t} = \bigcup_{i=1}^{n^*-t+1} A_i^{\geqslant t}$. Thus $A^{\geqslant t}$ is a polygonal region, bounded from below and from the right by a a polyline consisting of horizontal, vertical, and diagonal segments, and the regions A^t are sandwiched between such polylines; see Figure 1a. We call the polylines that form the boundary between consecutive regions A^t *boundary polylines*.

The B-map can be constructed in a similar, but easier manner. Indeed, B^t is the locus of all intervals such that $\mathcal{Q}$ can win t voters (but no more) from $V \cap (x, y)$, and this is the case if and only if $|V \cap (x, y)| = t$. Hence, B^t is the union of the rectangular regions $[v_i, v_{i+1}) \times (v_{i+t}, v_{i+t+1}]$ (intersected with R), for $0 \leqslant i \leqslant n^* - t$, where $v_0 := -\infty$ and $v_{n^*+1} := \infty$, as shown in Figure 1b.

As mentioned, by overlaying the A- and B-map, we get the *gain map*. For any given region on this map, the corresponding intervals have equal α-gain and equal β-gain.

▶ **Lemma 8.** *The complexity of the gain-map is $O((n^*)^2)$.*

Proof. The boundary polylines in the A-map are xy-monotone and comprised of vertical, horizontal, and diagonal lines. The B-map is essentially a grid of size $O((n^*)^2)$ defined by the lines $x = v_i$ and $y = v_i$, for $1 \leqslant i \leqslant n^*$. Since each of these lines intersects any xy-monotone polyline at most once – in a point or in a vertical segment – the complexity of the gain map is also $O((n^*)^2)$. ◀

Using the gain map, we can compute the values $\text{span}(x_0, n, a, b)$ for a given $x_0 \in \mathbb{R}$ and for all triples n, a, b satisfying $1 \leqslant n \leqslant n^*$, and $0 \leqslant a \leqslant n^*$ and $0 \leqslant b \leqslant n^*$, as follows. First, we compute the intersection points of the vertical line $x = x_0$ with (the edges of) the gain map, sorted by increasing y-coordinates. (If this line intersects the gain map in a vertical segment, we take the topmost endpoint of the segment.) Let $(x_0, y_1), \ldots, (x_0, y_z)$ denote this sorted sequence of intersection points, where $z \leqslant 2n^*$ denotes the number of intersections. Let a_i and b_i denote the α-gain and β-gain of the interval corresponding to the point (x_0, y_i), and let a_{z+1} and b_{z+1} denote the α-gain and β-gain of the unbounded region intersected by the line $x = x_0$. Define $n_i = \text{argmax}_n\, v_n < y_i$. Then we have

$$\text{span}(x_0, n, a, b) = \begin{cases} y_i & \text{if } a = a_i \text{ and } b = b_i, \text{ and } n = n_i, \text{ for some } 1 \leqslant i \leqslant z \\ +\infty & \text{if } a = a_{z+1} \text{ and } b = b_{z+1} \text{ and } n = n^* \\ -\infty & \text{for all other triples } n, a, b \end{cases} \tag{2}$$

Our algorithm presented below moves a sweep line from left to right over the gain map. During the sweep we maintain the intersections of the sweep line with the gain map. It will be convenient to maintain the intersections with the A-map and the B-map separately. We will do so using two sequences, $A(x_0)$ and $B(x_0)$.

- The sequence $A(x_0)$ is the sequence of all diagonal or horizontal edges in the A-map that are intersected by the line $x = x_0$, ordered from bottom to top along the line. (More precisely, the sequence contains (at most) one edge for any boundary polyline. When the sweep line reaches the endpoint of such an edge, the edge will be removed and it will be replaced by the next non-vertical edge of that boundary polyline, if it exists.)
- The sequence $B(x_0)$ is the sequence of the y-coordinates of the horizontal segments in the B-map intersected by the line $x = x_0$, ordered from bottom to top along the line.

The number of intersections of the A-map, and also of the B-map, with the line $x = x_0$ is equal to $n^* - n_0 + 1$, where $n_0 = \text{argmin}_n\, v_n > x_0$. Hence, the sequences $A(x_0)$ and $B(x_0)$ have length $n^* - n_0 + 1 \leqslant n^* + 1$.

If we have the sequences $A(x_0)$ and $B(x_0)$ available then, using Equation (2), we can easily find all triples n, a, b such that $\text{span}(x_0, n, a, b) \neq -\infty$ (and the corresponding y-values) by iterating over the two sequences. We can summarize the results of this section as follows.

▶ **Observation 9.** *Given the sequences $A(x_0)$ and $B(x_0)$, we can compute all the values $\text{span}(x_0, n, a, b)$ with $1 \leqslant n \leqslant n^*$ and $0 \leqslant a, b \leqslant n$ that are not equal to $-\infty$ in $O(n^*)$ time in total.*

This observation, together with Lemma 7 forms the basis of our dynamic-programming algorithm.

2.5 The Sweep-Line Based Dynamic-Programming Algorithm

We will use a sweep-line approach, moving a vertical line from left to right over the gain map. We will maintain a table X, indexed by subproblems, such that when the sweep line is at position x_0, then $X[I]$ holds the best solution known so far for subproblem I, where the effect of all the subproblems with solution smaller than x_0 have been taken into account. When our sweep reaches a subproblem I', then we check which later subproblems I can use I' in their solution, and we update the solutions to these subproblems.

Recall that the algorithm works with a fixed threshold value $\tau \in \{1, \ldots, \lfloor n^*/\ell^* \rfloor\}$ and that its goal is to compute the values $X_{\max}(n^*, k^*, \ell^*, \gamma, \delta)$ for all $0 \leqslant \gamma \leqslant n^*$ and $\delta \in \{\text{strict}, \text{loose}\}$. Our algorithm maintains the following data structures.

- $A[0..n^*]$ is an array that stores the sequence $A(x_0)$, where x_0 is the current position of the sweep line and $A[i]$ contains the i-th element in the sequence. When the i-th element does not exist then $A[i] = \textsc{Nil}$.
- Similarly, $B[0..n^*]$ is an array that stores the sequence $B(x_0)$.
- X: This is a table with an entry for each subproblem $I = \langle n, k, \ell, \gamma, \delta\rangle$ with $0 \leqslant n \leqslant n^*$, and $0 \leqslant k \leqslant k^*$ and $0 \leqslant \ell \leqslant \ell^*$, and $0 \leqslant \gamma \leqslant n^*$ and $\delta \in \{\text{strict}, \text{loose}\}$. When the sweep line is at position x_0, then $X[I]$ holds the best solution known so far for subproblem I, where the effect of all the subproblems with solution smaller than x_0 have been taken into account. More precisely, in the right-hand side of the equation in Lemma 7 we have taken the maximum value over all subproblems $I' = \langle n', k-1, \ell - j, \gamma - n + n' + a \oplus_j b, \delta'\rangle$ with $X_{\max}(I') < x_0$. In the beginning of the algorithm the entries for elementary subproblems are computed in constant time and all other entries have value $-\infty$.
- E: This is the event queue, which will contain four types of events, as explained below.

The event queue E is a min-priority queue on the x-value of the events. There are four types of events, as listed next, and when events have the same x-value then the first event type (in the list below) has higher priority, that is, will be handled first. When two events of the same type have equal x-value then their order is arbitrary. Note that events with the same x-value are not degenerate cases – this is inherent to the structure of the algorithms, as many events take place at x-coordinates corresponding to voters.

A-map events, denoted by $e_A(a, s, s')$: At an A-map event, the edge s of the A-map ends – thus the x-value of an A-map event is the x-coordinate of the right endpoint of s – and the array A must be updated by replacing it with the edge s'. Here s' is the next non-vertical edge along the boundary polyline that s is part of, where $s' = \textsc{Nil}$ if s is the last non-vertical edge of the boundary polyline. The value a indicates that the edge s is on the boundary polyline between A^a and A^{a+1}. In other words s (and s', if it exist) are the a-th intersection point, $0 \leqslant a < n^*$, with the A-map along the current sweep line, and so we must update the entry $A[a]$ by setting $A[a] \leftarrow s'$.

B-map events, denoted by $e_B(v_n)$: At a B-map event, a horizontal edge of the B-map ends. This happens when the sweep line reaches a voter v_n – that is, when $x_0 = x_n$ – and so the x-value of this event is v_n. The bottommost intersection of he sweep line with the B-map now disappears (see Figure 1b), and so we must update B by shifting all other intersection points one position down in B and setting $B[n^* - n] \leftarrow \textsc{Nil}$.

Subproblem events, denoted by $e_X(n', k', \ell', \gamma', \delta')$: At a subproblem event the solution to the subproblem given by $I' = \langle n', k', \ell', \gamma', \delta'\rangle$ is known and the x-value of this event is equal to $X_{\max}(I')$. Handling the subproblem event for I' entails deciding which later subproblems I can use I' in their solution and how they can use it, using the sets $\Delta(\tau, a, b)$, and updating the solutions to these subproblems.

In the beginning of the algorithm all the events associated with elementary subproblems are known. The events associated with non-elementary subproblems are added to the event queue when handling an update event $e_E(v_n)$, as discussed next.

Update events, denoted by $e_E(v_n)$: At the update event happening at x-value v_n, all subproblem events of size n are added to the event queue E. These are simply the subproblems $\langle n, k, \ell, \gamma, \delta\rangle$ for all $k, \ell, \gamma \in \{0, \ldots, n\}$ and $\delta \in \{\text{strict}, \text{loose}\}$. The reason we could not add them at the start of the algorithm was that the x-value of such a subproblem I was now known yet. However, when we reach v_n then $X_{\max}(I)$ is determined, so we can add the event to E with $X_{\max}(I)$ as its x-value.

The pseudocode below summarizes the algorithm.

Algorithm 1 ComputeSolutions(τ, V, k^*, ℓ^*).

```
1  for i ← 0 to n* − 1 do
2      A[i] ← (v_i, v_{i+1}) − (v_{i+1}, v_{i+1});   B[i] ← v_{i+1}        ▷ define v_0 := v_1 − 1
3  A[n*] ← Nil;   B[n*] ← Nil
4  Initialize X by the solutions to elementary subproblems
5  Initialize E by all map events, update events, and elementary subproblem events
6  while E is not empty do
7      e ← extractMin(E);   x_0 ← x-value of e
8      switch e do
9          case e_A(a, s, s') do
10             A[a] ← s'
11         case e_E(v_n) do
12             B[n* − n] ← Nil
13             for i ← 0 to n* − n − 1 do
14                 B[i] ← v_{n+i+1}
15         case e_X(n', k', ℓ', γ', δ') do
16             for all span(x_0, n, a, b) = y where y ≠ −∞ do
17                 for all (δ', j, δ) ∈ Δ(τ, a, b) do
18                     I ← ⟨n, k' + 1, ℓ' + j, γ' + n − n' − gain^j(a, b), δ⟩
19                     X(I) ← max(X(I), y)
20         case e_E(v_n) do
21             Add all the events for subproblems of size n to E, as explained above
```

▶ **Lemma 10.** *Algorithm 1 correctly computes the solutions for subproblems* $\langle n, k, \ell, \gamma, \delta\rangle$ *for the given value* τ*, for all* $n, k, \ell, \gamma, \delta$ *with* $0 \leqslant n \leqslant n^*$*, and* $0 \leqslant k, \ell, \gamma \leqslant n$*, and* $\delta \in \{\text{strict}, \text{loose}\}$*, and* $\ell < 2(k+1)$*. The running time of the algorithm is* $O(k^*\ell^*(n^*)^3)$*.*

Proof. We handle the A-map and B-map events before a subproblem event so that A and B data structures are up-to-date when we want to compute the span function on handling a subproblem event. We also handle a subproblem event before an update event so that when we want to add a new subproblem event to the event queue on handling an update event, its entry in table X has the correct value. The correctness of the algorithm now follows from the discussion and lemmas above.

The running time is dominated by the handling of the subproblem events. By Observation 9, the algorithm handles each subproblem in $O(n^*)$ time, plus $O(\log n^*)$ for operations on the event queue, and there are $O(k^*\ell^*(n^*)^2)$ subproblems. Hence, the total running time is $O(k^*\ell^*(n^*)^3)$. ◀

By Lemmas 5 and 10, the algorithm described at the beginning of Section 2.2 computes $\Gamma_{k^*,\ell^*}(V)$ correctly. Since this algorithm calls ComputeSolutions $\lfloor n^*/\ell^* \rfloor$ times in Step 1, we obtain the following theorem.

▶ **Theorem 11.** *There exists an algorithm that computes* $\Gamma_{k^*,\ell^*}(V)$*, and thus solves the one-dimensional case of the one-round discrete Voronoi game, in time* $O(k^*(n^*)^4)$*.*

▶ Remark. We can also solve the one-dimensional case of the one-round discrete Voronoi game when voters are weighted, i.e. each voter $v \in V$ has an associated weight $\omega(v)$ and the players try to maximize the total weight of the voters they win. In this case, the α-gain and β-gain of an interval is defined as the total weight of voters the second player can win in that interval by placing one point and two points, respectively. The number of possible thresholds is not an integer in range $[0, n^*]$, but the sum of any sequence of consecutive voters define a threshold, which makes a total of $O((n^*)^2)$ different thresholds. The gain map also becomes more complex and in the algorithm we need to spend $O((n^*)^2)$ time (instead of $O(n^*)$) to handle each subproblem event, which results in an algorithm with running time $O(k^*\ell^*(n^*)^5)$.

3 Containment in $\exists\forall\mathbb{R}$ and the Algorithm for $d \geqslant 2$

We now consider the one-round discrete Voronoi game in the L_p-norm, for some arbitrary p. Then a strategy $P = \{p_1, \ldots, p_k\}$ can win a voter $v \in V$ against a strategy $Q = \{q_1, \ldots, q_\ell\}$ if and only if the following Boolean expression is satisfied:

$$win(v) \coloneqq \bigvee_{i\in[k]} \bigwedge_{j\in[\ell]} (\mathrm{dist}_p(p_i, v))^p \leqslant (\mathrm{dist}_p(q_j, v))^p,$$

where dist_p is the L_p-distance. This expression has $k\ell$ polynomial inequalities of degree p. The strategy P is winning if and only if the majority of the expressions $win(v_1), \ldots, win(v_n)$ are true. Having a majority function *Majority* that evaluates to true if at least half of its parameters evaluates to true, player $\mathcal{P}$ has a winning strategy if and only if

$$\begin{aligned}&\exists x_1(p_1), \ldots, x_d(p_1), \ldots, x_1(p_k), \ldots, x_d(p_k)\\ &\qquad \forall x_1(q_1), \ldots, x_d(q_1), \ldots, x_1(q_\ell), \ldots, x_d(q_\ell) : \quad Majority(win(v_1), \ldots, win(v_n))\end{aligned}$$

is true, where $x_i(\cdot)$ denotes the i-th coordinate of a point.

Ajtai et al. [2] show that it is possible to construct a sorting network, often called the AKS sorting network, composed of comparison units configured in $c \cdot \log n$ levels, where c is a constant and each level contains exactly $\lfloor n/2 \rfloor$ comparison units. Each comparison unit takes two numbers as input and outputs its input numbers in sorted order. Each output of a comparison unit (except those on the last level) feeds into exactly one input of a comparison unit in the next level, and the input numbers are fed to the inputs of the first level. The outputs of the comparison units in the last level (i.e., the outputs of the network) give the numbers in sorted order.

Using AKS sorting networks we can construct a Boolean formula of size $O(n^c)$ for some constant c that tests if the majority of its n inputs are true as follows. Assuming the boolean value *false* is smaller than the boolean value *true* value, we make an AKS sorting network that sorts n boolean values. This is possible using comparison units that get p and q as input, and output $p \wedge q$ and $p \vee q$. It is not hard to verify that the $\lceil n/2 \rceil$-th output of the network is equal to the value of the majority function on the input boolean values. By construction, we can write the Boolean formula representing the value of this output as *logical and* ($\wedge$) and *logical or* ($\vee$) combination of the input boolean values, and the size of the resulting formula is $O(n^c)$.

Thus we can write $Majority(win(v_1), \ldots, win(v_n))$ as a Boolean combination of $O(n^c k\ell)$ polynomial inequalities of degree p, where each quantified block has kd and ℓd variables respectively. Basu et al. [7] gave an efficient algorithm for deciding the truth of quantified

formulas. For our formula this gives an algorithm with $O((n^c k\ell)^{(kd+1)(\ell d+1)} p^{k\ell d^2})$ running time to decide if $\mathcal{P}$ has a winning strategy for a given instance $\langle V, k, \ell\rangle$ of the Voronoi game problem. Note that this is polynomial when k, ℓ and d are constants.

For the L_∞ norm, we can define $F(v)$ as follows:

$$F(v) := \bigvee_{i\in[k]} \bigwedge_{j\in[\ell]} \bigvee_{s'\in[d]} \bigwedge_{s\in[d]} |x_s(p_i) - x_s(v)| \leqslant |x_{s'}(q_j) - x_{s'}(v)|,$$

By comparing the squared values instead of the absolute values, we have a formula which demonstrates that even with the L_∞ norm, the problem is contained in $\exists\forall\mathbb{R}$ and there exists an algorithm of complexity $O((n^c k\ell d^2)^{(kd+1)(\ell d+1)} 2^{k\ell d^2})$ to solve it.

▶ **Theorem 12.** *The one-round discrete Voronoi game $\langle V, k, \ell\rangle$ in $\mathbb{R}^d$ with the L_p norm is contained in $\exists\forall\mathbb{R}$. Moreover, for fixed k, ℓ, d there exists an algorithm that solves it in polynomial time.*

De Berg et al. [9] introduced the notion of personalized preferences. More precisely, given a natural number p, assuming each axis defines an aspect of the subject voters are voting for, the voter v_i gives different weights to different axes, and v_i has a weighted L_p distance $(\sum_{j\in[d]} w_{ij}(x_j(p) - x_j(v_i))^p)^{1/p}$ from any point $p \in \mathbb{R}^d$. For the weighted L_∞ distance, v_i is at distance $\max_{j\in[d]}(w_{ij}|x_j(p) - x_j(v_i)|)$ from any point $p \in \mathbb{R}^d$. This approach also works when voters have personalized preferences.

4 Σ_2^P-Hardness for $d \geqslant 2$

In this section, we present the most important ideas behind our proof that the one-round discrete Voronoi game is Σ_2^P-hard in $\mathbb{R}^2$. To prove Σ_2^P-hardness, it suffices to show that deciding if $\mathcal{Q}$ has a winning strategy against every possible strategy of $\mathcal{P}$ is Π_2^P-hard. Our proof will use a reduction from a special case of the quantified Boolean formula problem (QBF), as defined next. Let $S := \{s_1, \ldots, s_{n_s}\}$ and $T := \{t_1, \ldots, t_{n_t}\}$ be two sets of variables, and let $\bar{S} := \{\bar{s}_1, \ldots, \bar{s}_{n_s}\}$ and $\bar{T} := \{\bar{t}_1, \ldots, \bar{t}_{n_t}\}$ denote their negations. We consider Boolean formulas B of the form

$$B \quad := \quad \forall s_1, \ldots, s_{n_s} \exists t_1, \ldots, t_{n_t} : c_1 \wedge \cdots \wedge c_{n_c}$$

where each clause c_i in $C := \{c_1, \ldots, c_{n_c}\}$ is a disjunctive combination of at most three literals from $S \cup \bar{S} \cup T \cup \bar{T}$. Deciding if a formula of this form is true is a Π_2^P-complete problem [17].

Consider the undirected graph $G_B := (N, A)$ representing B, where $N := S \cup T \cup C$ is the set of nodes of G_B and $A := \{(c_i, s_j) : s_j \in c_i \vee \bar{s}_j \in c_i\} \cup \{(c_i, t_j) : t_j \in c_i \vee \bar{t}_j \in c_i\}$ is the set of edges of G_B. Lichtenstein [12] showed how to transform an instance of QBF in polynomial time to an equivalent one whose corresponding graph is planar (and of quadratic size). Thus we may start our reduction from a formula B such that G_B is planar. Our reduction then creates an instance $\langle V, k, \ell\rangle$ of the Voronoi game such that B is true if and only if $\mathcal{Q}$ has a winning strategy.

Define D_i, the *disk of* v_i with respect to a given set P, as the disk with center v_i and radius $\mathrm{dist}(v_i, P)$, that is, D_i is the largest disk centered at v_i that has no point from P in its interior.

▶ **Observation 13.** *$\mathcal{Q}$ wins a voter v_i against P iff she places a point q in the interior of D_i.*

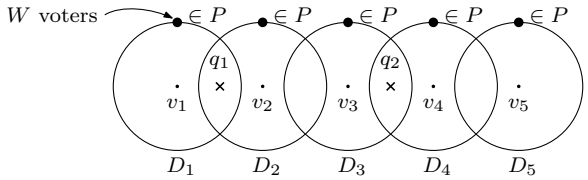

Figure 2 When all the heavy-weight clusters of W voters are chosen by $\mathcal{P}$, the best strategy of $\mathcal{Q}$ to win the remaining single voters is to put her points in every other intersection of the disks.

The idea of the construction is that a cluster of W coinciding voters, for a sufficiently large W, forces $\mathcal{P}$ to put a point on top of that cluster. The disk D_i of a voter v_i is then prescribed by cluster closest to v_i. This allows us to create gadgets for the variables s_i, and clause gadgets, consisting of (sets of) disks. Because the graph G_B is planar, we can carry information from the variable gadgets to the clause gadgets using non-crossing chains of disks. This is done in such a way that $\mathcal{Q}$ must either place points in the "even-numbered" intersections or in the "odd-numbered" intersections in a chain, corresponding to the truth settings of the variables t_j; see Figure 2. An optimal choice of $\mathcal{Q}$ will also carry the bits so that the clauses of B can be checked. The detailed construction is described in the full version.

5 Concluding Remarks

We presented the first polynomial-time algorithm for the one-round discrete Voronoi game in $\mathbb{R}^1$. The algorithm is quite intricate, and it would be interesting to see if a simpler (and perhaps faster) algorithm is possible. Finding a lower bound for the 1-dimensional case is also open.

We also showed that the problem is Σ_2^P-hard in $\mathbb{R}^2$. Fekete and Meijer [11] conjectured that finding an optimal strategy for the multi-round continuous version of the Voronoi game is PSPACE-complete. We conjecture that in the multi-round version of the discrete version, finding an optimal strategy is PSPACE-hard as well. Note that using the algebraic method presented in this paper, it is easy to show that this problem is contained in PSPACE. While the algebraic method we used is considered a standard technique, it is, as far as we know, the first time this method is combined with polynomial-size boolean formulas for the majority function. We think it should be possible to apply this combination to other problems as well.

References

1 Hee-Kap Ahn, Siu-Wing Cheng, Otfried Cheong, Mordecai Golin, and Rene Van Oostrum. Competitive facility location: the Voronoi game. *Theoretical Computer Science*, 310(1-3):457–467, 2004.

2 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

3 Aritra Banik, Bhaswar B Bhattacharya, and Sandip Das. Optimal strategies for the one-round discrete Voronoi game on a line. *Journal of Combinatorial Optimization*, 26(4):655–669, 2013.

4 Aritra Banik, Bhaswar B Bhattacharya, Sandip Das, and Sreeja Das. Two-round discrete Voronoi Game along a line. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 210–220. Springer, 2013.

5 Aritra Banik, Bhaswar B Bhattacharya, Sandip Das, and Satyaki Mukherjee. The discrete Voronoi game in R2. *Computational Geometry*, 63:53–62, 2017.

6 Aritra Banik, Jean-Lou De Carufel, Anil Maheshwari, and Michiel Smid. Discrete Voronoi games and epsilon-nets, in two and three dimensions. *Computational Geometry*, 55:41–58, 2016.

7 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM*, 43(6):1002–1045, 1996.

8 Otfried Cheong, Sariel Har-Peled, Nathan Linial, and Jiří Matoušek. The One-Round Voronoi Game. *Discrete & Computational Geometry*, 31(1):125–138, 2004.

9 Mark de Berg, Joachim Gudmundsson, and Mehran Mehr. Faster algorithms for computing plurality points. *ACM Transactions on Algorithms*, 14(3):36:1–36:23, 2018.

10 Michael G Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzążewski. $\forall\exists\mathbb{R}$-Completeness and Area-Universality. In *Proceedings of the 44th International Graph-Theoretic Concepts in Computer Science (WG 2018)*, pages 164–175, 2018.

11 Sándor P Fekete and Henk Meijer. The one-round Voronoi game replayed. *Computational Geometry*, 30(2):81–94, 2005.

12 David Lichtenstein. Planar formulas and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

13 Wei-Yin Lin, Yen-Wei Wu, Hung-Lung Wang, and Kun-Mao Chao. Forming Plurality at Minimum Cost. In *Proceedings of the 9th International Workshop on Algorithms and Computation*, pages 77–88, 2015.

14 Richard D McKelvey and Richard E Wendell. Voting equilibria in multidimensional choice spaces. *Mathematics of operations research*, 1(2):144–158, 1976.

15 Michael Paterson and Uri Zwick. Shallow circuits and concise formulae for multiple addition and multiplication. *Computational Complexity*, 3(3):262–291, 1993.

16 Joachim Spoerhase and H-C Wirth. (r, p)-centroid problems on paths and trees. *Theoretical Computer Science*, 410(47-49):5128–5137, 2009.

17 Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

18 Sachio Teramoto, Erik D Demaine, and Ryuhei Uehara. The Voronoi game on graphs and its complexity. *Journal of Graph Algorithms and Applications*, 15(4):485–501, 2011.

19 Yen-Wei Wu, Wei-Yin Lin, Hung-Lung Wang, and Kun-Mao Chao. Computing plurality points and Condorcet points in Euclidean space. In *International Symposium on Algorithms and Computation*, pages 688–698, 2013.