Research in design of percentile mechanisms for facility location problems in multidimensional space

Alex Francois Nienaber
Department of Computer Science
University of Toronto

August 31, 2012

Introduction

The importance of decision making as a driving force behind socio-economic mechanics makes it the focus of many disciplines (e.g. Public Choice theory, Decision theory, Game theory, etc.). In a social environment, the importance of decision making is all the more visible; and implementing a fair way of catering to everyone's preferences can be nontrivial in situations with intelligent individuals who have the capacity to strategize.

We will start by covering historical results in the domains of Social Choice theory and Voting theory, providing a motivational background for the field of Algorithmic Mechanism Design. Links with related domains such as Game Theory and Mechanism Design will be exposed, and explored concepts will eventually build up to the current research in Algorithmic Mechanism Design.

We then shift our perspective to a more specific approach undertaken recently, which is the design of percentile mechanism for facility location problems. At this point, Computer Science considerations and methodologies are introduced to complement an explanation of the algorithms used to optimize these mechanisms. These are then tested on various samples of artificially produced data.

Finally, the focus is relayed to applying these mechanisms to real world data. Several approximation methods and heuristics are presented, in order to be used and combined to form algorithms to process the actual data. These are then tested on Irish election data.

We conclude on the practical significance of this research, emphasizing the gap between other more abstract approaches that end up lacking in realistic pertinence.

1 Overview of Social Choice theory

Social choice theory is the field of study* concerned with aggregating individual preferences to form a collective decision. It stems from Kenneth Arrow's *Social Choice and Individual Values* (1951) in which he laid the ground for the field by defining some of its core concepts such as the general impossibility theorem.

1.1 Arrow's Impossibility theorem

Looking for a 'fair' and wholesome voting system lead Arrow to prove that such a result was impossible. His axiomatic approach to voting theory defines a 'fair' voting system to be one where the following rules hold:

non-dictatorship	all individual preferences are assessed the same way, no emphasis is given to an individual over another
universality	the system should provide a unique result for any combination of preferences in a deterministic manner
I.I.A.**	when two alternatives are ranked the same way by two voters, the resulting preference should be consistent with that ordering; in other words, the way the other alternatives are ranked by the two voters should not affect the resulting relative positions of these two alternatives
unanimity	if an outcome is preferred by all participants over another, the resulting ranking should maintain that order

The theorem states that there is no voting system that can satisfy those conditions when the number of alternatives presented to the participants exceeds two. It is interesting to see that the implications of this founding result are representative of the field; i.e. in a lot of problems addressed by social choice theorists, there is no 'perfect' system, thus the focus is delegated to finding a 'good' system.

*Amartya Sen exposes the fact that social choice theory can be viewed as both a field of study as well as *"a particular approach or a collection of approaches typically used in that field of study"*, qualifying the latter definition as narrower [1]

**Independence of Irrelevant Alternatives

1.2 Problems in Social Choice

Arrow's axiomatic method has since been the de facto approach in Social Choice. In Moulin's *Axioms of Cooperative Decision Making* (1988), the author relies on the systematic approach to reduce the set of solutions to several problems in social choice:

"Of course, each microeconomic problem that we will examine has more than one plausible solution: There are many 'good' voting rules, several plausible 'values' for cooperative games [...] and so on. Ideally, the axiomatic method can help our choice by, first, reducing the number of plausible solutions as much as possible" [2]

The spectrum of problems addressed in Social Choice is pretty broad (elections, auctions, etc.). A common problem, and the focus of this paper, is facility location.

In its most general form, a facility location problem has the following characteristics:

setting	a set of individuals with preferences regarding the position of facilities, a set of facilities, and miscellaneous constraints on the space on which facilities have to be placed
outcome	the location of the facility(ies)

However, in most facility location problems the outcome takes into consideration various requirements such as minimizing social cost (in the described setting, the cost of a single participant is her distance to the closest facility; the social cost is the sum of the all participants' costs), minimizing the maximum distance (the highest individual cost) or minimizing the maximum load (in the described setting, the load of a facility is the number of participants for which that facility is the closest).

It is interesting to note that several social choice problems, when extended to a multidimensional space, can be viewed as facility location problems:

"many social choice problems can be interpreted as "facility location" problems when viewed as choice in a higher dimensional space, such as selection of political/committee representatives, product design, and the like" [3]

2 Algorithmic Mechanism Design

Game theory is the field of study that focuses on strategic decision making, mainly in a socio-economic context. Nobel prize co-winner R.B. Myerson describes it as *"the study of mathematical models of conflict and cooperation between intelligent rational decision-makers. Game theory provides general mathematical techniques for analyzing situations in which two or more individuals make decisions that will influence one another's welfare"* [4]. The name of the field refers to mathematical objects known as games, which formalize interactions between participants (agents).

Mechanism design is the branch of game theory which considers the solutions to a certain class of games, that have the following characteristics:

1. they involve multiple self-interested agents, each with private information (their preferences)
2. the game designer has control over the structure of the game
3. the game designer is focused on the outcome of the game

In algorithmic mechanism design, the focus is partially related to the implementation of those solutions, thus it inherits from mechanism design and computer science. The name (algorithmic mechanism design) was first coined by Noam Nisan, who describes mechanism design as *"rather unique within economics in having an engineering perspective"* [5]. Indeed, the field is concerned with both economic aspects (e.g. social welfare, individual incentives to strategize, etc.) and computational aspects (e.g. running time, approximation ratios, etc.) of the solutions.

2.1 The Social Choice function

Consider the following setting: a set of n agents I, a set of p alternatives A, and a set of preferences over the alternatives L. We consider the simplest definition of 'preferences', and view them as linear rankings of the alternatives: ' $<$ ' is a total order over the alternatives (for every couple (a, b) of alternatives in A, there exists an ordering '$a < b$' (b is preferred to a) or '$b < a$' (a is preferred to b) denoting preference for one over the other).

Definition: a <u>social welfare function</u> is a function that aggregates the preferences of all participants into a single preference ordering over the set of alternatives

$$F: L^n \rightarrow L$$

Definition: a <u>social choice function</u> is a function that aggregates the preferences of all participants into a single alternative

$$F: L^n \rightarrow A$$

The more general version of Arrow's theorem states that any social welfare function where card(A) › 2, that satisfies unanimity and I.I.A., is a dictatorship (i.e. there exists a voter i such that regardless of the preferences of other voters, societal preference will be given to his ordering; in other words, for all n-tuples in L^n, for any two alternatives a and b, if voter i prefers a over b, then the resulting preference ordering will order a over b).

2.2 Incentive Compatibility

If the agents are aware of the procedure that aggregates their preferences (which is often the case in public decision making), it is possible for them to report false preferences that might change the outcome in their favour. Taking this into account, we define the concept of incentive compatibility, which is that the procedure cannot be strategically manipulated by reporting false preferences; that is: every agent is at least as well off by reporting his true preferences.

Formally, a social choice function f is incentive compatible if and only if no agent with true preference $<_i$, can strategically manipulate it by reporting $<_{i'}$ and change the outcome from a to b where b $_i>$ a. This constraint is equivalent to the monotonicity of the function:

f is monotone <=> f $(< , <_i)$ = a **and** f $(< , <_{i'})$ = b **then** b $<_i$ a **and** a $<_{i'}$ b
<=> **f is incentive compatible**

Where $(< , <_i)$ *is an n-tuple in the domain of definition of f, such that voter* ***i*** *reports a preference ranking* $<_i$ *, and* $(< , <_{i'})$ *is an n-tuple in the domain of definition of f, such that voter* ***i*** *reports a preference ranking* $<_{i'}$*.*

The Gibbard-Satterthwaite theorem is the counterpart of Arrow's theorem for social choice functions, it states that any incentive compatible social choice function where card(A) › 2 and where for every alternative in A, there is a antecedent by the function in I (the function is onto, i.e. every alternative can be reached by a certain n-tuple of preferences) is a dictatorship.

2.3 Mechanisms

At this point, it seems clear that our mechanisms will have to solve the dual problem of preference aggregation as well as information revelation.

In its most primitive form, a mechanism is a collection S of strategy sets (corresponding to the space of all possible actions that can be undertaken by a given agent) and an outcome function mapping n-tuples in S to an alternative in A (where is the set of alternatives). There is a conceptual distinction to be made between types and strategies; a type is an agent's own personal valuation of the alternatives, whether a strategy is an induced course of action an agent can take relating to the situation.

Definition: A mechanism M = $((S_i)_{i\in N}, g)$ is a couple of S, the Cartesian product of strategy sets S_i for each agents $i \in N$, and an outcome function $g: S \rightarrow A$.

The previous definition is very general; in a social choice setting, let us consider a special case of mechanism, where the collection S is equal to the set of agents' actual types (in our setting: preferences) and the outcome function is equal to the social choice function of the setting. This type of mechanism, in which agents reveal their types (truthfully or not) and the outcome procedure is the social choice function, is called a direct revelation mechanism, while all other mechanisms are indirect mechanisms.

Definition: A direct revelation mechanism D = $((\Theta_i)_{i\in N}, f)$ is a couple of Θ, the Cartesian product of type sets Θ_i for each agents $i \in N$, and a social choice function $f: \Theta \rightarrow A$.

To further understand mechanisms, we need to take a step back and look at them from a game theoretic perspective. In game theory, agents playing a game have sets of strategies at their disposal. A pure strategy is a deterministic strategy, whether a mixed strategy is an assignment of probabilities to a set of pure strategies. The equilibrium of a game arises when all the players are playing strategies which are best responses to each other (i.e. no agent has the incentive to change his strategy if the remaining players don't). A good explanation of the correlation between mechanisms and games is given in the paper *Foundations of Mechanism Design, A Tutorial*:

"In view of the definition of the indirect mechanism and direct revelation mechanism, we can say that a social planner can either use an indirect mechanism M or a direct mechanism D to elicit the information about the agents' preferences in an indirect or a direct manner, respectively. As we assumed earlier, all the agents are rational and intelligent. Therefore, after knowing about the mechanism M = $((S_i)_{i\in N}, g)$ chosen by the social planner, each agent i starts doing an analysis regarding which action s_i will result in his most favourable outcome [...] This phenomenon leads to a [...] Bayesian game of incomplete information that gets induced among the agents when the social planner invokes this mechanism as a means to solve the information elicitation problem." [6]

Thus, a mechanism will implement a social choice function if it matches the outcome of that induced game (that is the equilibrium strategies of each agent) to the result of the social choice function.

Definition: A mechanism M = $((S_i)_{i\in N}, g)$ implements a social choice function f if the game induced by M has a pure strategy equilibrium s* (the set of agents' strategies at equilibrium), and if the outcome of f is the same as the outcome of g for that equilibrium s*.

2.4 The Revelation Principle

This core result in mechanism design was first introduced in the context of dominant strategy equilibrium by Allan Gibbard in his 1973 paper *Manipulation of voting schemes: a general result* while discussing what would be referred to as the Gibbard-Satterthwaite theorem. Myerson, among others, later extended it to Bayesian Nash equilibria, however in order to avoid too many definitions, we choose to focus on its definition in the setting of dominant strategy equilibrium. A <u>dominant strategy</u> for agent i is the best possible strategy agent i can play regardless of what other players do. A <u>dominant strategy equilibrium</u> is a special case of pure strategy equilibrium for which the equilibrium strategies of the agents are dominant strategies.

Definition: A mechanism $M = ((S_i)_{i \in N}, g)$ implements a social choice function f in dominant strategies if the game induced by M has a dominant strategy equilibrium s^d (the set of agents' dominant strategies at equilibrium), and if the outcome of f is the same as the outcome of g for that equilibrium s^d.

The revelation principle for dominant strategy equilibrium states the following: if there is a mechanism M that implements a social choice function f in dominant strategies, then f is truthfully implementable in dominant strategy equilibrium by a direct mechanism.

Definition: A social choice function f is truthfully implementable in dominant strategies, if the direct revelation mechanism $D = ((\Theta_i)_{i \in N}, f)$ induces a game with a dominant strategy equilibrium s^d, in which $\forall i \in N, \forall \theta_i \in \Theta_i, s_i^d(\theta_i) = \theta_i$.

Where $s_i^d(\theta_i)$ is agent i's strategy in the equilibrium and θ_i a type in agent i's type space.

In other words, for the case of dominant strategy equilibrium, the revelation principle states that the set of all social choice functions implementable by direct mechanisms is equal to the set of all social choice functions implementable by indirect mechanisms.

The result is of major importance to the field, since it narrows the set of all possible mechanisms that would implement a certain social choice function to the set of direct revelation mechanisms that would implement it. Hence, if there is no direct revelation mechanism that can implement a social choice function, then that function is not truthfully implementable.

3 Mechanisms for Facility Location Problems

Now that we laid the ground with some important concepts of Social choice theory and Algorithmic mechanism design, we will examine the design of mechanisms for facility location problems.

3.1 Single-peaked preferences

First off, since we want a non-dictatorial strategy-proof (incentive compatible) mechanism, we need to work around the Gibbard-Satterthwaite theorem by restricting the domain of agents' types. Interestingly, outside the scope of the theorem, *"whenever there are at most two agents and three alternatives any preference profile is single-peaked"* [7]. Thus, there is certain motivation to want to retain that property when there are more than 2 agents and 3 alternatives.

Single-peaked preferences were popularized by Duncan Black in *"On the rationale of group decision-making"* (1948), where the author notes that in 'some important practical problems' agents' preferences will take the form of single-peaked. A preference profile is single-peaked when there is a linear ordering of the alternatives that is compatible with all the agents' preferences, i.e. it is possible to order the alternatives on a line in such a way that, for any agent, every alternative to the left of his most preferred alternative is ordered consistently with his preferences (that is they are ordered from least preferred to most preferred), and same goes to every alternative to the right (that is they are ordered from most preferred to least preferred). That most preferred alternative is referred to as the peak.

By restricting the domain of preferences of the agents in our setting, we are able to circumvent the Gibbard-Satterthwaite theorem ('intransitivities' or 'cycles' in preference ordering cannot be realised in single-peaked profiles) [8]. To complement this choice with an intuitive justification, consider an agent's preferred location for a facility, it seems coherent that the further away the actual location of the facility is, the less satisfied the agent will be.

3.2 Median Mechanisms

Recall the facility location problem setting described earlier. We will consider a simple 1 dimensional scenario, where a single facility has to be placed on a line. The game designer has to ask agents to reveal their most preferred location for the facility (in practice, it will most often be their own location). He announces that the procedure for aggregating preferences will take the average of their preferences. Also, consider that the agents have some information on other agents' locations. Is that procedure strategy-proof? No, since it is possible for an agent to manipulate the average by reporting an untruthful location, and thus bias the location of the facility in his favour.

To achieve truthfulness, we will have to use a median mechanism, since *"every onto strategy-proof social choice function on a single-peaked domain is a generalized median voter scheme"* [9]. In the previous example, a procedure that would aggregate agents' preferences by taking the median would be a simple example of a median mechanism: no agent has the incentive to misreport his preference, because the only way for a single agent to manipulate the median would lead to the median being further away from the agent's actual preferred location. Interestingly, the mechanism is also group strategy-proof, but we won't dwell too long on this notion; suffice to say that no coalition of agents has the incentive to report a set of preferences that will produce an outcome such that every agent in the group will be better off.

3.2 Percentile Mechanisms

There are multiple ways to extend such mechanisms to additional dimensions and facilities [8]. In 2009, Ariel D. Procaccia and Moshe Tennenholtz came up with approximation mechanisms that remained group strategy-proof while providing closer to optimal results.

For instance, the median mechanism in the above example gives an approximation ratio of 2 for the maximum load. Approximation ratios are ratios of the worst case result in the current setting (in the example: consider a case where the median is as far away from the average as possible) by the optimal result (in the example: the average). Since no better approximation ratio can be achieved by deterministic mechanisms [10], Procaccia and Tennenholtz introduce, for this basic setting, the following randomized mechanism: the mechanism will return the location of the left-most agent with probability 1/4, the location of the right-most agent with probability 1/4, and the average location with probability 1/2. The mechanism is still (group) strategy-proof, since it partially relies on the first and last statistics of the space of agents' locations (when ordered on the line, the left-most location is the first order statistic, and the right-most is the last); however the approximation ratio is now:

$$[1/4 * d + 1/2 * d/2 + 1/4 * d] \,/\, d = 3/2$$

Where d is the distance between the left-most and right-most location in the space of agents' locations.

More recently [3], Xin Sui, Craig Boutilier and Tuomas Sandholm came up with another form of generalized median mechanisms, percentile mechanisms. They too rely on selecting certain statistics in order to retain group strategy-proofness even when extended to multi-dimensional and multi-facility scenarios. Consider all agents' locations ordered on a dimension. The mechanism will return percentiles (corresponding to the position of an agent on a given dimension) for every facility, per dimension. Back to our example, the simple median mechanism in the example is viewed as a (0.5)-percentile mechanism since it returns the 50th percentile (i.e. the median).

Finally, there are still more ways to view/extend the acclaimed class of generalized median mechanisms (e.g. in Moulin's *On strategy-proofness and single peakedness*, 1980). It is interesting to note that these generalizations each provide very good results (in terms of approximation, computational time, etc.) for different considerations and types of facility location problems, some giving extremely good results for a very narrow range of problems while others may give slightly less good results, but for a broader class of problems.

3.3 Optimization of Percentile Mechanisms

We choose to focus on percentile mechanisms because of their practical superiority, in certain scenarios, over other kinds of generalized median mechanisms. Indeed, we can empirically derive percentiles from 'real world' data (instead of getting them through worst-case analysis of each scenario) and produce mechanisms that will provide far better results, especially in multi-dimensional and multi-facility settings. To make this class of mechanisms so adaptable, we provide several algorithms to derive the optimal percentiles of such mechanisms from sample data.

Every algorithm takes a set of sample profiles corresponding to individuals preferences, a number of dimensions to be considered (the samples should provide consistent locations for the agents, i.e. they should give the location of every agent on every dimension considered), the number of facilities to be placed, and the distance metric, which can be Euclidean or Manhattan (again, since those are the most common in practice). The algorithms are tested, at first, on artificially produced samples, derived from a particular distribution that is observable in practice (e.g. a city's population geographical organisation might closely resemble a Gaussian distribution with several clusters).

The first algorithm was developed by Xin Sui and takes the following approach: it goes through every possible combination of percentiles at a given level of discretization (i.e. we define the notion of a step as how much 'space' there is between a considered percentile and the next, so for example, with a step of 20, the percentiles considered are the first, the 20th, 40th, 60th, 80th and last) finding the best combination to minimize social cost, then reiterates the same process but with a smaller step and in the bounds defined in the previous process (e.g. if the 40th percentile was found to be the best position of a facility for a given dimension with a step of 20, the next iteration will go through percentiles 20 to 60 with a smaller step); so on until the desired precision. This algorithm has been referred to as the abstraction method, and takes exponential time to compute the optimal percentiles for the given scenario.

It was first conjectured that the social cost function of the facilities' locations admitted a single local minimum (that is the mapping by the social cost function of the optimal location of the facilities), and it thus seemed like a logical step to implement faster algorithms that would take advantage of that fact by performing a <u>local search</u> over the function. However it turned out this was not true in theory, but since scenarios in which the social cost function admits several local minima are relatively unlikely to happen in practice, and since the local search algorithms can be run several times from random initial facility locations, there still is a strong incentive to develop these metaheuristics.

3.4 Local Search Algorithms for Optimizing Percentile Mechanisms

There is a multitude of ways of approaching the implementation of the local search perspective to the problem of finding the optimal percentiles for these mechanisms, we will describe 3 different algorithms and conclude on their performance.

A note on the main data structure used, the position matrix. The location of the facilities on every considered dimension are stored in this two dimensional array, the first dimension recording the facilities and the second keeping track of the position of a facility on each dimension.

Every algorithm starts by creating a random position matrix, and ends up with a positional matrix containing 'the close to optimal' percentiles for the location of every facility. They all integrate the (local search) notion of a step, which is the distance between considered percentiles, that will be reduced iteratively until the desired degree of precision is reached. Since they have different ways of getting to the same result, we will be concerned by computational time and precision of the results.

3.4.1 Greedy local search

This algorithm starts with a random position matrix and computes the induced social cost. It then does so for every neighbouring matrix (a neighbouring matrix is one that differs by a single value such that that value [which is the position of a facility on a single dimension] is either a step to the left of the original one [rounded to the first percentile if the computed value is negative] or a step to the right [rounded to the last percentile if the computed value is greater than the total number of agents]). It then moves to that matrix and reiterates this process until reaching a positional matrix that cannot improve for the given step. The step is then reduced and the process reinstantiated.

The name comes from the fact that a greedy method is used while local searching, the best neighbouring matrix is chosen.

3.4.2 Coordinate local search

This algorithm starts with a random position matrix. It then proceeds systematically through every value in the matrix, scanning at first the whole considered dimension according to the step (recall the abstraction method), computing the social cost for every considered percentile, then adopting the best percentile (position) before moving on to the next value in the position matrix (this is very different from the greedy search that computes the social costs of all of its neighbours before updating its position). Once it has traversed the matrix once (and possibly 'moved' several times), it does so again until the position matrix converges for the given step. When reiterating the process for the next step, it doesn't scan the whole dimension again, but a range defined by the previous step (this is similar to the abstraction method).

The name comes from the fact that the whole dimension is searched over instead of just neighbouring positions.

3.4.3 Strict convergence local search

This algorithm starts with a random position matrix. It then takes a slightly different approach than the previous 2 algorithms when traversing the matrix. In fact, it will go through a single facility, and for each of its dimensions, compute the social cost for its current position, for the position a step to its left (rounded to the first percentile if necessary), for the position a step to its right (rounded to the last percentile if necessary), then update to the best position and proceed to the next dimension. However, instead of proceeding to the next facility, it will repeat the same procedure for the current facility until it converges before moving on to the next facility. Once it has traversed the position matrix once, it will repeat the same process until the whole matrix has converged for the given step. Again, the step is then reduced and the procedure reinstantiated.

The name comes from the fact that we not only wait for the whole positional matrix to converge (as in the coordinated local search), but also each facility. This might provide more robustness in higher dimensional or facility spaces. It is also possible to consider a coordinated version of this algorithm.

3.4.4 Results

The algorithms were written in the C++ language, the results are consigned in appendix A. Each was run a 100 times for various settings specified in bold. For instance, Gaussian-2D-3F-L2-101a-100s translates to the setting where test samples were produced using the Gaussian distribution, with 2 dimensions to consider, 3 facilities to place, the distance metric is L2 (Euclidian), 101 agents playing, and the algorithm was fed 100 of the samples. The table presents the minimum social cost found across the 100 tests, the maximum, average and standard deviation of the social cost, as well as the average running time.

First off, every algorithm eventually finds the optimal solution, as presented in the 'minimum' column; note that it is the same found by the abstraction method. The coordinated local search algorithm has a higher 'fail rate' than the others since it boasts a higher maximum and standard deviation than the others, however is the fastest when the number of dimensions or facilities increases. The greedy local search method seems to provide the most consistent results, however becomes slower as time increases. As conjectured, in higher dimensions, the strict convergence algorithm provides a blend of precision and efficiency.

4 Approximate Single-Peaked Preferences

The results of these heuristic algorithms are pretty satisfactory; we can build close to optimal strategy-proof mechanisms in reasonable time assuming we have access to preference data that we can consider single-peaked. However, real world data might not be as friendly as the manufactured samples we used.

Throughout this part, we will be working with actual Irish election data from Dublin West, courtesy of www.dublincountyreturningofficer.com. Our objective is to find an interpretation of the voter's preferences, such that we can consider the data single-peaked consistent.

4.1 Checking Single-Peakedness Consistency in a Profile

The Irish election data from Dublin West contains partial orderings of the 9 candidates to be elected by about 50,000 voters. That is, every voter has ranked at least one of the candidates. We start by extracting a sample of all the total orders over the candidates; that is only the preference rankings of voters who ordered all 9 candidates. This results in a profile of 3800 complete rankings.

We now proceed to find an axis with respect to which the voters in the profile have single-peaked consistent preferences. In their paper *"Single-peaked consistency and its complexity"* [11], Escoffier, Lang and Ozturk describe an algorithm to build such an axis from a profile.

We provide a brief rundown of the algorithm with respect to our implementation: the input is a profile of preference orderings, and the output will be either a consistent axis or a contradiction in the profile that prevents the formation of such an axis. The algorithm builds a partial axis from its endpoints, iteratively placing candidates from conditions imposed by rankings in the profile. At each iteration of the build, the precondition is that all orderings in the profile are consistent with the current partial axis, and the post condition is the same; when contradictory conditions are found, the algorithm terminates. During an iteration, we start by finding the set L of least preferred candidates in the orderings, that have not yet been placed on the partial axis; thus L is the junction of singletons, one per preference ranking, containing, out of the remaining candidates to be placed on the axis, the one ranked last by the considered voter. The candidates in L are then carefully placed on the partial axis: when there is one or two candidates in L, we check if there is an ordering that requires (in order to remain single-peaked consistent with respect to the partial axis) these candidates to be placed before the left endpoint (or any value that has been placed before it) or before the right endpoint. A contradiction arises when two orderings require a candidate to be placed at both sides at the same time, when card(L) › 3 (since at most 1 candidate in L can be placed before one of the 2 endpoints), or when a specific organisation of the candidates is required by an ordering, and that specific organisation cannot be respected by at least one of the rankings.

We run the algorithm on our profile to find out, with not much surprise, that it is not strictly single-peaked consistent. Therefore, we have to turn our attention to methods of single-peakedness approximation in order to find that underlying quality in our profile.

4.2 Definitions of Approximate Single-Peakedness

Multiple ways of approximating single-peaked preferences have been proposed [12], we explain the 'margin of error' that is allowed by a few of these notions for the profile to be approximately single-peaked with respect to it.

k-Maverik	the profile is k-Maverik single-peaked if it is single-peaked consistent by removing at most k orderings (votes) from the profile
k-Candidate Deletion	the profile is k-candidate deletion single-peaked if it is single-peaked consistent by removing at most k candidates from the set of all considered candidates
k-Local Cand. Deletion	the profile is k-local candidate deletion single-peaked if it is single-peaked consistent by removing at most k candidates from each of the orderings in the profile
k-Global Swap	the profile is k-global swap single-peaked if it is single-peaked consistent by performing at most k adjacent swaps (that is swapping two adjacent candidates in a preference ordering) in the whole profile
k-Local Swap	the profile is k-local swap single-peaked if it is single-peaked consistent by performing at most k adjacent swaps in each ordering
k-Additional Axes	the profile is k-additional axes single-peaked if it is possible to partition the voters into at most k clusters, in which all the candidates are single-peaked consistent with respect to a particular axis

In order to find a reasonable method of approximation (perhaps by combining several of these notions), we consider several ways in which our electoral data might have single-peaked qualities. First off, it seems reasonable that voters have different concerns regarding the candidates' policies (e.g. certain voters might rank candidates according to their environmental views, to their policy on taxes, etc.); it is with that respect possible to have several clusters dividing voters by which issue they find the most important. We choose to adopt the k-Additional Axes method in order to build these clusters and find the axis, that might represent a particular ordering of candidates (representative of how they differ on a given socio-political issue). However, those clusters do not purely contain voters with a single issue in mind and there might be some 'noise' in the preferences (e.g. people might not have the same understanding of that issue, etc.), thus we want to account for that margin of error. Since we consider individual orderings, the k-local candidate deletion and k-local swap methods come to mind. Beyond their conceptual difference, they differ by how much freedom in consistent rankings these methods induce: for a fixed axis and a given k, the set of all rankings that can be k-local candidate deletion single-peaked consistent contains the set of all rankings that can be k-local swap single-peaked consistent. Therefore, we choose to focus on the k-local candidate deletion method to filter rankings when constructing those clusters, however we will include exact single-peaked filtering for reference. Finally, since we are going to use a greedy method to build these clusters (the process of building the optimal clusters is NP hard) that might not be optimal and we might still get some high level noise outside these clusters (e.g. voters who chose their candidate ordering for more obscure reasons, voters who ordered candidates in a blend of ways that prevent our 'accounting for noise in the clusters method' to find which underlying feature is the most important for them, etc.), we might also consider only covering 80%-99% of the voters in our profile; hence we choose to keep k-Maverik in mind for some high level filtering of the more peculiar orderings.

4.3 Branch and Bound Algorithm to construct Single-Peaked Axes

Using a similar method of partial axis construction to the one presented earlier [11], we develop a branch and bound algorithm that constructs a tree in which each node inherits a partial axis from its parent, the first parent possessing the null axis.

For a given parent, we build its children nodes by considering every possible combination of candidate placement on the parent's partial axes, with the following restriction: only 2 candidates* can be placed at the 'current endpoints' of the partial axis (where those are the first spots available in the axis before the actual endpoints, so for the first generation, the current endpoints are the actual endpoints of the axis, for the second generation, they are the positions one after the left endpoint and one before the right endpoint, and so on). For each possible child axis, we filter the candidates who were consistent with its parent, by using either exact single-peakedness filtering or k-local candidate deletion filtering; hence at each node we keep track of all the preference orderings that are consistent (for some measure of single-peakdness) with the partial axis defining that node. The number of consistent orderings in a node gives us an upper bound on the number of consistent orderings in the leaf nodes (nodes at the bottom of our tree, i.e. that have complete axes) of its induced subtree.

We use bounding to prevent going into subtrees with an upper bound lesser than our currently defined lower bound. To traverse the least nodes possible while reaching for a leaf with a better lower bound (that is, explore the less axes possible), we use a greedy method for selecting which child is going to be traversed first: children are ranked by upper bound, they will be traversed (or not) from the highest upper bound to the lowest (so, heuristically, the subtree that has the greatest likelihood of containing the leaf with the maximum lower bound will be traversed first). The currently defined lower bound will at first be 1, to allow the algorithm to get to the first leaf (as defined by the greedy method); it will then be updated to that leaf's number of consistent preference rankings (since the axis in a leaf is not partial anymore, that number is now a lower bound on the greatest number of consistent preferences for an axis in our profile).

After traversing the whole tree, branching and bounding, we end up with the biggest cluster of preference rankings that are consistent with a single axis in the profile, its size being the final lower bound remaining.

4.3 Results

Results are confined in appendix B. We first run the algorithm on our data for various settings of k (appendix BI). For k=0, i.e. no approximation (we use exact single-peakedness filtering to create the children nodes), we find out that the biggest cluster contains 0.8% of the preference rankings. Thus, our second level of approximation, k-local candidate deletion filtering, is definitely a must. Note the resulting axis: 1 2 3 4 5 6 7 8 9; this ordering can probably be explained by the fact that the candidates were already ordered by the Dublin West County administration in a certain way.

With k=1, the biggest cluster covers 4.5% of the voters; this is still not enough to use k-additional axes approximation, since we most certainly will end up with more than 50 axes to cover around 80% of the votes (however this is better than with no approximation, where we might have needed over 2000 axes).

With k=2, we can cover 14% of the voters with a single axis. We thus choose to iteratively rerun the algorithm on the remaining candidates (the profile minus the candidates in the cluster obtained at the previous iteration) until we cover around 80% of the voters (this is the greedy method for cluster building we alluded to earlier). The results at each iteration are in section BII of the appendix. It takes 14 clusters to cover as many votes; at this point considering k-additional axes with 14 different axes seems plausible.

With k=3, the biggest cluster covers 32% of the votes. We rerun our algorithm in the same manner as before; the results are assigned to section BIII of the appendix. This time, only 6 axes are necessary to cover over 85% of the votes.

We choose to stop at k=3, because deleting over a third of the candidates in each preference ranking doesn't meet our standards of approximation anymore. The best results we have so far allow us to cover around 80% of the votes using 3 methods of approximation:

(1) The profile is (14-additional axes, 2-local candidate deletion, 777-Maverik) approximately single-peaked.

(2) The profile is (6-additional axes, 3-local candidate deletion, 746-Maverik) approximately single-peaked.

Conclusion

Final words are hard to find when it comes to make conclusions on ongoing research. However, even though we explored only a few combinations of methods of single-peaked approximation (there is yet to find the optimal one, or to provide an algorithm to build the optimal one from sample data), or might have omitted to provide a way of carrying over electoral data for the design of a percentile mechanism, we might have given an appreciation for the empirical research that can be done in the field of Algorithmic Mechanism Design.

Where a lot of research in the domain is constrained to proving abstractly what is possible to be defined/computed in relation to concepts such as generalized median mechanisms or approximate single-peaked preferences, the research we exposed in this paper takes a semi-empirical approach to these notions, which is more prone to see its results applied. From the percentile mechanisms we mentioned, through the local search algorithms to optimize those, to our triple approximation implemented in the branch and bound algorithm; while thinking about constructing optimal ways to resolve the problems at hand, we kept a keen eye on what it is we hope to accomplish.

With this in mind, we can eventually hope to circumvent the famous impossibility theorems (Arrow, Gibbard-Satterthwaite), and develop methods to produce strategy-proof and close to optimal mechanisms for any social choice problem that can be related to a multi-dimensional facility location problem (e.g. elections, auctions, etc.).

Appendix A

Local Search Algorithms - Table of Results

	Gaussian-2D-3F-L2-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	100.213				
Local-greedy	100.213	101.147	100.4968	0.3857244	0.9317
Local-coordinate	100.213	101.94	100.72913	0.488497338	1.0237
Local-strict	100.213	101.07	100.54904	0.391271972	1.0498

	Uniform-2D-3F-L2-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	23.4709				
Local-greedy	23.4709	23.7029	23.506644	0.053286781	0.9046
Local-coordinate	23.4709	23.8311	23.540162	0.079359635	1.0012
Local-strict	23.4709	23.7029	23.521158	0.066762323	1.0094

	GaussianMixture-2D-3F-L2-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	126.034				
Local-greedy	126.034	126.051	126.04046	0.005961865	0.882
Local-coordinate	126.034	133.094	126.31488	0.816948217	1.0084
Local-strict	126.034	126.051	126.03973	0.005361394	0.9657

	Gaussian-4D-2F-L1-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	423.894				
Local-greedy	423.894	432.973	425.11215	2.864318575	1.5331
Local-coordinate	423.894	436.023	426.57497	3.819665902	1.4025
Local-strict	423.894	433.003	424.63979	2.333742861	1.5212

	Uniform-4D-2F-L1-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	86.2768				
Local-greedy	86.2768	87.0444	86.493337	0.21244066	1.4875
Local-coordinate	86.2768	87.3388	86.557049	0.229784903	1.4102
Local-strict	86.2768	86.9824	86.478491	0.196845946	1.5426

	GaussianMixture-4D-2F-L1-101a-100s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	696.558				
Local-greedy	695.558	704.114	695.94496	1.498186964	1.9699
Local-coordinate	695.558	709.406	696.89335	2.320999307	1.7979
Local-strict	695.558	704.516	696.4562	2.356890641	1.878

	GaussianMixture-4D-3F-L2-101a-200s				
	Minimum SC	Maximum SC	Average SC	Std deviation of SC	Average run time
Abstraction	n/a				
Local-greedy	385.805	386.054	385.84889	0.076142993	11.9074
Local-coordinate	385.805	392.419	386.24143	0.904887395	9.2833
Local-strict	385.805	389.638	385.93522	0.550739092	9.6035

Appendix B

(I) Greatest cluster for the profile at a given k		
k	Axis for biggest cluster	Number of consistent preference orderings
0	1 2 3 4 5 6 7 8 9	30
1	8 7 6 5 4 3 2 1 9	170
2	8 7 6 3 5 4 2 1 9	536
3	8 7 5 3 6 4 2 1 9	1198

(II) Greatest cluster at each iteration, for k=2		
i	Voters covered (cumulative)	Number of consistent preference orderings for biggest cluster at i
1	14.11%	536
2	23.79%	368
3	31.89%	308
4	39.32%	282
5	45.74%	244
6	51.13%	205
7	56.18%	192
8	60.39%	160
9	64.47%	155
10	68.05%	136
11	71.45%	129
12	74.42%	113
13	77.08%	101
14	79.55%	94

(III) Greatest cluster at each iteration, for k=3		
i	Voters covered (cumulative)	Number of consistent preference orderings for biggest cluster at i
1	31.53%	1198
2	50.08%	705
3	62.79%	483
4	73.26%	398
5	80.37%	270
6	85.66%	201

References

[1] Jon Elster and Aanund Hylland. *Foundations of Social Choice Theory* (pp. 213-238)

[2] Hervé Moulin. *Axioms of Cooperative Decision Making* (p. 2)

[3] Xin Sui, Craig Boutilier, Tuomas Sandholm. *Analysis and optimization of Multi-dimensional Percentile Mechanisms* (pp. 1-7)

[4] Roger B Myerson. *Game Theory: Analysis of Conflict* (pp. 1-2)

[5] Noam Nisan, Tim Roughgarden, Eva Tardos, Vijay V. Vazirani. *Algorithmic Mechanism Design*
(pp. 209-239)

[6] Dinesh Garg, Y. Narahari, Sujit Gujar. *Foundations of Mechanism Design: A Tutorial* (p. 17)

[7] Miguel Angel Ballester, Guillaume Haeringer. *A Characterization of Single-Peaked Preferences* (p. 2)

[8] Pinyan Lu, Yajun Wang, Yuan Zhou. *Tighter Bounds for Facility Games* (pp. 1-12)

[9] Salvador Barbera. *Generalized Median Voter Schemes and Committees* (pp. 1-2)

[10] Ariel D. Procaccia, Moshe Tennenholtz. *Approximate Mechanism Design Without Money* (pp. 1-10)

[11] Bruno Escoffier, Jérôme Lang, Meltem Ozturk. *Single-peaked consistency and its complexity* (pp. 1-5)

[12] Gabor Erdelyi, Martin Lackner, Andreas Pfandler. *The Complexity of Nearly Single-Peaked Consistency* (pp. 1-15)