Improving massive experiments with threshold blocking*

Michael J. Higgins† Fredrik Sävje‡ Jasjeet S. Sekhon§

April 5, 2016

Abstract

Inferences from randomized experiments can be improved by blocking: assigning treatment in fixed proportions within groups of similar units. However, the use of the method is limited by the difficulty in deriving these groups. Current blocking methods are restricted to special cases or run in exponential time; are not sensitive to clustering of data points; and are often heuristic, providing an unsatisfactory solution in many common instances. We present an algorithm that implements a new, widely applicable class of blocking—threshold blocking—that solves these problems. Given a minimum required group size and a distance metric, we study the blocking problem of minimizing the maximum distance between any two units within the same group. We prove this is a NP-hard problem and derive an approximation algorithm that yields a blocking where the maximum distance is guaranteed to be at most four times the optimal value. This algorithm runs in O(n log n) time with O(n) space complexity. This makes it the first blocking method with an ensured level of performance that works in massive experiments. While many commonly used algorithms form pairs of units, our algorithm constructs the groups flexibly for any chosen minimum size. This facilitates complex experiments with several treatment arms and clustered data. A simulation study demonstrates the efficiency and efficacy of the algorithm; tens of millions of units can be blocked using a desktop computer in a few minutes.

1. Introduction

Properly executed experiments with random assignment guarantee that estimated treatment effects are equal to the true causal effects of interest in expectation. However, only one assignment is realized for a particular experiment, and there could be chance differences between treatment and control groups that muddle any comparison. Indicative of such differences are imbalances in observed baseline characteristics between the treatment groups. For example, in a medical study on the effect that a drug has on life expectancy, it may occur by chance that the control group is older and sicker than the treatment group. Whenever imbalances in prognostically important covariates are observed, there is reason to suspect that the resulting estimates are inaccurate. Studies that do not attended to this issue cannot be considered to follow the gold standard of randomized experiments [1]: viewed before assignment, investigators allowed for unnecessarily high variance; viewed after assignment, they allowed the estimator to be biased conditional on the observed distribution of covariates.

Since R.A. Fisher's canonical treatment [2], *blocking* has been the default experimental design to deal with this problem. With this design, the investigator forms groups of units, or *blocks*, that are as similar as possible. Treatments are then randomly assigned in fixed proportions within blocks and independently across them. This prevents imbalances in observed covariates, which can increase precision if these covariates are predictive of outcomes.

Unadjusted estimates for even massive experiments are often too variable to enable reliable inferences because the effects of interest may be small and distributional issues result in surprisingly large variances. A prominent case is A/B testing of the effectiveness of online advertising [3]. The effects of the adverts are generally very small (although economically relevant due to the low costs), and consumers' behaviors tend to follow distributions with fat tails [4]. Another example is offered by a recent fifteen-million person experiment on social influence and political mobilization, where covariate adjustment was needed to obtain significant results [5].

Moreover, with the rise of massive data, researchers, policy makers and industry leaders have become increasingly interested in making fine-grained inferences and targeting treatments to subgroups [6]. The recent focus on

*We thank Peter Aronow, Walter R. Mebane, Jr., Marc Ratkovic, and Yotam Shem-Tov for helpful comments. This research is partially supported by Office of Naval Research (ONR) grant N00014-15-1-2367.

†Department of Statistics, Kansas State University.
‡Department of Economics, Uppsala University.
§Department of Political Science and Department of Statistics, University of California, Berkeley.

personalized and precision medicine is a noteworthy example [7]. Even with large experiments, subgroups of interest often lack data because of the vagaries of random assignment and the curse of dimensionality. Blocking enables researchers to define the subgroups of interest *ex ante*. This ensures that there will be sufficient data to make fine-grained inferences.

Finally, because blocking adjusts for covariates in the design of the study, it limits both the need for and the effect of adjusting the experiment *ex post*. Such adjustments often lead to incorrect test levels if investigators specify models based on the observed treatment assignments [8], or if they pick models based on test results—a habit that appears to be prevalent [9].

In short, blocking is an essential tool for experiential design. It enables one to follow Fisher's advice that Nature should be asked one question at a time, which is the central motivation for random assignment in the first place [10].

Despite its providence, usefulness, and wide applicability, there are many situations where an effective blocking design is desirable but where none is possible or feasible. In particular, current blocking algorithms have primarily focused on the special case where blocks with exactly two units are desired, the so called *matched-pair design* [11]. There exist optimal, polynomial time algorithms for this design, such as non-bipartite matching [12], but they are limited to experiments with only two treatment conditions. While there exist heuristic algorithms that can facilitate larger block sizes, their theoretical properties are unknown and their performance has not been fully evaluated [13]. In many cases, even with relatively modest samples and considerable computational power, several years would be required to obtain results using algorithms with a proven level of optimality. For the largest of experiments, existing algorithms are too computationally demanding even for the matched-pair design.

In this paper, we introduce the first algorithm that produces guaranteed near-optimal blockings for any desired block size. Specifically, we consider the blocking problem where one wants to minimize the greatest within-block dissimilarity, as measured by an arbitrary distance metric, subject to a minimum required block size. We prove that this problem is NP-hard, and we provide an approximation algorithm that in the worst case produces a solution that is four times greater than the optimum. The algorithm uses computational resources very efficiently: it is guaranteed to terminate in linearithmic time with linear space complexity. This makes it applicable in many cases where existing algorithms are impractical, including experiments with large samples or multi-armed treatment schemes.

In additional to large data, our approximation algorithm is likely to perform well in traditional, smaller experiments, not the least when designs other than matched-pairs are desired. Our formulation of the blocking problem, *threshold blocking*, differs from the past literature in that it allows for some flexibility in the block structure. This leads to blockings that respect natural clusters of units which may improve performance.

2. Blocking as a graph partition problem

A *blocking* of an experiment's sample is a partition of its units into disjoint sets, referred to as *blocks*. The blocking problem is to find a blocking where units assigned to the same block are as similar as possible—either to minimize differences on prognostically important covariates or to facilitate the study of subgroups of interest. In the former case, when treatments are assigned in fixed proportions within blocks, blocking reduces imbalances between treatment groups and improves the precision of estimated effects.

Blocking problems can be viewed as graph partitioning problems [12, 14]. Each experiment yields a weighted graph where vertices represent units in the sample. Edges connect each pair of units, and edge costs are measured dissimilarity between corresponding units (e.g., the Euclidean or Mahalanobis distance between their covariate vectors). Minimizing the within-block edge costs when this graph is partitioned subject to a cardinality condition is equivalent to deriving an optimal blocking. In the matched-pair design, the objective is to minimize the sum of all within-block edge costs subject to that each block contains exactly two vertices.

To improve blockings and facilitate the approximation algorithm, we consider a formulation of the blocking problem that differs from the past literature in three aspects. First, we facilitate designs other than matched-pairs by allowing for any desired block size. Second, we consider blockings where each block is required to contain at least the desired number of units. Such *threshold blockings* have several advantages compared to the *fixed-sized* blockings derived by previous methods, where blocks are forced to be exactly of the desired size. Every fixed-sized blocking is also a threshold blocking; hence for any sample and objective function, the optimal solution for the latter case is guaranteed to be at least as good as in the former [15]. In particular, fixed-sized blocks might not respect natural clusterings of units, and one is sometimes forced to assign similar units to different blocks just to satisfy the cardinality condition.

Third, we consider a *bottleneck* objective function. That is, we wish to find a blocking that minimizes the maximum within-block edge cost—making the two least similar units assigned to the same block as similar as

possible. The bottleneck objective has some advantages over the commonly used sum (or average) objective. Going back to at least Cochran, statisticians have observed that few large imbalances are often more problematic than many small ones, especially when blocking is combined with *ex post* adjustments [16]. Furthermore, parallel to monotonic imbalance bounding in observational studies [17], controlling the maximum imbalance within a block guarantees that the average imbalance cannot exceed this maximum after treatments are assigned. If an infinity norm is used to measure dissimilarity (i.e., the Chebyshev distance), this also applies to each covariate in isolation. Minimizing sums or averages does not provide such guarantees. Finally, bottleneck optimization problems often have approximate solutions that can be found efficiently [18]. While the algorithm cannot readily be extended to other objective functions, it has a local optimality property that provides good performance with respect to the average within-block edge cost.

2.1. The bottleneck threshold blocking problem

Let k denote a threshold for the minimum block size. Consider the complete graph $G = (V, E)$ describing an experimental sample, where V denotes the set of n vertices (the experimental units) and E denotes the set of edges connecting all pairs of vertices.[1] For each $ij \in E$ there is an associated cost, c_{ij}, indicating the dissimilarity between i and j; lower costs mean that units are more similar. We require that these costs satisfy the triangle inequality:

$$\forall ij, j\ell, i\ell \in E, c_{ij} + c_{j\ell} \geq c_{i\ell}. \tag{1}$$

This ensures that the direct route between two vertices is no longer than a detour through a third vertex. All distance metrics fulfill this criterion by definition.

Definition 1 *A threshold blocking with threshold k is a partition $\mathbf{b} = \{V_1, \cdots, V_m\}$ of V where each block satisfies the size threshold:*

$$\forall\, V_x \in \mathbf{b},\ |V_x| \geq k. \tag{2}$$

Definition 2 *The subgraph generated by a blocking $\mathbf{b} = \{V_1, \ldots, V_m\}$, denoted $G(\mathbf{b}) = (V, E(\mathbf{b}))$, is the union of subgraphs of G induced by the components of $\mathbf{b}$; that is, an edge $ij \in E(\mathbf{b})$ only if i and j are in the same block:*

$$E(\mathbf{b}) \equiv \{ij \in E : \exists V_x \in \mathbf{b}, i, j \in V_x\}. \tag{3}$$

Let $\mathbf{B}_k$ denote the set of all possible threshold blockings of G with a threshold of k. The bottleneck threshold blocking problem is to find a blocking in $\mathbf{B}_k$ such that the maximum within-block dissimilarity is minimized. This amounts to finding an optimal blocking $\mathbf{b}^* \in \mathbf{B}_k$ such that the largest edge cost in $G(\mathbf{b}^*)$, is as small as possible; let λ denote this minimum:

$$\max_{ij \in E(\mathbf{b}^*)} c_{ij} = \min_{\mathbf{b} \in \mathbf{B}_k} \max_{ij \in E(\mathbf{b})} c_{ij} \equiv \lambda. \tag{4}$$

Definition 3 *An α-approximation algorithm for the bottleneck threshold blocking problem derives a blocking $\mathbf{b} \in \mathbf{B}_k$ with a maximum within-block cost no larger than $\alpha\lambda$:*

$$\max_{ij \in E(\mathbf{b})} c_{ij} \leq \alpha\lambda. \tag{5}$$

In the appendix, we show that, unless $\mathbf{P} = \mathbf{NP}$, no polynomial-time $(2 - \epsilon)$-approximation algorithm exists for any $\epsilon > 0$. Therefore, the problem is NP-hard, and finding an optimal solution is computationally intractable except for special cases or very small samples.

3. An approximately optimal blocking algorithm

We present a 4-approximation algorithm for the threshold blocking problem. Outside of an initial construction of a nearest neighbors graph, this algorithm has $O(kn)$ time and space complexity. Hence, it can be used in experiments with millions of units. Although the algorithm guarantees a threshold blocking with maximum within-block cost no larger than 4λ, simulations indicate that derived blockings are much closer to the optimum in practice.

3.1. The algorithm

Given the graph representation of the experimental sample, $G = (V, E)$, and a pre-specified threshold k, the approximate blocking algorithm proceeds as follows:

1. Construct a $(k-1)$-nearest neighbor subgraph of G. Denote this graph $G_{nn} = (V, E_{nn})$.

2. Find a maximal independent set of vertices, $\mathbf{S}$, in the second power of the $(k-1)$-nearest neighbor subgraph, G^2_{nn}. Vertices in $\mathbf{S}$ are referred to as the block seeds.

3. For each seed $i \in \mathbf{S}$, create a block comprised of its closed neighborhood in G_{nn}, $V_i = \mathrm{N}_{G_{nn}}[i]$.

4. For each yet unassigned vertex, assign it to any block that contains one of its adjacent vertices in G_{nn}.

[1] Refer to the appendix for graph theoretical terminology and notation used in this paper.

When the algorithm terminates, the collection of blocks, $\mathbf{b}_{alg} = \{V_i\}_{i \in \mathbf{S}}$, is a valid threshold blocking of the experimental units that satisfies the optimality bound.

Informally, the algorithm constructs the blocking by selecting suitable vertices, the seeds, from which the blocks are grown. Seeds are spaced sufficiently far apart so as not to interfere with each other's growth, but they are dense enough so that all non-seed vertices have a seed nearby. Specifically, the second step of the algorithm prevents any vertex from being adjacent to two distinct seeds in the $(k-1)$-nearest neighbor subgraph, but also never more than a walk of two edges away from a seed. This ensures that the seeds' closed neighborhoods do not overlap, while vertices assigned to the same block are at a close geodesic distance. Figure 1 illustrates how the algorithm constructs blocks in an example sample.

3.2. Validity and complexity

We first prove that the algorithm is guaranteed to produce a valid threshold blocking, and then its time and space complexity.

Lemma 1 *For any non-seed vertex, $i \notin \mathbf{S}$:*

1. *There exist no two seeds both adjacent to i in G_{nn}.*
2. *There exists a walk in G_{nn} of two or fewer edges from i to the seed of the block that i is assigned to.*

Proof. The lemma follows from that $\mathbf{S}$ is a maximal independent set in G_{nn}^2. Refer to the appendix for a complete proof. □

Theorem 1 (Validity) *The blocking algorithm produces a threshold blocking: $\mathbf{b}_{alg} \in \mathbf{B}_k$.*

Proof. By Lemma 1, each vertex assigned in the third step is adjacent to exactly one seed, thus it will be in exactly one block. In the fourth step, vertices are assigned to exactly one block each. This ensures that the blocks are disjoint and span V, thus $\mathbf{b}_{alg}$ is a partition of V.

All seeds have at least $k-1$ adjacent vertices in G_{nn}. In the third step these vertices and the seeds themselves will form the blocks ensuring that each block contains at least k vertices. This satisfies Definition 1. □

Theorem 2 (Complexity) *The blocking algorithm terminates in polynomial time using $O(kn)$ space.*

Proof. Naively, the $(k-1)$-nearest neighbor subgraph can be constructed by sorting each vertex' edge costs and finding its $k-1$ nearest neighbors. Thus, G_{nn} is constructed in at most $O(n^2 \log n)$ time [19]. To enable constant time access to the neighbors of any vertex, store the nearest neighbor subgraph in n lists containing each vertex' edges.

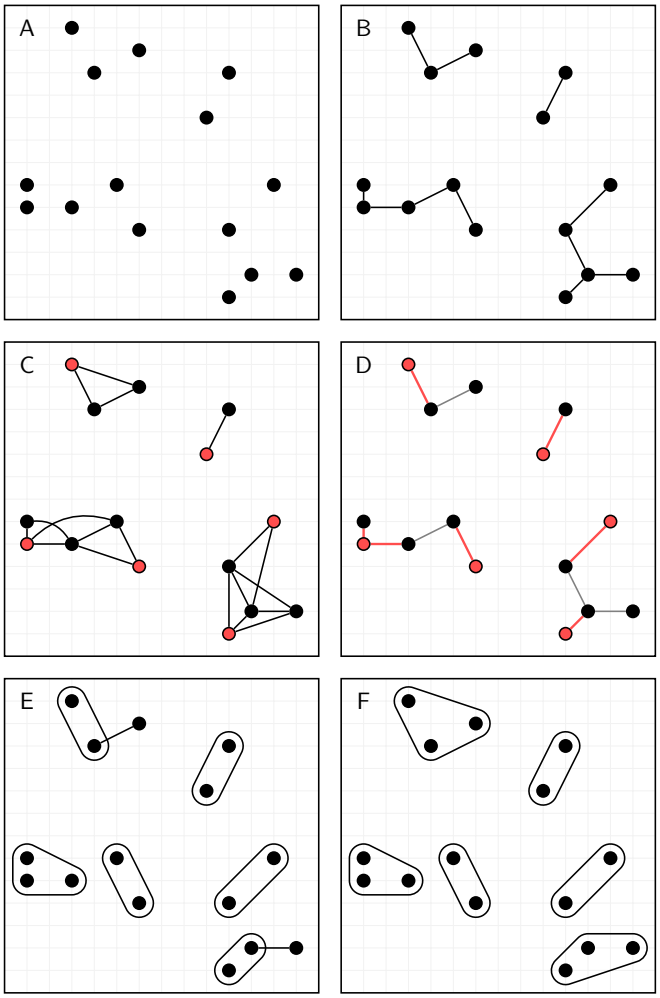

Figure 1: An illustration of the approximation algorithm for a sample with two-dimensional covariate data when a minimum block size of two is desired ($k = 2$). (A) The algorithm is provided with a set of data points and forms the graph by drawing an edge between all possible pairs of units. The edges are here omitted to ease presentation. (B) A $(k-1)$-nearest neighbor subgraph is constructed. (C) The second power of the nearest neighbor subgraph is derived, as shown by the edges, and a maximal independent set is found, as shown by the red vertices (the seeds). (D) All vertices adjacent to a seed in the nearest neighbor subgraph are included in the blocks formed by the seeds, as shown by the edges marked in red. (E) The two yet unassigned vertices are assigned to the blocks that contain one of their adjacent vertices in the nearest neighbor subgraph. (F) The final blocking.

There can at most be $(k-1)n$ edges in G_{nn}. This implies an $O(kn)$ space complexity for the edge lists.

Using the edge lists, a maximal independent set in the second power of G_{nn} can be found in $O(kn)$ time without changing the space complexity. See the appendix for details on this subroutine. The third step is completed within $O(n)$ time as Lemma 1 ensures that at most n units will be assigned to blocks in this step and the edge lists enables constant time access to the seeds' neighbors. In the fourth step, it will never be necessary to search through all edge lists more than once, implying a complexity of $O(kn)$. □

Remark 1 *After the initial construction of the $(k-1)$-nearest neighbor subgraph, the algorithm terminates in $O(kn)$ time. As the nearest neighbor search problem is well-studied, the naive subroutine in the proof can be improved on in most applications. In particular, most experiments will have reasonably low-dimensional metric spaces. Using specialized algorithms, the subgraph can in that case be constructed in $O(kn \log n)$ expected time [20] or worst-case time [21]. If the covariates are not few to begin with, it is often advisable to use some dimensionality reduction technique before blocking so to extract the most relevant information.*

Run time can also be improved by using an approximate nearest neighbor search algorithm. However, approximate optimality is not guaranteed in that case.

Remark 2 *It is rarely motivated to increase the block size as the sample grows, thus k can be considered fixed in the typical experiment. When k is fixed and one can use a specialized procedure to derive the nearest neighbor subgraph, the algorithm has $O(n \log n)$ time and $O(n)$ space complexity.*

3.3. Approximate optimality

To prove the optimality bound, we will first show that the edge costs in the $(k-1)$-nearest neighbor subgraph are bounded. As the algorithm ensures that vertices in the same block are at a close geodesic distance in that subgraph, approximate optimality follows from the triangle inequality.

Lemma 2 *No edge cost in G_{nn} can be greater than the maximum cost in the optimal blocking:*

$$\forall ij \in E_{nn},\ c_{ij} \leq \lambda. \tag{6}$$

Proof. Consider the graph:

$$G_\lambda = (V, E_\lambda = \{ij \in E : c_{ij} \leq \lambda\}). \tag{7}$$

For all edges in an optimal blocking, $ij \in E(\mathbf{b}^*)$, we have $c_{ij} \leq \lambda$ from optimality. It follows that $E(\mathbf{b}^*) \subseteq E_\lambda$.

Let $c_+ = \max\{c_{ij} : ij \in E_{nn}\}$ and consider:

$$G_+ = (V, E_+ = \{ij \in E : c_{ij} < c_+\}). \tag{8}$$

The minimum degree of this graph, $\delta(G_+)$, must be less than $k-1$. If not, a $(k-1)$-nearest neighbor graph exists as a subgraph of G_+. As this new graph does not contain c_+, it is contradictory that c_+ is the maximum edge cost in G_{nn}.

Suppose that $c_+ > \lambda$. It then follows that $E_\lambda \subseteq E_+$, thus:

$$\delta[G(\mathbf{b}^*)] \leq \delta(G_\lambda) \leq \delta(G_+) < k-1. \tag{9}$$

That is, there exists a vertex in $G(\mathbf{b}^*)$ with fewer than $k-1$ edges. It follows that there must exist a block in $G(\mathbf{b}^*)$ with fewer than k vertices and, as Definition 1 then is violated, it cannot be a valid blocking. The contradiction proves that $c_+ \leq \lambda$ which bounds all edges in E_{nn}. □

Theorem 3 (Approximate optimality) *The blocking algorithm is a 4-approximation algorithm:*

$$\max_{ij \in E(\mathbf{b}_{alg})} c_{ij} \leq 4\lambda. \tag{10}$$

Proof. Let $\mathbf{b}_{alg}$ denote the blocking produced by the algorithm. Consider any within-block edge $ij \in E(\mathbf{b}_{alg})$. We must show that c_{ij} is bounded by 4λ.

If $ij \in E_{nn}$, we have $c_{ij} \leq \lambda$ by Lemma 2. If $ij \notin E_{nn}$ and $i \notin \mathbf{S}$, $j \in \mathbf{S}$, then by Lemma 1, there exists some ℓ so that $i\ell, \ell j \in E_{nn}$. Lemma 2 applies to both these edges. By Equation **1**, the triangle inequality, it follows:

$$c_{ij} \leq c_{i\ell} + c_{\ell j} \leq \lambda + \lambda = 2\lambda. \tag{11}$$

If $ij \notin E_{nn}$ and $i, j \notin \mathbf{S}$, let $\ell \in \mathbf{S}$ be the seed in the block that vertices i and j are assigned to. From above we have $c_{i\ell}, c_{\ell j} \leq 2\lambda$, and by the triangle inequality:

$$c_{ij} \leq c_{i\ell} + c_{\ell j} \leq 2\lambda + 2\lambda = 4\lambda. \tag{12}$$

As there is exactly one seed in each block, $i, j \in \mathbf{S}$ is not possible and we have considered all edges in $E(\mathbf{b}_{alg})$. □

Remark 3 *In some settings, a slight reduction in the sample size is acceptable or required, e.g., for financial constraints or when blocks are constructed before units are sampled using secondary data sources. In these cases, the algorithm can easily be altered into a 2-approximation algorithm. By terminating at the end of the third step and disregarding the unassigned vertices, one ensures that all remaining vertices are at most a distance of λ from the seed (where λ refers to the maximum distance in the optimal blocking of the selected subsample). Applying the triangle inequality proves that all edge costs in the blocking of the subsample is bounded by 2λ. It is also possible*

to apply a caliper to the blocking so to restrict the maximum possible edge cost by excluding some hard-to-block vertices.

A concern when using a bottleneck objective is that densely populated regions of the sample space will be ignored as the blocks in these regions will not affect the maximum edge cost. This is especially worrisome when there are a few hard-to-block vertices that result in a large λ. This can lead to poor performance as covariate balance often can be improved by ensuring good block assignments for all vertices. However, as the presented algorithm does not directly use the bottleneck objective to form the blocks, it avoids this issue. Instead, its optimality follows from the use of the nearest neighbor subgraph as the basis of blocking, and this graph's connection with the optimal edge cost as shown in Lemma 2.

The following theorem shows that our algorithm leads to approximate optimality not only in the complete sample, but also in all subsamples. Thus, if there is densely populated region, the algorithm ensures that the blocking is near-optimal also within that region.

Theorem 4 (Local approximate optimality) *Let* $\mathbf{b}_{sub} \subseteq \mathbf{b}_{alg}$ *be any subset of blocks from a blocking constructed by the algorithm. Define* $V_{sub} = \bigcup_{V_x \in \mathbf{b}_{sub}} V_x$ *as the set of all vertices contained in the blocks of* $\mathbf{b}_{sub}$*. Let* λ_{sub} *denote the maximum edge cost in an optimal blocking of* V_{sub}*. The subset of blocks is an approximately optimal blocking of* V_{sub}*:*

$$\max_{ij \in E(\mathbf{b}_{sub})} c_{ij} \leq 4\lambda_{sub}. \tag{13}$$

Proof. Theorem 4 is proven in the appendix.

3.4. Heuristic improvements

The algorithm allows for several improvements of heuristic character. While the guaranteed optimality bound or complexity level remains unchanged, we expect these changes to improve general performance. In particular, the algorithm has a tendency to construct blocks that are too large. While flexibility in the block size is beneficial—the main idea behind threshold blocking—the current version tends to overuse that liberty.

The first improvement exploits an asymmetry in the nearest neighbor subgraph which currently is disregarded. The cardinality condition is met as each seed and its $k-1$ nearest neighbors are assigned to the same block. However, in addition to those necessary neighbors, the current version assigns vertices that have the seed as their nearest neighbor to the block. With some minor alterations, detailed in the appendix, the algorithm can use a $(k-1)$-nearest neighbor *digraph* to form the blocks. This digraph is such that an arc (i.e., directed edge) is drawn from i to j if j is among the $(k-1)$ closest neighbors of i. Using the digraph, one can differentiate whether a vertex is a neighbor of the seed and *vice versa*.

There is rarely a unique maximal independent set in the second power of the nearest neighbor graph (i.e., the seeds). The current version selects one arbitrarily. The second improvement is to choose the seeds more deliberately. As each seed assigns at least $k-1$ vertices to its block, a straight-forward way to reduce the block sizes is to maximize the number of seeds—a larger set is expected to produce better blockings. The ideal may be the *maximum* independent set, but deriving such a set is a NP-hard problem. Most heuristic algorithms are, however, expected to perform well.

Third, despite the above improvements, the algorithm will occasionally produce blocks that a much larger than k. Whenever a block contains $2k$ or more vertices it can safely be split into two or more blocks, each containing at least k vertices. As the algorithm ensures that all edge costs satisfy the optimality bound and no edges are added by splitting, this can only lower the maximum within-block cost. In the appendix, we describe a greedy threshold algorithm for splitting blocks that runs fast and is expected to perform well. This greedy algorithm can also be used to block the complete sample, but will not perform on par with the approximation algorithm.

A fourth improvement changes how vertices are assigned in the fourth step. With larger desired block sizes, some blocks may contain peripheral vertices that are far from their seeds. In these cases, it is often beneficial to assign the remaining vertices in the fourth step to the block containing their closest seed instead of the block containing a closest neighbor. This avoids the situation where a vertex is assigned to distant block due to having a peripheral vertex close by. The optimality bound is maintained as Theorem 3 ensures that at least one seed exists at a distance of at most 2λ. If a vertex is not assigned to that seed, it must have been assigned to a seed that is at a closer distance.

Finally, once a blocking is derived, searching for moves or swaps of vertices between blocks can lead to improvements as in other partitioning problems [22]. It is, however, not feasible to let such searches continue until no additional improvements are possible (i.e., until a local optimal is found) as the flexible block structure allows for a vast number of moves and swaps.

4. Simulation study

We provide the results from a small simulation study. Apart from the original algorithm, we include versions that employ the improvements discussed in the previous section. Specifically, we include a version using the

nearest neighbor digraph (the first improvement) and a version using the first three improvements. A version that uses all four improvements is generally only beneficial with larger block sizes and is included when such settings are investigated in the appendix.

For comparison, we also include the greedy threshold blocking algorithm discussed in the appendix and the currently best-performing greedy fixed-sized blocking algorithm [13]. When $k = 2$, we can also include a commonly used implementation of the non-bipartite matching algorithm [12].

We investigate a simple setting where each data point is sampled independently from a uniform distribution over a two-dimensional plane:

$$x_1, x_2 \sim \mathcal{U}(0, 10), \tag{14}$$

and similarity is measured as the Euclidean distance on this plane. While many experiments will have data with higher dimensions, it is often not motivated to include all those dimensions when deriving blocks. Typically, one wants to reduce the dimensionality in a preprocessing step to extract the information that is most predictive of the potential outcomes [23]. The investigated algorithms are, however, not restricted to low-dimensional data.

All simulations are run on a single CPU core reflecting the performance of a modern desktop computer. See the appendix for details about the implementation of the algorithm and the hardware used to run the simulations.

4.1. Run time and memory

To investigate the resource requirements, we let each algorithm block samples containing between one hundred and 100 million data points generated from the model above. Each setting was replicated 250 times. As time and memory usage are quite stable over the runs, these replications suffice to investigate even small differences. In these simulations, the directed version performs almost identically to the original version, and it is omitted from the graphs to ease presentation.

Figure 2 presents the time and memory needed for the algorithms to successfully terminate. All versions of the approximation algorithm run fast and terminate within a minute for samples sizes up to a few millions. With 100 million data points the original version terminates within 11 minutes while the version will all three refinements does so in less than 16 minutes—all very manageable in real applications. Memory usage is increasing linearly at a slow rate for both versions. A modern desktop computer would have enough memory to block samples with tens of millions of units without problems.

The three comparison algorithms paint another picture altogether. For the rather modest sample size of 20,000 data points, these algorithms take more than 20 minutes—up to two hours—to terminate. Even more problematic is their extensive memory use. For samples larger than 50,000 data points, all three algorithms try to allocate more than 48 gigabytes of memory; under these settings, these algorithms do not terminate successfully, and no results can be shown.

Detailed results for these simulations and simulations with input data with higher dimensionality are presented in Tables 3, 4 and 5 in the appendix.

4.2. Minimizing distances

To investigate how well the algorithms minimize distances, we increase the number of replications to 5,000; this statistic is less stable and the differences between algorithms are smaller. However, with this number of replications, no difference can be attributed to simulation error.

The first panel of Table 1 shows the maximum within-block distance, averaged over the simulation rounds, when the desired minimum block size is two. Refer to Table 6 in the appendix for results when the desired minimum block size is four. We report values normalized by the performance of the approximation algorithm to ease interpretation. The two improved versions of the approximation algorithm lead to quite substantial decreases in the maximum distance. All versions of the approximation algorithm outperform the two greedy algorithms—for the fixed-sized version, drastically so. However, only the version with all three improvements performs better than non-bipartite matching.

In the second panel, the average within-block distance is presented. This is the objective of the greedy fixed-sized algorithm and non-bipartite matching, so it is not surprising that these algorithms show better performance on this statistic. Non-bipartite matching is here the best performing algorithm and only the approximation algorithm with full improvements outperforms the fixed-sized greedy algorithm.

The last panel presents the average size of the blocks produced by the different algorithms. The improvements discussed in the preceding section are shown to be effective in taming the original algorithm's tendency to construct blocks that are too large. However, smaller blocks do not automatically lead to better performance. This is evident from the greedy threshold algorithm which produces smaller blocks than the approximation algorithms but has worse performance. The two fixed-sized blocking algorithms produce blocks of constant size by construction.

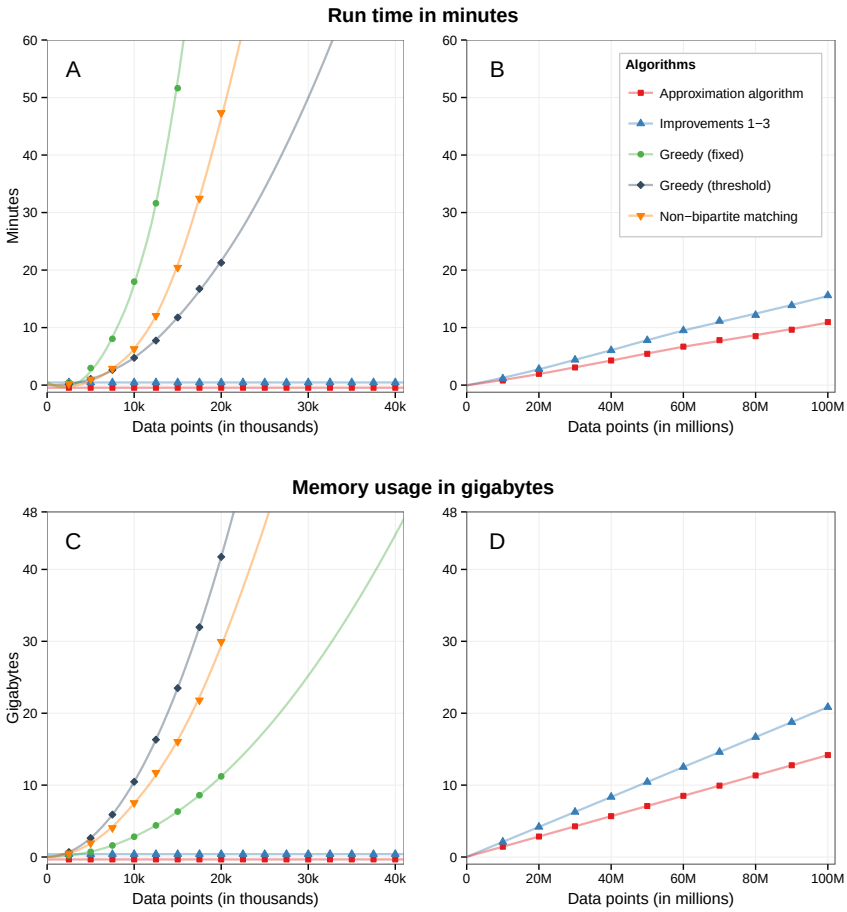

Figure 2: Run time (A, B) and memory usage (C, D) of five blocking algorithms with two-dimensional input data over a range of sample sizes. Marker symbols are actual simulation results, and the connecting lines are interpolations. Results are presented with different scales due to the large differences in performance. Results are presented for all algorithms for sample sizes up to 40,000 data points (A, C) while results for sample sizes up to 100 million data points are only shown for the two approximation algorithms (B, D). No simulations were successful for the greedy algorithms and non-bipartite matching for sample sizes larger than 20,000 due to excessive run time or memory use. The approximation algorithm presented in the paper (shown in red) has almost identical run time and memory usage as the version using the nearest neighbor digraph, as described in the section on heuristic improvements, and its results are not shown in the figure. See Table 3 in the appendix for detailed results.

Table 1: Performance of blocking algorithms by sample size: maximum and average within-block distances relative to the approximation algorithm and average block size

	Max. within-block distance			Avg. within-block distance			Avg. block size		
Algorithm	10^2	10^3	10^4	10^2	10^3	10^4	10^2	10^3	10^4
Approximation algorithm	1.000	1.000	1.000	1.000	1.000	1.000	2.67	2.66	2.66
Directed version	0.917	0.895	0.883	0.933	0.933	0.933	2.55	2.54	2.54
Improvements 1-3	0.791	0.755	0.729	0.825	0.826	0.826	2.31	2.30	2.30
Fixed greedy	3.126	8.149	21.938	0.931	0.924	0.908	2.00	2.00	2.00
Threshold greedy	1.076	1.148	1.191	1.079	1.116	1.127	2.33	2.33	2.33
Non-bipartite matching	0.838	0.795	0.765	0.742	0.732	0.728	2.00	2.00	2.00

4.3. Reducing uncertainty

To investigate how the blockings affect an estimator's performance, one must specify a data generating process for the outcome. The results are highly sensitive to the details of this process. Even blockings that are optimal with respect to within-block distances need not lead to the lowest variance. An extensive investigation is beyond the scope of this paper and we provide only indicative results.

We consider a simple setting with two treatment conditions when the desired minimum block size is two. Refer to Table 7 in the appendix for results when the minimum block size is four. The outcome (y) is the product of the covariates with additive, normally distributed noise:

$$y = x_1 x_2 + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, 1). \tag{15}$$

Note that the treatment does not enter into the model and thus has no effect—the potential outcomes are equal. The covariates are highly predictive in this model—more so than one can expect in a real application. This enables the methods to make the most out of the covariate information and helps us differentiate between the algorithms' performances. For all blocking methods, we will use the *block-size weighted difference-in-means* estimator to estimate treatment effects. Refer to the appendix for additional details on estimation. We run 5,000 replications in this setting and results are presented relative to the approximation algorithm. Tables 8 and 9 in the appendix present the results without normalization.

Table 2 presents results for the root of the average squared difference between estimates and the true treatment effect (RMSE) for each method's estimator. All blocking methods with proven optimality level perform well. For the smaller sample sizes, the approximation algorithm with all three improvements performs best, while non-bipartite matching is slightly better in the larger samples. All blocking methods seem, however, to converge as the sample size grows.

In addition to the six blocking methods, Table 2 includes the results of two methods that do not use blocking. In both cases, the RMSE is markedly higher than any of the methods using blocking. When controlling for imbalances using ordinary least squares regression, the RMSE is at least twice as large as that of any blocking method with proven optimality. When the estimate is completely unadjusted for imbalances, the RMSE is up to 20 times higher than for the approximation algorithm. However, this difference certainly overstates the benefits of blocking that one can expect in real applications as the covariates are unusually predictive in this simulation.

Table 2: Root mean square error relative to the approximation algorithm by sample size

Method	10^2	10^3	10^4
Approximation algorithm	1.000	1.000	1.000
Directed version	0.973	0.987	0.999
Improvements 1-3	0.931	0.960	0.992
Fixed greedy	1.609	1.598	1.152
Threshold greedy	1.207	1.146	1.041
Non-bipartite matching	0.952	0.949	0.983
Unadjusted	6.092	15.158	20.710
OLS adjustment	2.352	5.776	7.900

5. Discussion

Our approximation algorithm enables large and complex experiments to make use of blocking. No feasible algorithm with proven optimality properties has been available for massive experiments. Although many of the commonly used blocking algorithms run in polynomial time, none run in quasilinear time as the approximation algorithm does. Polynomial time is not a sufficient condition for tractability with very large data [24]. One can see this in our simulations.

The threshold blocking algorithm is expected to perform well in most cases given its approximate optimality. However, non-bipartite matching, when it is feasible, is likely the best choice in experiments with matched-pair designs because it is exactly optimal. Our simulation results seem to point towards this conclusion as well. The matched-pair design is, however, limited to the case of only two treatment conditions, and the design complicates the estimation of standard errors because block sizes of larger than two are sometimes needed; for example, to estimate conditional variances [25]. Our approximation algorithm is therefore an important arrow in the quiver of experimental design.

Whenever blocking reduces imbalances in prognostically important covariates, blocking will improve the expected precision of estimates. In some settings, even when the covariates contain no information about the outcomes, blocking cannot increase the variance of the treatment effect estimator compared to when no blocking is done [25, 26]. However, theoretical results depend on the randomization model, estimand, and estimator used. There are some rare instances where blocking may decrease precision.

As an alternative to blocking, some advocate rerandomization when a given randomization results in poor balance in observed covariates [27, 28]. Rerandomization restricts the randomization scheme, as assignments with poor balance are ruled out. If the rule for

which randomizations are acceptable is precise and set *a priori*, randomization inference is well-defined. One worries, however, that researchers will not use well-specified rules that they will later recall to restrict randomization distributions. Especially if the initial randomization results in good balance, it is doubtful that investigators will adjust their test levels as they had planned.

Far more common than blocking or re-randomization are *ex post* methods of adjusting experimental data such as post-stratification or using a model-based estimator that incorporates covariate information. Such methods can work well. For example, post-stratification is nearly as efficient as blocking: the difference in their variances is on the order of $1/n^2$, with a constant depending on treatment proportion [29]. However, post-stratification can increase variance if the number of strata is large and the strata are poorly chosen. Regression adjustment can provide significant gains in precision [30, 31], and model-based hypothesis tests can be asymptotically valid even when the adjustment model is misspecified [32]. However, regression adjustment, like post-stratification, may increase the finite sample variance, and will do so on average for any sample size, if the covariates are not informative [33].

A key argument in favor of blocking as opposed to *ex post* adjustment is that one is increasing the transparency of the analysis by building covariate adjustment into the experimental design. The results cited regarding post-stratification and model adjustment assume that the investigator did not pick the strata or model as a function of the realized treatment assignment. One further assumes that the investigator does not run a number of adjustment models, and then only report the one with the desired results. Human nature being what it is, this assumption is probably optimistic. A major benefit of randomized experiments, aside from the randomization, is that the design stage is separated from the analysis stage by construction [34]. Blocking allows one to use design-based estimators that adjust for covariate information [35]. The less that there is to do at the analysis stage, the less likely it is that the investigator will fish for particular results, unconsciously or not.

When researchers select adjustment models based on observed p-values, it is called p-hacking, and the habit appears to be prevalent [9]. Because of concerns about p-hacking, there has been a move towards creating pre-analysis plans for experimental studies both in medicine and the social sciences. Such plans force researchers to lay out in advance how they will analyze the experiment and what subgroups and outcomes are of primary interest. Unfortunately, evidence from medicine, where the practice is best established, shows that pre-analysis plans are often ignored and allow significant leeway in selection of covariates, subgroups, outcomes, and adjustment methods, and readers and reviewers are rarely informed of departures [36]. Blocking allows one to encode into the design of a study the covariates the investigator *a priori* thinks are important. After randomization, one may still adjust for these variables as small imbalances may remain. Blocking then acts as an effective signal that the investigator intended to do such adjustments before seeing initial results. Moreover, some covariates may not be measured at the time of randomization, and they could be adjusted *ex post*, although concerns about p-hacking may arise.

Finally, blocking is motivated by partial knowledge about how the covariates relate to the outcomes. The performance of any blocking algorithm depends on how well the chosen similarity measure captures this relationship. As this choice is subject-specific, general recommendations are hard to come by. However, as noted in our remarks, if one has many covariates, some dimension reduction to those that most likely relate to the outcomes is often advantageous. If more complete knowledge exists, such as a good estimate of the potential outcomes under control, one would gain more precision by directly blocking on that estimate.

There are no free lunches in statistics, but blocking comes close. It has few downsides and risks relative to complete randomization, other than computational challenges, and any experimenter should be motivated to block their sample. In this paper, we have enabled the technique for experiments where it previously was infeasible. Fast, near-optimal algorithms could be useful also when *ex post* adjustments are needed, e.g. when the investigator has no or limited control over treatment assignment. While the our algorithm is not directly applicable to these settings, we plan to extend it to post-stratification, matching and clustering in future work. We also plan to analyze the properties of different estimators, and how they differ between threshold versus fixed-sized blocking [15, 37].

References

[1] Rubin DB (2008) Comment: The design and analysis of gold standard randomized experiments. *J Am Stat Assoc* 103(484):1350–1353.

[2] Fisher RA (1926) The arrangement of field experiments. *Journal of the Ministry of Agriculture of Great Britain* 33:503–513.

[3] Lewis R, Rao J (2015) The unfavorable economics of measuring the returns to advertising. *Q J Econ* 130(4). In press.

[4] Fithian W, Wager S (2015) Semiparametric expo-

nential families for heavy-tailed data. *Biometrika* 102(2):486–493.

[5] Jones JJ, Bond RM, Bakshy E, Eckles D, Fowler JH (2015) Social influence and political mobilization: Further evidence from a randomized experiment in the 2012 U.S. presidential election. PNAS Big Data and Causality Colloquium paper.

[6] Athey S, Imbens G (2015) Machine learning methods for estimating heterogeneous causal effects. PNAS Big Data and Causality Colloquium paper.

[7] Ashley EA (2015) The precision medicine initiative: A new national effort. *JAMA* 313(21):2119–2120.

[8] Permutt T (1990) Testing for imbalance of covariates in controlled experiments. *Stat Med* 9(12):1455–1462.

[9] Simmons JP, Nelson LD, Simonsohn U (2011) False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychol Sci* 22(11):1359–1366.

[10] Speed TP (1992) in *Breakthroughs in Statistics*, Springer Series in Statistics, eds. Kotz S, Johnson NL. (Springer, New York), pp. 71–81.

[11] Imai K, King G, Nall C (2009) The essential role of pair matching in cluster-randomized experiments, with application to the Mexican universal health insurance evaluation. *Stat Sci* 24(1):29–53.

[12] Greevy R, Lu B, Silber JH, Rosenbaum P (2004) Optimal multivariate matching before randomization. *Biostatistics* 5(2):263–275.

[13] Moore RT (2012) Multivariate continuous blocking to improve political science experiments. *Polit Anal* 20(4):460–479.

[14] Rosenbaum PR (1989) Optimal matching for observational studies. *J Am Stat Assoc* 84(408):1024–1032.

[15] Sävje F (2015) The performance and efficiency of threshold blocking. arXiv:1506.02824.

[16] Cochran WG (1965) The planning of observational studies of human populations. *J R Stat Soc Ser A* 128(2):234–266.

[17] Iacus SM, King G, Porro G (2011) Multivariate matching methods that are monotonic imbalance bounding. *J Am Stat Assoc* 106(493):345–361.

[18] Hochbaum DS, Shmoys DB (1986) A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM* 33(3):533–550.

[19] Knuth DE (1998) *Sorting and searching*, The Art of Computer Programming. (Addison Wesley Longman, Redwood City, CA) Vol. 3, 2$^{\text{th}}$ edition.

[20] Friedman JH, Bentley JL, Finkel RA (1977) An algorithm for finding best matches in logarithmic expected time. *ACM Trans Math Softw* 3(3):209–226.

[21] Vaidya PM (1989) An o(n log n) algorithm for the all-nearest-neighbors problem. *Discrete Comput Geom* 4(1):101–115.

[22] Kernighan B, Lin S (1970) An efficient heuristic procedure for partitioning graphs. *Bell Syst Tech J* 49(2):291–307.

[23] Imbens GW, Rubin DB (2015) *Causal Inference for Statistics, Social, and Biomedical Sciences.* (Cambridge University Press, New York).

[24] National Research Council (2013) *Frontiers in massive data analysis.* (The National Academies Press, Washington, DC).

[25] Imbens G (2011) Experimental design for unit and cluster randomized trials. International Initiative for Impact Evaluations.

[26] Imai K (2008) Variance identification and efficiency analysis in randomized experiments under the matched-pair design. *Stat Med* 27(24):4857–4873.

[27] Hayes RJ, Moulton LH (2009) *Cluster randomised trials.* (CRC press, London).

[28] Morgan KL, Rubin DB (2012) Rerandomization to improve covariate balance in experiments. *Ann Stat* 40(2):1263–1282.

[29] Miratrix LW, Sekhon JS, Yu B (2013) Adjusting treatment effect estimates by post-stratification in randomized experiments. *J R Stat Soc Series B Stat Methodol* 75(2):369–396.

[30] Bloniarz A, Liu H, Zhang CH, Sekhon JS, Yu B (2015) Lasso adjustments of treatment effect estimates in randomized experiments. PNAS Big Data and Causality Colloquium paper.

[31] Rosenblum M, van der Laan MJ (2010) Simple, efficient estimators of treatment effects in randomized trials using generalized linear models to leverage baseline variables. *Int J Biostat* 6(1).

[32] Rosenblum M, van der Laan MJ (2009) Using regression models to analyze randomized trials: Asymptotically valid hypothesis tests despite incorrectly specified models. *Biometrics* 65(3):937–945.

[33] Lin W (2013) Agnostic notes on regression adjustments to experimental data: Reexamining Freedman's critique. *Ann Appl Stat* 7(1):295–318.

[34] Rubin DB (2008) For objective causal inference, design trumps analysis. *Ann Appl Stat* 2(3):808–840.

[35] Aronow PM, Middleton JA (2013) A class of unbiased estimators of the average treatment effect in randomized experiments. *J Causal Inference* 1(1):135–154.

[36] Humphreys M, de la Sierra RS, van der Windt P (2013) Fishing, commitment, and communication: A proposal for comprehensive nonbinding research registration. *Polit Anal* 21(1):1–20.

[37] Higgins M, Sävje F, Sekhon JS (2015) Blocking estimators and inference under the Neyman-Rubin model. arXiv:1510.01103.

[38] Kirkpatrick DG, Hell P (1978) *On the completeness of a generalized matching problem*, Proceedings of the Tenth Annual ACM Symposium on Theory of Computing. (ACM, New York), pp. 240–245.

[39] Chen Y, Davis TA, Hager WW, Rajamanickam S (2008) Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans Math Softw* 35(3):22:1–22:14.

Appendices

A. Graph theoretical definitions

Let $G = (V, E)$ be an arbitrary graph.

Complete graph G is complete if $ij \in E$ for any two vertices $i, j \in V$. A complete graph with n vertices is denoted K_n.

Spanning A graph $G' = (V, E')$ is a spanning subgraph of G if they contain the same set of vertices and $E' \subseteq E$.

Induced A subgraph $G' = G[V'] = (V', E')$ is induced on G by $V' \subseteq V$ if G' contains all edges in $ij \in E$ that connects vertices in V' and no other edges:

$$E' \equiv \{ij \in E : i, j \in V'\}.$$

Adjacent Vertices i and j are adjacent in G if $ij \in E$.

Degree The degree of vertex i in G is its number of adjacent vertices:

$$\deg(i) \equiv |\{j \in V : ij \in E\}|.$$

Minimum degree The minimum degree, $\delta(G)$, of G is the minimum degree among the vertices in G:

$$\delta(G) \equiv \min_{i \in V} \deg(i)$$

Neighborhood A neighborhood of vertex i, $\mathrm{N}_G(i)$, in G is the set of vertices adjacent to i:

$$\mathrm{N}_G(i) \equiv \{j \in V : ij \in E\}.$$

A *closed* neighborhood also contains i: $\mathrm{N}_G[i] \equiv \mathrm{N}_G(i) \cup i$.

Independent set A set of vertices $I \subseteq V$ is independent in G if no vertices in the set are adjacent:

$$\nexists\, i, j \in I,\ ij \in E.$$

Maximal independent set An independent set of vertices I in G is maximal if for any additional vertex $i \in V$ the set $i \cup I$ is not independent:

$$\forall\, i \in V \setminus I,\ \exists\, j \in I,\ ij \in E.$$

The *maximum independent set* in G is the maximal independent set with the largest cardinality among all independent set in G.

Walk A walk from $i = k_0$ to $j = k_m$ of length m in G is a $(m+1)$-tuple of vertices, with an edge between each adjacent pair, connecting i and j:

$$(k_0, k_1, \cdots, k_m) :\ \forall\, 1 \leq \ell \leq m,\ k_{\ell-1}k_\ell \in E.$$

Power The d^{th} power of G is a graph $G^d = (V, E^d)$ where an edge $ij \in E^d$ if there exists a walk from i to j in G of d or fewer edges.

Partition A partition of V is a set of subsets $\mathbf{p} = \{V_1, \ldots, V_m\}$ satisfying:

1. (Non-empty) $\forall\, V_x \in \mathbf{p},\ \emptyset \neq V_x \subseteq V$,
2. (Disjoint) $\forall\, V_x, V_y \in \mathbf{p},\ (V_x \neq V_y) \ \Rightarrow\ (V_x \cap V_y = \emptyset)$,
3. (Spanning) $\bigcup_{V_x \in \mathbf{p}} V_x = V$.

Nearest neighbor subgraph The k-nearest-neighbor subgraph of G is a subgraph $G_{nn} = (V, E_{nn})$ where an edge $ij \in E_{nn}$ only if j is one of the k nearest vertices to i or i is one of the k nearest vertices to j:

$$E_{nn} = \{ij \in E : (i, j) \in E_{dnn} \vee (j, i) \in E_{dnn}\}.$$

Nearest neighbor digraph The k-nearest-neighbor digraph of G is a directed subgraph $G_{dnn} = (V, E_{dnn})$ where an arc $(i, j) \in E_{dnn}$ only if j is one of the k nearest vertices to i. Rigorously, for a vertex i, let $i_{(\ell)}$ denote the vertex j that corresponds to the ℓ^{th}

smallest value of $\{c_{ij} : ij \in E\}$ (where ties are broken arbitrarily but deterministically):

$$c_{ii_{(1)}} \leq c_{ii_{(2)}} \leq \cdots,$$

then:

$$E_{dnn} = \left\{(i,j) : ij \in E \wedge j \in \{i_{(\ell)}\}_{\ell=1}^{k}\right\}.$$

B. Proof of NP-hardness

We prove NP-hardness by showing that a partition problem considered by Kirkpatrick and Hell [38] can be reduced to the bottleneck threshold blocking problem. For an arbitrary graph $G = (V, E)$, the PART $[\{K_t : t \geq k\}]$-problem asks whether there exists a set of subgraphs $\mathbf{G} = \{G_1 = (V_1, E_1), \cdots, G_m\}$ of G such that $\{V_1, \ldots, V_m\}$ is a partition of V and each subgraph $G_x \in \mathbf{G}$ is isomorphic to a complete graph with k or more vertices, $K_{t \geq k}$. This problem is NP-complete for any $k \geq 3$ [38].

Theorem 5 *Let $\epsilon > 0$. The bottleneck threshold blocking problem with minimum block size $k \geq 3$ does not allow for a polynomial time $(2-\epsilon)$-approximation algorithm unless $\mathbf{P} = \mathbf{NP}$.*

Proof. Consider an instance of the PART $[\{K_t : t \geq k\}]$-problem on a graph $G = (V, E)$. Create a weighted complete graph, $H = (V, E')$, that shares the vertex set with G. For all edges $ij \in E'$, let the corresponding costs be:

$$c_{ij} = \begin{cases} 1, & \text{if } ij \in E, \\ 2, & \text{if } ij \notin E. \end{cases} \tag{16}$$

These costs satisfy the triangle inequality. Note that for any cost ratio higher than two between the two types of edges, the triangle inequality will be violated for some input graph.

We consider solving the bottleneck threshold blocking problem on H. We show that the minimum maximum within-block cost $\lambda = 1$ if and only if PART $[\{K_t : t \geq k\}]$ is true. Since $\lambda \in \{1, 2\}$, it follows that $\lambda = 2$ if and only if PART $[\{K_t : t \geq k\}]$ is false. Hence, any $(2-\epsilon)$-approximation algorithm will produce a blocking with maximum within-block distance of 1 if and only if PART $[\{K_t : t \geq k\}]$ is true, and hence, can be used to solve PART $[\{K_t : t \geq k\}]$. Thus, such an algorithm terminates in polynomial time only if $\mathbf{P} = \mathbf{NP}$.

Suppose $\lambda = 1$. Consider an optimal blocking $\mathbf{b}^* = \{V_1, \ldots, V_m\}$ of H and the set of subgraphs induced on the input graph by the components of the blocking:

$$\mathbf{G} = \{G[V_1], G[V_2], \cdots, G[V_m]\}. \tag{17}$$

All $ij \in E'(\mathbf{b}^*)$ have cost $c_{ij} = 1$, and so, must exist in G. Thus all blocks induced on G must be isomorphic to a complete graph. Furthermore, as H and G share the same vertex set and $\mathbf{b}^*$ partitions H, $\mathbf{G}$ partitions G. It follows that $\mathbf{G}$ is a valid $\{K_t : t \geq k\}$-partition and PART $[\{K_t : t \geq k\}]$ is true.

Suppose PART $[\{K_t : t \geq k\}]$ is true. Let $\mathbf{G} = \{G_1 = (V_1, E_1), \cdots, G_m = (V_m, E_m)\}$ denote a $\{K_t : t \geq k\}$-partition. Consider a blocking constructed from the vertex sets of $\mathbf{G}$: $\mathbf{p} = \{V_1, \cdots, V_m\}$. As all subgraphs in $\mathbf{G}$ are complete, the within-block edges are exactly $E_1 \cup E_2 \cup \cdots \cup E_m$. As these edges exist in G, the corresponding edge costs in H are one. Thus the maximum edge cost in the blocking given by $\mathbf{b}$ is $\lambda = 1$. □

C. Proof of Lemma 1

Lemma 1 states that:

1. For any $i \notin \mathbf{S}$, there exists no two seeds both adjacent to i in G_{nn}:

$$\forall j, \ell \in \mathbf{S}, ij \notin E_{nn} \vee i\ell \notin E_{nn}. \tag{18}$$

2. For any $i \notin \mathbf{S}$, let $j \in \mathbf{S}$ denote the seed of the block that i is assigned to. There exists a walk of two or fewer edges from i to j in G_{nn}:

$$ij \in E_{nn} \ \vee \ \exists \ell, \ i\ell, \ell j \in E_{nn}. \tag{19}$$

If the first statement is false, then there exists a walk of two edges between j and ℓ, going through i. This implies that j and ℓ are adjacent in G_{nn}^2, but this contradicts the definition of $\mathbf{S}$ as an independent set of G_{nn}^2.

The second statement follows from how vertices are assigned to blocks. Vertices assigned in the third step of the algorithm are, by construction, adjacent to their block seeds in G_{nn}. Vertices assigned in the fourth step are adjacent in G_{nn} to a vertex in their block that is assigned in the third step. This vertex (i.e., ℓ) is in turn adjacent to the block seed, forming a walk of two edges from i to j. Every vertex unassigned after the third step must be adjacent to at least one vertex in the closed neighborhood of a block seed. Otherwise, there is no walk of two edges from that unassigned vertex to a block seed; hence, that unassigned vertex is independent of all block seeds in G_{nn}^2, contradicting the maximality of $\mathbf{S}$.

D. Subroutine for the second step of the algorithm

The obvious way to derive the seeds is to construct the second power of the $(k-1)$-nearest neighbor subgraph and then find a maximal independent set in this graph using conventional algorithms. The second power will,

however, not always to be a sparse matrix, even when k is fixed, and this procedure will therefore increase time and space complexity. Using the subroutine presented in this section gives a complexity of $O(kn)$ independently of how G_{nn} is structured.

As discussed in the paper, we store the $(k-1)$-nearest neighbor subgraph using edge lists for all vertices. This allows for access to a vertex' edges with constant time complexity. Consider the following procedure that takes these edge lists as input:

1. Initialize $\mathbf{S}$ and $\mathbf{A}$ to the empty set.
2. Let i iterate over all vertices in V:
   a) If the closed neighborhood of i contains any vertex in $\mathbf{A}$, $\mathrm{N}_{G_{nn}}[i] \cap \mathbf{A} \neq \emptyset$, continue to the next iteration.
   b) Else, set $\mathbf{S} \leftarrow \mathbf{S} \cup i$ and $\mathbf{A} \leftarrow \mathbf{A} \cup \mathrm{N}_{G_{nn}}[i]$.

When this routine terminates, the set $\mathbf{S}$ will be a maximal independent set in G_{nn}^2. This can be shown with a proof by contradiction. Note that, at any iteration of the loop, $\mathbf{A}$ contains all vertices in $\mathbf{S}$ and all of their adjacent vertices.

Suppose that $\mathbf{S}$ is not independent. Two $i, j \in \mathbf{S}$ that are adjacent in the second power must then exist. That is, either $ij \in E_{nn}$ or, for some ℓ, we have $i\ell, \ell j \in E_{nn}$. As vertices are added to $\mathbf{S}$ sequentially, a state must have existed such that (with arbitrary labeling):

$$i \in \mathbf{S}, \qquad j \notin \mathbf{S}, \quad \text{and} \qquad \mathrm{N}_{G_{nn}}[j] \cap \mathbf{A} = \emptyset. \tag{20}$$

If not, j would not have been added to $\mathbf{S}$ when its iteration came. When i was added to $\mathbf{S}$, it and all of its neighbors were added to $\mathbf{A}$. This implies that if $ij \in E_{nn}$, then j is in $\mathbf{A}$, and if ℓ exists so that $i\ell, \ell j \in E_{nn}$, then $\ell \in \mathbf{A}$. Subsequently, such state is not possible and $\mathbf{S}$ must be independent.

Suppose that $\mathbf{S}$ is not maximal. There must then exists an $i \in V \setminus \mathbf{S}$ which is not adjacent to any vertex in $\mathbf{S}$ in G_{nn}^2 when the algorithm terminates. This implies that $\mathrm{N}_{G_{nn}}[i] \cap \mathbf{A} = \emptyset$ is true. As vertices only are added to $\mathbf{A}$, this must also have been true throughout the run of the algorithm. But if $\mathrm{N}_{G_{nn}}[i] \cap \mathbf{A} = \emptyset$ always was true, i would have been added to $\mathbf{S}$ in its iteration. The algorithm can therefore not terminate with such a state and $\mathbf{S}$ must be maximal.

To show complexity, note that in the worst case, one has to check whether each vertex' closed neighborhood is in $\mathbf{A}$. As there can be at most $2(k-1)n$ entries in the edge lists, there are at most $O(kn)$ checks needed. By storing set membership in a random access data structure, set queries are done in $O(1)$ time, which gives a complexity of $O(kn)$ for the complete subroutine.

A straightforward way to incorporate the second heuristic improvement discussed in the paper is to change the order which the subroutine iterates over the vertices.

E. Proof of Theorem 4

Theorem 4 states that any subset of blocks from a blocking constructed by the algorithm, $\mathbf{b}_{sub} \subseteq \mathbf{b}_{alg}$, will be approximately optimal with respect to the blocking problem only containing vertices in the blocks of $\mathbf{b}_{sub}$. Formally, define $V_{sub} = \bigcup_{V_x \in \mathbf{b}_{sub}} V_x$ as the set of all vertices in the blocks of $\mathbf{b}_{sub}$. Let λ_{sub} denote the maximum edge cost in an optimal blocking of V_{sub}. Theorem 4 states that:

$$\max_{ij \in E(\mathbf{b}_{sub})} c_{ij} \leq 4\lambda_{sub}. \tag{21}$$

Let G_{sub} denote the complete graph on V_{sub}. Recall that $G_{nn} = (V, E_{nn})$ is the $(k-1)$-nearest neighbor subgraph of G. Let $G_{ind} = (V_{sub}, E_{ind}) = G_{nn}[V_{sub}]$ denote the graph induced by V_{sub} on G_{nn}. That is, E_{ind} contains all edges in E_{nn} between vertices that are in V_{sub}. Finally, let $G_{sub,nn} = (V_{sub}, E_{sub,nn})$ denote the $(k-1)$-nearest neighbor subgraph of G_{sub}.

Observe that $E_{ind} \subset E_{sub,nn}$: if $i, j \in V_{sub}$ and j is one of the $(k-1)$ nearest neighbors of i in G, then it also must be one of the $(k-1)$-nearest neighbors of i in $G_{sub} \subset G$. From Lemma 2, we have that $\forall ij \in E_{sub,nn},\ c_{ij} \leq \lambda_{sub}$, and so, $\forall ij \in E_{ind},\ c_{ij} \leq \lambda_{sub}$.

As implied by Lemma 1, there is a walk in G_{nn} of four or fewer within-block edges between any two vertices contained in the same block in $\mathbf{b}_{alg}$. As G_{ind} retains all within-block edges in G_{nn} of the blocks in $\mathbf{b}_{sub}$, there is a walk of four or fewer between any two vertices in the same block also in G_{ind}. Theorem 3 can, therefore, be applied using E_{ind} in place of E_{nn}. This bounds all within-block edge costs in $\mathbf{b}_{sub}$ by $4\lambda_{sub}$.

F. Algorithm using nearest-neighbor digraphs

The approximation algorithm presented in the paper tends to construct too large blocks. A slightly modified version using a k-nearest-neighbor digraph will often produce better blockings. In particular, to avoid collisions, one only needs to ensure that no seed adds a vertex to its block that is either a seed itself or which some other seed want to add to its block. The undirected version enforce that no non-seed vertex wants to add a seed if it was to be a seed—an unnecessary requirement which the digraph avoids.

Redefine $\mathrm{N}_G[i]$ to be the directed version of a closed neighborhood:

$$\mathrm{N}_G[i] \equiv \{j \in V : (i, j) \in E\} \cup i. \tag{22}$$

Consider the following algorithm that takes the graph, G, describing the experimental sample as input:

1. Construct a $(k-1)$-nearest neighbor digraph of G so that an arc (i,j) exists if j is among the $(k-1)$ nearest neighbors of i. Denote that graph $G_{dnn} = (V, E_{dnn})$.
2. Find a set of vertices, $\mathbf{S}$, so that:
   a) There exist no $i,j \in \mathbf{S}$ so that $(i,j) \in E_{dnn}$.
   b) There exist no $i,j \in \mathbf{S}$ and $\ell \in V \setminus \mathbf{S}$ so that $(i,\ell) \in E_{dnn}$ and $(j,\ell) \in E_{dnn}$.
   c) Adding any $i \in V \setminus \mathbf{S}$ to $\mathbf{S}$ would violate either (a) or (b).
3. For each $i \in \mathbf{S}$ form a block with that vertex and all of its adjacent vertices: $V_i = \mathrm{N}_{G_{dnn}}[i]$.
4. Assign vertices that are yet unassigned to a block that contains one of their nearest assigned neighbors. That is, for an unassigned vertex i assign it to any V_x such that $\exists\, j \in V_x : (i,j) \in E_{dnn}$.

An illustration of this algorithm, with a comparison to the undirected version, is given in Figure 3.

The resulting blocking is approximately optimal for the same reasons as for the original algorithm. The second step ensures that no two seeds have outward-pointing arcs to the same vertex. This makes the blocking disjoint and satisfying the size requirement. The second step also ensures that all vertices are at most two arcs (of any directionality) away from their seeds and, following the same proof as in the paper, all vertices are at distance of at most 2λ from their seeds. By the triangle inequality this proves approximate optimality.

The $(k-1)$-nearest neighbor digraph will have exactly $(k-1)n$ arcs and can thus be stored in $O(kn)$. With only trivial changes the steps of this algorithm can be done in the same way as in the original version, thus preserving complexity. In particular, the subroutine presented in the previous section can still be used to complete the second step when $\mathrm{N}_G[i]$ is redefined to the directed version as above.

G. Greedy threshold algorithm

The third heuristic improvement discussed in the paper is to split blocks that contain $2k$ or more vertices into smaller blocks. Any algorithm can be used to do this split as the approximation algorithm ensures that all edges satisfy the optimality bound. One approach would be to use the approximation algorithm once more. However, as the large blocks are a consequence of the structure of the nearest neighbor subgraph, the algorithm will often return the block unchanged.

The greedy algorithm presented in this section seems to perform well in many cases where splitting is desired. The algorithm's input is an arbitrary valid threshold blocking, $\mathbf{b}$, and it returns a blocking where no edge is greater than in the original blocking and no block contains more than $2k-1$ vertices.

Figure 4 provides pseudocode that describes the algorithm. Informally, it searches among the existing blocks to find a splittable block, i.e., one that contains $2k$ or more vertices. In such blocks, it finds the two vertices farthest apart and construct two new blocks based on them. Each of the two vertices picks enough vertices from the original block to fulfill the size requirement, and remaining vertices are assigned to the vertex which is closest. This is repeated until no block contains $2k$ or more vertices.

As a blocking with a single block, $\mathbf{b} = \{\{1, \cdots, n\}\}$ is a valid threshold blocking, this algorithm can be use to block the whole sample as well.

H. Simulation study

H.1. Implementation and hardware

The approximation algorithm is implemented in the `R` and `C++` languages using the `CHOLMOD` library for operations on sparse matrices [39] and the `ANN` library for nearest neighbor searching written by David M. Mount and Sunil Arya. The source code is publicly available at an online code repository and can otherwise be obtained on request.

The simulations where run on the SAVIO computational cluster at UC Berkeley using Intel Xeon E5-2670 processors for which each core is clocked at 2.5 GHz. Each round of the simulations was allocated a single core, largely reflecting the performance of an ordinary desktop computer, and was limited to a maximum of 48 GB random access memory.

H.2. Estimation methods

After each algorithm has derived their blockings, treatment was assigned independently across blocks using *block-balanced complete randomization* [37]. For a block V_x with an experiment with t treatment conditions, $\lfloor |V_x|/t \rfloor$ units are randomly assigned to each of the treatments. Then $|V_x| \pmod t$ treatment conditions are picked at random and randomly assigned to the units still without treatments. When t divides all blocks, e.g., when using fixed-sized blockings, this randomization scheme is equivalent to ordinary complete randomization within the blocks.

For methods that use blocking, the block-size weighted difference-in-means estimator was used [37]. This estimator first estimates the treatment effect separately within

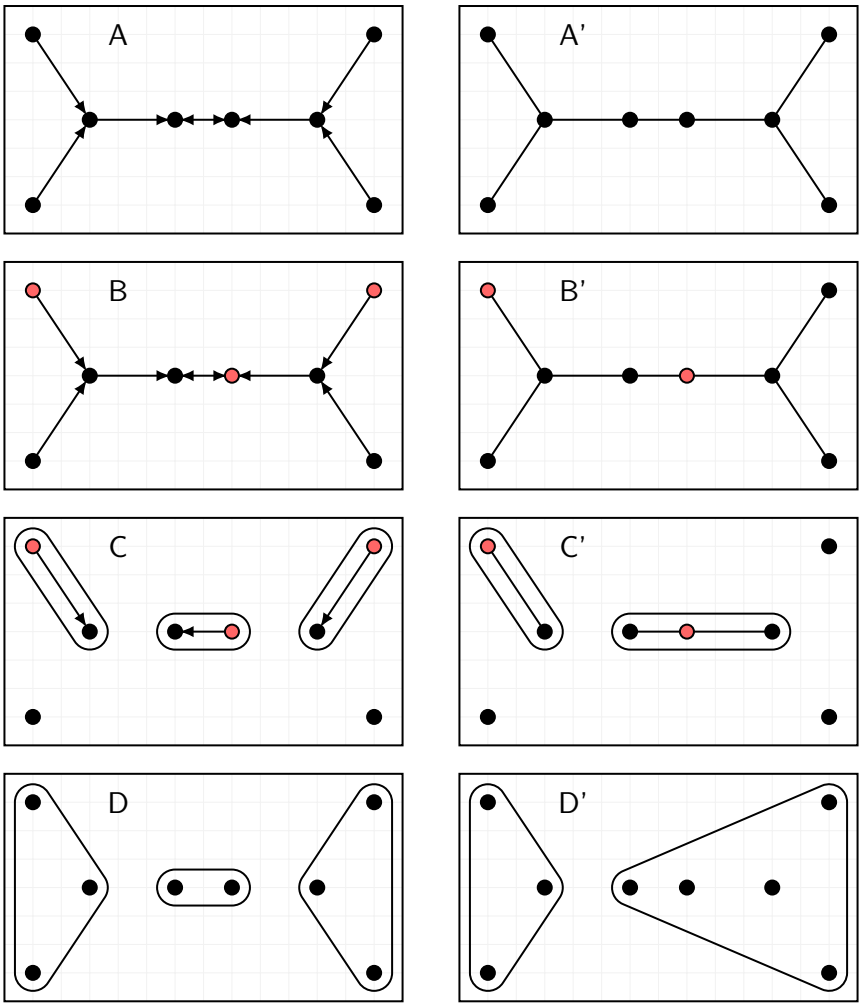

Figure 3: An illustration of the directed version of the blocking algorithm with a comparison with the undirected version. (A) The directed algorithm creates a nearest neighbor digraph by drawing an arc from each vertex to its closest neighbor. (A') The undirected algorithm draws the same graph but disregard the directionality of the edges. (B) The directed version finds seeds (red vertices) so that no two seeds points towards the same vertex. (B') As the undirected version disregard the direction of the edges, it forces all seeds to be at least on distance of three edges and thereby misses one possible seed. (C) Blocks are formed with the seeds' closest neighbors. (C') Blocks are formed both with the seeds' closest neighbors and vertices that have the seeds as their closest neighbors. (D,D') Remaining vertices are assigned to the block containing their closest neighbor.

G: weighted graph describing a sample
$\mathbf{p}$: arbitrary valid blocking of G

ThresholdGreedy($\mathbf{p}$, G):
  while $V_x \in \mathbf{p} : |V_x| \geq 2k$:
    $\mathbf{p} \leftarrow \mathbf{p} \setminus V_x$
    $i, j \leftarrow \arg\max_{i,j \in V_x} c_{ij}$
    $V_y \leftarrow \mathrm{NN}_{V_x \setminus \{j\}}[i]$
    $V_z \leftarrow \mathrm{NN}_{V_x \setminus V_y}[j]$
    foreach $\ell \in V_x \setminus (V_y \cup V_z)$:
      if $c_{\ell i} \leq c_{\ell j}$:
        $V_y \leftarrow V_y \cup \ell$
      else:
        $V_z \leftarrow V_z \cup \ell$
    $\mathbf{p} \leftarrow \mathbf{p} \cup V_y \cup V_z$
  return $\mathbf{p}$

Figure 4: Greedy threshold blocking algorithm. $\mathrm{NN}_{V'}[i]$ denotes the union of i and i's $k-1$ nearest neighbors in the graph induced on G by vertices V'.

each block, and derives an estimate for the sample by taking a weighted average based on the sizes of the blocks. When a fixed-sized blocking method is used, this estimator is equivalent to the ordinary difference-in-means estimator.

Let $\mathbf{T}$ and $\mathbf{C}$ collect all units in the two treatment conditions for which the contrast is of interest. Let $\hat{\beta}_{V_x}$ be the estimated treatment effect within block V_x using the ordinary difference-in-means estimator:

$$\hat{\beta}_{V_x} = \sum_{i \in V_x \cap \mathbf{T}} \frac{y_i}{|V_x \cap \mathbf{T}|} - \sum_{i \in V_x \cap \mathbf{C}} \frac{y_i}{|V_x \cap \mathbf{C}|}. \tag{23}$$

The estimated treatment effect for the complete sample, $\hat{\beta}$, is then given by averaging over all blocks, weighted by their size:

$$\hat{\beta} = \sum_{V_x \in \mathbf{b}} \frac{|V_x|}{n} \hat{\beta}_{V_x}. \tag{24}$$

The ordinary least squares estimator investigated in addition to the blocking methods adjusts for imbalances in the covariates linearly. That is, the estimator of the contrast between treatments $\mathbf{T}$ and $\mathbf{C}$ (presuming exactly two treatment conditions) is given by the $\hat{\beta}$ that solves the following optimization problem:

$$\underset{\{\hat{\alpha},\hat{\beta},\hat{\gamma}_1,\hat{\gamma}_2\}}{\arg\min} \sum_{i=1}^{n} \Big(y_i - \hat{\alpha} - \hat{\beta}\, \mathbf{1}[i \in \mathbf{T}] - \hat{\gamma}_1 x_{i1} - \hat{\gamma}_2 x_{i2} \Big)^2. \tag{25}$$

Table 3: Run time and memory use for blocking algorithms by sample size: two-dimensional input data

Algorithm	10^2	10^3	10^4	2×10^4	5×10^4	10^5	10^6	10^7	10^8
Panel A: Run time in seconds									
Approximation algorithm	0.2	0.2	0.2	0.2	0.3	0.5	4.0	47.8	657.4
Directed version	0.2	0.2	0.2	0.2	0.3	0.5	3.8	45.2	618.7
Improvements 1-3	0.2	0.2	0.2	0.2	0.4	0.6	5.7	68.1	935.0
Improvements 1-4	0.2	0.2	0.2	0.2	0.4	0.7	6.5	78.4	1,109.0
Fixed greedy	0.2	4.4	1,078.9	7,119.8					
Threshold greedy	0.2	1.1	284.3	1,276.1					
Non-bipartite matching	2.2	4.3	377.1	2,840.1					
Panel B: Memory use in megabytes									
Approximation algorithm	29	30	31	30	33	34	174	1,467	14,514
Directed version	29	30	31	30	32	34	174	1,467	14,514
Improvements 1-3	30	30	31	32	36	42	229	2,152	21,343
Improvements 1-4	30	30	31	32	36	43	236	2,230	22,122
Fixed greedy	30	58	2,891	11,490					
Threshold greedy	30	136	10,711	42,745					
Non-bipartite matching	52	135	7,705	30,666					

Note: Blank cells indicate that the corresponding algorithm uses more than 48 GB of memory or does not successfully terminate within three hours (effective time) for the corresponding sample size.

Table 4: Run time and memory use for blocking algorithms by sample size: five-dimensional input data

Algorithm	10^2	10^3	10^4	2×10^4	5×10^4	10^5	10^6	10^7	10^8
Panel A: Run time in seconds									
Approximation algorithm	0.2	0.2	0.2	0.3	0.8	1.6	19.7	222.4	2,794.0
Directed version	0.2	0.2	0.2	0.3	0.8	1.6	19.4	220.4	2,764.3
Improvements 1-3	0.2	0.2	0.3	0.4	0.8	1.8	21.7	244.6	3,127.9
Improvements 1-4	0.2	0.2	0.3	0.4	0.9	2.0	25.4	289.9	3,741.5
Fixed greedy	0.2	4.6	1,043.6	7,121.5					
Threshold greedy	0.2	1.1	306.6	1,351.3					
Non-bipartite matching	2.2	4.6	665.5	5,085.7					
Panel B: Memory use in megabytes									
Approximation algorithm	30	30	30	32	34	43	247	2,269	22,525
Directed version	30	30	30	32	34	41	247	2,269	22,525
Improvements 1-3	30	30	31	33	37	52	316	2,993	29,742
Improvements 1-4	30	30	31	32	37	52	325	3,077	30,577
Fixed greedy	30	58	2,892	11,492					
Threshold greedy	30	136	10,702	42,748					
Non-bipartite matching	52	126	7,628	33,399					

Note: Blank cells indicate that the corresponding algorithm uses more than 48 GB of memory or does not successfully terminate within three hours (effective time) for the corresponding sample size.

Table 5: Run time and memory use for blocking algorithms by sample size: ten-dimensional input data

Algorithm	10^2	10^3	10^4	$2{\times}10^4$	$5{\times}10^4$	10^5	10^6	10^7	10^8
Panel A: Run time in seconds									
Approximation algorithm	0.2	0.2	0.7	1.8	7.7	22.3	386.3	4,970.6	
Directed version	0.2	0.2	0.7	1.8	7.8	22.1	394.6	5,129.0	
Improvements 1-3	0.2	0.2	0.8	1.8	7.8	22.4	380.5	5,006.5	
Improvements 1-4	0.2	0.2	0.8	2.0	8.7	25.4	452.2	6,013.7	
Fixed greedy	0.2	4.4	1,064.7	7,102.8					
Threshold greedy	0.2	1.1	283.7	1,355.8					
Non-bipartite matching	2.2	5.0	776.2	6,026.1					
Panel B: Memory use in megabytes									
Approximation algorithm	30	30	32	34	39	54	361	3,451	
Directed version	30	30	32	34	39	52	361	3,451	
Improvements 1-3	30	30	32	35	44	61	437	4,236	
Improvements 1-4	30	30	33	35	44	65	447	4,323	
Fixed greedy	30	58	2,894	11,495					
Threshold greedy	30	136	10,704	42,750					
Non-bipartite matching	52	133	7,582	30,156					

Note: Blank cells indicate that the corresponding algorithm uses more than 48 GB of memory or does not successfully terminate within three hours (effective time) for the corresponding sample size. No algorithm terminates within three hours with 100 million ten-dimensional data points. In test-runs, the original version of the approximation algorithm terminates within 24 hours using approximately 34 GB of memory in that setting.

Table 6: Performance of blocking algorithms when $k = 4$ by sample size: maximum and average within-block distances relative to the approximation algorithm and average block size

	Max. within-block distance			Avg. within-block distance			Avg. block size		
Algorithm	10^2	10^3	10^4	10^2	10^3	10^4	10^2	10^3	10^4
Approximation algorithm	1.000	1.000	1.000	1.000	1.000	1.000	6.11	6.10	6.10
Directed version	0.923	0.919	0.916	0.916	0.912	0.911	5.63	5.60	5.59
Improvements 1-3	0.816	0.827	0.839	0.803	0.798	0.797	4.96	4.95	4.95
Improvements 1-4	0.780	0.759	0.739	0.804	0.799	0.798	4.87	4.87	4.87
Fixed greedy	2.263	6.056	16.858	0.899	0.904	0.894	4.00	4.00	4.00
Threshold greedy	0.958	1.008	1.041	0.979	1.005	1.015	5.03	5.01	5.01

Table 7: Root mean square error relative to the approximation algorithm when $k = 4$ by sample size

Method	10^2	10^3	10^4
Approximation algorithm	1.000	1.000	1.000
Directed version	0.941	0.961	0.978
Improvements 1-3	0.854	0.891	0.966
Improvements 1-4	0.846	0.892	0.957
Fixed greedy	1.226	1.421	1.203
Threshold greedy	1.031	1.057	1.010
Unadjusted	3.586	10.719	19.079
OLS adjustment	1.384	4.085	7.290

Table 8: Root mean square error when $k = 2$ by sample size, without normalization

Method	10^2	10^3	10^4
Approximation algorithm	0.7182	0.0921	0.0214
Directed version	0.6992	0.0909	0.0213
Improvements 1-3	0.6685	0.0884	0.0212
Fixed greedy	1.1557	0.1471	0.0246
Threshold greedy	0.8672	0.1055	0.0222
Non-bipartite matching	0.6837	0.0874	0.0210
Unadjusted	4.3754	1.3954	0.4424
OLS adjustment	1.6891	0.5317	0.1688

Table 9: Root mean square error when $k = 4$ by sample size, without normalization

Method	10^2	10^3	10^4
Approximation algorithm	1.2196	0.1302	0.0232
Directed version	1.1470	0.1252	0.0227
Improvements 1-3	1.0420	0.1160	0.0224
Improvements 1-4	1.0320	0.1162	0.0222
Fixed greedy	1.4946	0.1851	0.0279
Threshold greedy	1.2579	0.1377	0.0234
Unadjusted	4.3732	1.3961	0.4425
OLS adjustment	1.6878	0.5320	0.1691