Künstl Intell (2018) 32:9–17
DOI 10.1007/s13218-017-0517-5

TECHNICAL CONTRIBUTION

Big data algorithms beyond machine learning

Matthias Mnich[1,2]

Received: 15 April 2017 / Accepted: 7 September 2017 / Published online: 24 October 2017

Abstract The availability of big data sets in research, industry and society in general has opened up many possibilities of how to use this data. In many applications, however, it is not the data itself that is of interest but rather we want to answer some question about it. These answers may sometimes be phrased as solutions to an optimization problem. We survey some algorithmic methods that optimize over large-scale data sets, beyond the realm of machine learning.

Keywords Big data algorithms · Large-scale optimization · Kernelization · Dynamic algorithms

Mathematics Subject Classification 05C85 · 90B10 · 90C27

1 Introduction

The arrival and availability of large data sets in many areas of society have been hot trends over the last couple of years, with more to come. This “age of big data” is often described in terms of buzzwords starting with V, the three most cited ones being *volume*, *variety*, and *velocity*. Here, volume refers to massive amounts of data which must be handled. Variety refers to the many different forms of data which are stored. Velocity alludes to the increased speed at which data changes. In each of these V’s, big data sets differ from traditional ones, and these differences pose challenges which require the development of new algorithmic methods to analyse such data sets and solve optimization tasks about them. From these new methods we require several fundamental properties, such as *scalability* and *robustness*. The main motivations for these requirements are that we expect data sets to grow even more (and even faster) then until now, while at the same time the half-life of data is expected to decrease at similar rates. We also seek methods that make them easy to implement. Other V’s that have been used to characterize big data include *veracity* of data, and *value*. Veracity summarizes the situation that data may be incorrect or uncertain, and value reflects the increasing importance of data for business successes. We made this deliberate choice of definition for big data, being fully aware that other definitions exist and that a general consensus on the “right” definition seems to be lacking, as vividly illustrated by Ward and Barker’s “survey of big data definitions” [57].

In this survey, we take an algorithmic viewpoint to big data, and highlight some methods which can contribute to tackle the challenges associated with each of the V’s. By no means do we claim our survey to be exhaustive; rather, we accept a personal bias (towards multivariate algorithms). In particular, as reflected by the title of this survey, we skip methods addressing the exciting field of machine learning techniques for big data—if only to avoid competing with specific surveys for that topic [48, 58].

1.1 Addressing the V’s of big data

As starting point for our work, we take the V that stands for *value*. Our motivation for this choice is that we only

M. M. was supported by ERC Starting Grant 306465 (BeyondWorstCase).

✉ Matthias Mnich
mmnich@uni-bonn.de

[1] Department of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands

[2] Universität Bonn, Institut für Informatik, Friedrich-Ebert-Allee 144, 53113 Bonn, Germany

consider it worthwhile to invest significant amounts of financial, technical and human resources into collecting, storing and processing big data sets if there is a chance for us of extracting additional value from them, compared to other choices. We formalize thesearch for value in data sets in terms of optimization problems, from whose outcome or optimal solution we expect to make decisions that are feasible with respect to the bounded resources and constraints we impose on them. We restrict ourselves to *discrete optimization problems*, motivated by the fact that many economic applications require discrete decisions as well as that this is where our expertise lies.

1.1.1 Organization

Our survey is organized as follows. The first item on our menu is the aspect of variety, where our preference of algorithmic techniques lies with multivariate algorithms. The details are given in Sect. 2.

Data reduction algorithms address the volume challenge. They reduce and simplify the large parts of the data set which obscure the core parts which are relevant for the actual answer. Our pick here lies with linear-time algorithms targeted at volume reduction, which we discuss in Sect. 3.

Dynamic algorithms take care of the velocity aspect. Once we solved an optimization problem on a large data set, when there are frequent updates, we do not want to recompute the answers from scratch. In Sect. 4 we look at such algorithms handling data subject to temporal changes.

Regarding the aspect of veracity, we make the natural assumption that only a small part of the data is assumed to be corrupted. We then discuss techniques that preprocess the data in a way that reduces them to their "uncertain core"; see Sect. 5.

We conclude in Sect. 6.

2 Facing variety: multivariate algorithms

The variety aspect of big data means that the data sets we keep originate from very different sources, and come with very different characteristics. From this data we then want to extract relevant information by answering a query. The data can be seen as unstructured, of having very many dimensions. A typical scenario faced in applications is that the data is simply stored as documents, and we want to cluster the documents by topic (allowing documents to belong to multiple topics). Nielsen [45] models this problem by a large keyword-by-document incidence matrix I with K rows and D columns, wherein entry I_{kd} stores the number of occurrences of keyword k in document d. The goal is then to factor the matrix I as $I = WH$, where W is a non-negative matrix with K rows and $R < \min\{K, D\}$ columns and H is a non-negative matrix with R rows and D columns. The goal is to find the smallest *non-negative rank* R for which such a decomposition is possible. This approach has had a long history, and it often provides meaningful results: the columns of W can be interpreted as R exemplary documents with appropriate keyword proportions whereas the columns of H can be seen as a vector of weights representing each document in terms of the exemplary ones. So the exemplary documents can be seen as a sparse representation of the large data set of unstructured documents. The success of the approach is also due to the fact that despite the problem's NP-hardness, many fast algorithms exist. A crucial *reason* why these algorithms work so well despite the problem's (worst-case) intractability is that they, in a more or less obvious way, *do* identify and harness the structure of the input data—but this structure may not be obvious to the practitioner due to the high variety (and potentially also volume) of the data. To highlight this structure formally, and exploit it algorithmically, it is sensible to measure an algorithm's performance not just in terms of the input *size*—which we assume to be large—but also by other relevant *parameters*. For instance, Arora et al. [6] and Moitra [43] give algorithms that find such factorizations and which are efficient for small values of R, which is exactly the scenario encountered in applications. Moitra's algorithm [43] runs in time $(KD)^{O(R^2)}$—this *multivariate problem analysis* shows that efficient algorithms can indeed exist for the data sets which are encountered in practice. Likewise, the general intractability (measured by NP-hardness) can also be refined to include the relevant parameters—showing in this case, that an algorithm of run time $(KD)^{o(R)}$ is highly unlikely to exist. A formal way to understand a problem's multivariate complexity, and to take advantage of it algorithmically, is by means of Parameterized Complexity Theory. In there, problem instances are measured in terms of their size n as well as an integer parameter k that measures the optimal solution size, the treewidth or genus of the input graph, or any similar structural property. The desire is to come up with *fixed-parameter algorithms* that restrict the superpolynomiality (which is expected if $\mathsf{P} \neq \mathsf{NP}$) to terms of k only, and thus run in time $f(k)n^c$ for some computable function f and some fixed constant c independent of k and n. We refer to the recent book by Cygan et al. [15] for further background.

Another approach to the just-discussed task of *relevant information retrieval* leads to so-called combinatorial feature selection problems. Given a set S of multidimensional objects, the goal is to select a subset K of relevant dimensions (or features) such that some desired property Π holds for the set S restricted to K. Depending on Π, the goal could be to either maximize or minimize the size of the subset K. Coming back to our example of unstructured text collected in documents, a simple vector-space model views a document as a set of words and phrases—called "features"—that it contains. The algorithmic task is to reduce the feature set—which can be extremely large—to a small subset; this task is called *feature selection*. The main point is of course to choose the criteria according to which the small subset is chosen, so as to ease the processing and data management tasks, to reduce the risk of overfitting, and remove noisy features. Those criteria heavily depend on the precise optimization problem to solve. An example are "dimension reduction problems", where one is given a large set of objects represented by vectors in a high-dimensional space and one seeks a representation of the objects in a lower-dimensional space that still "explains"; formally, this means to pick a small subset of relevant dimensions such that all objects are still distinct in this lower-dimensional space. Charikar et al. [13] mention the "hidden clusters problem" in which one is given set of vectors in high-dimensional space that are known a priori to "cluster well", but, due to the presence of "noisy" dimensions the clustering is destroyed when all dimensions are considered together. Complementary to the previous example, one now wishes to remove as few noisy dimensions as possible so that the data clusters well in all the remaining dimensions.

As has been frequently observed (see, e.g., [13]), feature selection problems—when formalized as finding a smallest set of dimensions to retain or remove to preserve some property of the original data set of vectors—turn out to be computationally hard (intractable) even in very simple settings, for essentially all non-trivial properties to satisfy, as it is quite non-obvious which dimensions to pick. But again we do not want to perform combinatorial feature selection in general but only for the specific data sets available to us. So once more, defying the data set's large variety and the task's general intractability, we employ algorithmic tools that unravel for us the structures in the data pertinent to the query we wish to answer. Froese et al. [25] take this approach for the specific task of identifying a small set of documents from a large corpus which still represent the entire set of keywords (or features), thus avoiding a loss of information while compressing the corpus as much as possible. Formally, they seek a subset C of as few columns as possible from the aforementioned keyword by document incidence matrix I such that all rows in the restricted submatrix $I_{|C}$ are distinct. With keywords coming from an alphabet Σ, they then provide efficient algorithms for this task whose run time is essentially only dominated by the alphabet size $|\Sigma|$ and the number $|C|$ only, or the columns' maximum pairwise distinctness measured in terms of Hamming distance. Thus, if the corpus has many similar documents and covers only a few topics, we can expect to extract a small set of relevant documents reasonably fast.

Addressing variety of point sets has already been a long-term subject in the field of Computational Geometry. To reduce the variety, a popular approach is to cluster the points into a small set of "similar" points, where similarity can, for instance, be measured in terms of distance to a common center in a metric space. In typical clustering applications, such as pattern recognition or data mining, the point sets that must be clustered are often so large that any super-linear run times are prohibitive. In some applications, the data sets are even too large to fit into main memory—as a possible approach, sublinear-time algorithms have been suggested to cluster such data sets. Those algorithms generally select a sufficiently large subset of the input and then only cluster this subset. If the subset is chosen at random from a certain probability distribution, then one can often show that the clustering of the subset approximates the clustering of the entire input with high probability.

Two classical clustering problems that have been addressed under this regime are k-MEDIAN and k-MEANS. In the k-MEDIAN problem, one is asked to find a small set C of k center points from a large set V of points such that the centers in C minimize the sum of pairwise distances of all points in V to their nearest center, measured in some fixed metric μ. The k-MEANS problem has a very similar definition, except that the sum of squared pairwise distances is considered. Czumaj and Sohler [17] devised the following simple, yet extremely powerful strategy for these problems: pick a random sample S of points, run an approximation algorithm for the sample, and return the clustering induced by the solution for the sample. Their main contribution is a fine-grained analysis showing that this method yields approximation guarantees for k-MEANS, k-MEDIAN (and related problems) which are independent of the size of V.

If linear time is allowed as a run time, then better approximation guarantees can be achieved. Kumar et al. [38] give $(1 + \varepsilon)$-approximations with probability at least 1 / 2 for any $\varepsilon > 0$, which are obtained by algorithms that run in time $2^{(k/\varepsilon)^{O(1)}} \cdot O(d|V|)$ in d-dimensional metric spaces.

3 Facing volume: data reduction for big data

Having a large volume of data available clearly does not necessarily imply that it contains the relevant information. As already discussed, data often comes from a variety of sources and thus may be duplicated or corrupted. In order to clean the data and extract the relevant parts, quite often data reduction techniques are applied. Data reduction is part of a more general approach to optimization of big data sets known as preprocessing. Preprocessing consists of data integration, data cleaning, data transformation, data reduction, and finally discretization.

Data reduction is an ancient technique that has been already applied to data sets that were once considered large but are small by today's standards. The idea is to quickly detect parts of the input that can be removed, aggregated or simplified while maintaining the global structural properties of the entire input with respect to a given optimization problem. A prominent example of this success is delivered by commercial SAT solvers, which are able to decide feasibility of million-variable and clause formulas within seconds by pruning, which is to detect variables whose value can be fixed, and then remove them from the formula.

Despite their wide applicability in practical computing, it took quite a while before data reduction routines have been theoretically understood. The main goal is there to measure the power and limitations of such methods for given data sets. To make useful statements in this direction, again one needs to take into accounts structure within the data set, and not just its size. Then, using the aforementioned theory of Parameterized Complexity, the data reduction process can be formalized by the notion of *kernelization*. A kernelization algorithm compresses any given data set S in polynomial time to a potentially smaller data set S' such that all information relevant to a particular optimization question from S is preserved in S', while the size of S' depends only on some structures capturing particular aspects of the optimization question. Measuring the structures in terms of an integer k, we want to achieve that S' has size only $O(k^c)$ for some small constant c. Since the beginning of its systematic investigation in the early 2000's, kernelization has become a thorough object of study, with many deep results about algorithmic performance and complexity lower bounds having been published.

Yet, for large data sets occurring in big data applications, a polynomial run time for the data reduction procedure might be too slow. In this case, we are actually interested in kernelization algorithms that run in *linear time*. The research objective is to make the kernelization algorithm run in linear time while guaranteeing the so-called *kernel* S'—the reduced data set—to be as small as possible under plausible complexity-theoretic assumptions (to make c as small as possible). Until now, only very few such algorithms have been proposed, and there is no coherent picture yet which problems admit such algorithms and which problems do not. Examples of discrete optimization problems for which such feat has been achieved include Vertex Cover [44], Feedback Vertex Set [35], Dominating Set on planar graphs [29, 54], and Max-Cut [20], Edge Clique Cover [28], (Bi-)Cluster Editing [47], and Hitting Set [53]

As an example, we consider the Max-Cut problem. Formally, given an undirected graph G on m edges with non-negative edge weights w_{ij}, one seeks a bipartition $\{L, R\}$ of $V(G)$, such that the weight of the cut, defined as the sum of the weights on the edges connecting the two sets, is maximized. This problem has long served as a challenging test for researchers testing new methods. It has well known practical applications in several areas including statistical physics, VLSI design, classification, and social network analysis.

Due to its importance in statistics, much research is done on solving large-scale instances of Max-Cut for the purpose of big data applications. Recent approaches based on SDP relaxations can solve sparse instances of up to a million nodes in a few hours [26]. Also those approaches can benefit from an initial data reduction, as long as the data reduction is efficient. On the one hand, the celebrated Goemans–Williamson SDP relaxation gives a $0.878\ldots$-approximation, but its asymptotic run time is rather slow. On the other hand, if the size of the maximum cut is small—i.e., close to half the total number of edges—then a random bipartition of the graph gives a larger cut than guaranteed by the SDP in *linear time*. It is therefore natural to subtract the guaranteed value of $m/2 + (n - 1/4)$ in n-vertex connected graphs from the solution value; this is known as the Max-Cut Gain problem. For this task data reduction rules have been devised that preserve the gain; after their exhaustive application, fast SDP methods can be tested on the much reduced graph. The data reduction rules are simple, and in time $O(m)$ yield a graph G' of size only $O(k)$ where k is the gain. In fact, one only needs to compute $O(k)$ DFS/BFS trees, and performing this task has been well-engineered for very large data sets [1].

For a connected graph G and integer k, the first two rules consider those connected components C of the graph $G - v$ that is obtained by removing a cut vertex v from G for which C induces a clique. They remove such cliques from the graph, mark up to three vertices and then recurse. Another two rules are needed to handle "sparse" parts where certain non-edges appear; such parts are also removed, up to three vertices marked and the gain parameter k adjusted as usually there is a strictly positive gain in those parts. With these four rules, one either marks many (at least $3k$) vertices, in which case one can conclude that the gain is indeed at least k and return $O(k)$ vertices as a certificate for that; or apply two more rules to the graph G and the set M of marked vertices. Those additional rules simplify the graph further but do not mark any vertices or change the parameter.

This entire *set* of six rules can exhaustively be applied in time $O(m)$, and importantly, to any connected graph on at least one edge, at least one rule applies. The key property is that any connected graph exhaustively reduced by these rules

- has only $O(k)$ vertices,
- has gain exactly k' if and only if the original graph has a gain of exactly k'.

For details on the linear-time kernelization for Max-Cut, we refer to Etscheid and Mnich [20]. For further reading on kernelization as a whole, we recommend the upcoming textbook [24].

We remark here that kernelization so far solely focus on data reduction to preserving the exact optimal solutions. There are related notions in the literature that deal with preserving approximate solutions, such as core sets; for an example, we refer to the work by Feldman et al. [23].

The positive impact that linear-time preprocessing can have in practice for large data sets has been reported repeatedly. For instance, Strash [51] has shown that just two simple reductions (vertex folding and isolated vertex removal) are sufficient to make many real-world instances tractable for finding maximum independent sets. More advanced reduction rules can be used to solve real-world networks on millions of vertices [2]. Similarly, a linear-time kernelization algorithm by Iwata [35] for Feedback Vertex Set emerged as the winner of the First Parameterized Algorithms and Computational Experiments Challenge.

The algorithmic methods we discussed so far neglect the plausible circumstance that the data set which we want to optimize over is too large to fit into the computing unit with which we plan to execute the data reduction, or solve the optimization part. A possible way to overcome this impediment is to design data reduction algorithms which only look at a small part of the data (which fits into the main memory) at any given moment of time, reduce or process it, and then access the next yet-unprocessed part which is meanwhile stored on some large (but slower) external storage equipment. This type of algorithms has long been investigated under the name of external memory algorithms; the main successes of this fundamental algorithm design paradigm are surveyed by Vitter [55]. In the context of discrete optimization problems, Maheshwari and Zeh [41] designed external-memory algorithms to compute minimum-width tree decompositions of graphs with sorting complexity, which is a benchmark for optimal external-memory algorithms. Given such a tree decomposition, they are able to design dynamic programming routines for many fundamental discrete optimization problems, including the single-source shortest path problem and several problems on graphs of bounded treewidth. Their methods work in serial external memory models that take the caches of the memory hierarchy into account. An extension of these models which is particularly relevant for big data sets are parallel external memory models, in which the large data set is distributed over a number of storage and computing units which can act on it in parallel. Such parallel processing of data has become standard in scientific and industrial applications. Jacob et al. [37]lifted the aforementioned results to the parallel external memory model. They were then able to show that a large class of discrete optimization problems admit efficient data reduction with only sorting complexity on planar networks, to kernels whose size depends only linear on the sought-after optimal solution.

4 Facing velocity: algorithms for dynamic big data

A common aspect of data sets is that they change over time. For big data sets, this usually means that only a tiny fraction of the data set changes (in one time step), whereas most of it remains unchanged. So once we have spend a considerable amount of computational effort for solving an optimization problem on the entire data set, we do not want to recompute the entire solution from scratch, but re-use the optimal solution for the old instance to compute an optimal solution for the new instance in sublinear time. Such type of problems are addressed by the research area of dynamic algorithms.

The goal of dynamic algorithms is to design data structures that store a dynamically changing instance of a problem, which can answer queries about the current instance and can perform small changes on the instance. Then it is all about making the updates and queries as fast as possible.

Many a data structure have been proposed for dynamic graph problems, where updates are usually edge or vertex insertions and deletions. The main quest has been to devise such data structures for graph problems that can be solved in near-linear time in the static case, for the following reason. Any dynamic graph algorithm that can perform edge insertions can simulate static algorithm by starting with an empty graph and using m insertions to insert the m-edge input graph. That is, if the update time of the dynamic algorithm is $u(m)$ then the static problem can be solved in $O(m \cdot u(m))$ time, plus the time to query for the output. Hence, if a problem requires $\Omega(f(m))$ time to be solved statically, then any dynamic algorithm that can insert edges, and can be queried for the problem solution in $o(f(m))$ time, must need $\Omega(f(m)/m)$ (amortized) time to perform updates. So for near-linear time static algorithms one aims for dynamic algorithms with near-constant time updates—the holy grail of dynamic algorithms. Similar observations hold true for vertex insertions and most other types of updates. One, therefore, asks which static problems solvable in time $f(m)$ can be fully "dynamized", in the sense of having dynamic algorithms that support updates in $O(f(m) / m)$ time. This question has been answered affirmatively for many fundamental graph problems including connectivity (e.g., [30, 33, 34, 52]), reachability [32], shortest paths (e.g., [8, 18, 31]), and maximum matching [9, 27, 49].

Until now, all but a negligible fraction of dynamic algorithms have been suggested for polynomial-time solvable problems. Yet, a bulk of interesting optimization questions on large data sets are NP-hard. By the discussion above, dynamic algorithms for NP-hard problems necessarily have superpolynomial query/update times when it comes to exact solutions, assuming P $\neq$ NP. When only approximate solutions need to be maintained efficient dynamic algorithms have been obtained for some polynomial-time approximable problems such as dynamic approximate vertex cover [7, 9, 46].

Regarding exact solutions, the discussion abovestrongly suggests considering NP-hard problems admitting *linear-time fixed-parameter algorithms* in the static setting. Recall from Sect. 2 that instances of such problems are measured by their size n as well as a parameter k, and *linear-time fixed-parameter algorithms* solve them in time $f(k) \cdot n$ for some function f. When f exhibits only moderate growth, then such algorithms can be very practical for small values of k. Consequently, researchers aim for algorithms where the function f is as slow growing as possible, assuming a standard hypothesis such as the Exponential-Time Hypothesis. They have succeeded for many different parameterized problems; results in this direction include (1) all algorithms that follow from Courcelle's theorem[1] [14], (2) many branching tree algorithms such as those for Vertex Cover and d -Hitting Set, (3) many algorithms based on color-coding [4] such as for k -Path, and (4) many more [10, 19, 20, 36, 40, 53, 56].

Alman et al. [3] study which NP-hard optimization problems with (near-)linear time fixed-parameter algorithms can be made efficiently dynamic. Specifically, they investigate which problems solvable in $f(k)n^{1+o(1)}$ time have dynamic algorithms with update and query times at most $n^{o(1)}$ and which problems solvable in $f(k)n$ time have dynamic algorithms with update and query times that depend solely on k and not on n.

To answer those questions, they introduce two techniques for making fixed-parameter algorithms dynamic, and then use them to develop dynamic fixed-parameter algorithms for a multitude of fundamental optimization problems. An example problem they address is the task of covering points with lines. This task originates from applications where turns are considered very costly [50]. One particular big data application were this is the case is laying circuits as part of VLSI design, where one would like to find a traveling salesperson tour with as few bends as possible [5]. Formally, in the Point Line Cover problem for a set $\mathcal{P}$ of n points in the plane one must decide whether k lines suffice to pass through all its points. Such a set of lines covering all points in $\mathcal{P}$ is called a *line cover* of $\mathcal{P}$. They also consider the *dual* problem known as Line Point Cover, where for a set $\mathcal{L}$ of n lines in the plane and an integer $k \in \mathbb{N}$, one must find a set of at most k points passing through all the lines in $\mathcal{L}$.

It is folklore that these problems are equivalent, by replacing the point (a, b) with the line $y = ax - b$ or vice versa. These problems fall into a more general class of geometric problems, best described as *covering things with things* [39]. Alman et al. [3] give a dynamic algorithm for Point Line Cover which handles edge insertions in $O(g(k)^2)$ time, edge deletions in $O(g(k)^3)$ time, and queries in $O(g(k)^{2g(k)+2})$ time, under the promise that there is a computable function g such that the point set can always be covered by at most $g(k)$ lines. Essentially, they proceed as follows. First, if $g(k) < k$ then one has a "no"-instance, so they can assume that $g(k) \geq k$. Note that if there is a line cover with at most k lines, then any line which passes through at least $k + 1$ points must be contained in the line cover. In light of their promise, they only use the weaker fact that any line which passes through at least $g(k) + 1$ points must be contained in any line cover with at most k lines. They thus maintain a set $\mathcal{L}_H$ of lines which pass through at least $g(k) + 1$ points, and for each $\ell \in \mathcal{L}_H$, a set $\mathcal{P}_\ell$ of at least $g(k) + 1$ points on that line. They further maintain the set $\mathcal{P}'$ of points that are not in $\mathcal{P}_\ell$ for any $\ell \in \mathcal{L}_H$, and that each point is in exactly one such set.

Since every line in $\mathcal{L}_H$ must be in a line cover of size at most $g(k)$, their promise implies that we will always have $|\mathcal{L}_H| \leq g(k)$. Furthermore, since no line covers more than $g(k)$ points of $\mathcal{P}'$, we must always have $|\mathcal{P}'| \leq g(k)^2$. It is then straightforward to maintain $\mathcal{L}_H$, $\mathcal{P}_\ell$, and $\mathcal{P}'$ in $O(g(k)^2)$ time per insertion and $O(g(k)^3)$ time per deletion. When a new point p is inserted, they first check for each line ℓ in $\mathcal{L}_H$ whether p is on ℓ, and if so we add it to $\mathcal{P}_\ell$ and conclude. If it is not on any of these lines, they add it to $\mathcal{P}'$. Then, for each line formed by p and another point in $\mathcal{P}'$, they check whether that line contains at least $g(k) + 1$ points in $\mathcal{P}'$. If they find such a line ℓ', they add ℓ' to $\mathcal{L}_H$, and remove those $g(k) + 1$ points from $\mathcal{P}'$ and add them instead to $\mathcal{P}_{\ell'}$. When a point p is removed, they remove it from $\mathcal{P}'$ if it is in that set. If, instead, it is in $\mathcal{P}_\ell$ for some line $\ell \in \mathcal{L}_H$, then they remove it from $\mathcal{P}_\ell$. If $\mathcal{P}_\ell$ now consists of at most $g(k)$ points, then they remove ℓ from $\mathcal{L}_H$, and reinsert all the points from $\mathcal{P}_\ell$ as above.

Finally, to go from the sets maintained to the point line cover of size at most k (or the conclusion that none currently exists) in order to answer queries, they do the following. First, every line in $\mathcal{L}_H$ must be included since these lines each contain at least $g(k) + 1 \geq k + 1$ points. If there are more than k such lines, then there is no line cover of size at most k. Otherwise, if there are $a \leq k$ such lines, they need to determine if there is a line cover of $\mathcal{P}'$ with only $k - a$ lines. Since $|\mathcal{P}'| \leq g(k)^2$, this can be solved in $O(g(k)^{2g(k)+2})$ time, using a simple static branching algorithm (see, e.g., [39, Thm. 1]). If they find such a set S of lines, they return $S \cup \mathcal{L}_H$, and otherwise return that there is no line cover of $\mathcal{P}$ with size at most k.

An interesting question is whether the promise assumption for the Point Line Cover problem can be removed, and it is then still possible to have a dynamic algorithm.

Apart from positive results in form of dynamic fixed-parameter algorithms, Alman et al. [3] also attempt to show that (under plausible conjectures) certain parameterized

[1] Courcelle's theorem states that every problem definable in monadic second-order logic of graphs can be decided in linear time on graphs of bounded treewidth.

problems—that statically can be solved in $f(k)n^{1+o(1)}$ time—require $\Omega(f(k)n^{\delta})$ (for someconstant $\delta > 0$) update time to maintain dynamically. They then give such hardness results for several optimization problems on directed graphs, such as Longest Path or Directed Feedback Vertex Set, even for constant parameter values $k \leq 3$, under a certain hypothesis.

For now, the work of Alman et al. [3] is purely theoretical; it would be of great interest to understand the performance of their algorithms when dealing with big data sets.

5 Facing veracity: algorithms for uncertain big data

We now address the problem of solving combinatorial optimization problems over large-scale data sets, where part of the data is unknown. Our working hypothesis is that only a small part (compared to the overall size) of the data set is potentially corrupted or uncertain, for otherwise we deem it questionable whether any reasonable answer at all can potentially be derived from the data. To avoid trivialities, we judge this uncertain data part to be crucial for answering the respective optimization question. In contrast to the dynamic setting that we discussed in Sect. 4, where after each update we again have the complete data set at our disposal to solve the same optimization question, we now want to answer the question only once but then for every possible realisation of the uncertain data.

In particular, storing the uncertain data with some k bits, this leads to 2^k potential realisations, and precommitting to all of them leads to an exponential blow-up which is unacceptable for space and time reasons. For remedy, Fafiane et al. [22] aim to solve or preprocess as much of the data as possible without knowing the missing or uncertain parts, instead of committing to a solution for the optimization question on the full data set. This allows them to still perform computations once the entire input is known/certain.

Prototypical big data applications coming to mind here are route planning tasks for large road networks which exhibit a mostly regular pattern garnished with small but crucial irregularities due to constructionworks or other causes of congestion. However, as every commuter reliant on car travel knows, traffic times can experience high degrees of uncertainty due to congestion; still we want our navigation system to quickly come up with a fast route. Formally, we are facing a road network N with given travel times between pairs of cities, such that for a certain set R of roads their travel times are unknown (due to congestion) at the time of planning a shortest route from a city A to a city B. If we do not have any information about the travel times on R, then we cannot expect to find a route whose cost is within any bounded factor of the fastest route, as any segment of our route may take time that is infinitely longer than traversing it on a fastest route. Fafiane et al. [21] address this challenge by preprocessing the large road network N to a small network N' whose size depends only on R, and thus on the level of uncertainty. They observe that the final fastest route path will consist in some arbitrary way of roads in R, and fastest ($C \rightarrow C'$)-routes avoiding all roads in R for cities C, C' being either A, B or lying at the end of some road in R. Such fastest (C, C') routes can be precomputed by taking fastest routes in $N - R$. All travel time information can be stored in a small network with cities A, B and those at the ends of roads in R, by letting the travel time from C to C' be the time on the fastest way in N–F. The roads in R are then additional parallel roads and the actual fastest route from A to B can be computed once their travel times are certain. This way, Fafiane et al. [21] show that instead of finding the fastest route in the large network N, once all travel times in R are known it suffices to solve the problem on the small network with only $2 + 2|R|$ cities.

6 Conclusions

Optimization over Big Data Sets is a vast and highly active research area, connecting many different fields. With this survey we wanted to give a small overview of algorithmic techniques which are available for discrete optimization tasks. It would be imprudent to claim that we have covered all such methods, even in this narrow subarea of research. We thus hasten to add references to surveys about sublinear-time algorithms [16], streaming algorithms [42], and convex optimization [12]. Even for the methods mentioned here, space limitations did not allow us to go into too much detail, so we focussed on some easily accessible examples.

For further reading, we refer to the textbook of Cygan et al. [15] which provides a thorough treatment of multivariate algorithms; the upcoming textbook by Fomin et al. [24] will study data reduction by preprocessing in great detail; dynamic algorithms for discrete optimization problems are surveyed by Boria andPaschos [11]; external memory algorithms are covered by Vitter [55].

Acknowledgements The author thanks Uli Meyer for creating the opportunity of writing this survey. He also thanks Alexander Grigoriev, and two anonymous reviewers, for helpful feedback that led to an improved presentation.

References

1. Ajwani D, Meyer U, Osipov V (2007) Improved external memory BFS implementations. Proc ALENEX 2007:3–12
2. Akiba T, Iwata Y (2016) Branch-and-reduce exponential/FPT algorithms in practice: a case study of vertex cover. Theoret Comput Sci 609(Part 1):211–225
3. Alman J, Mnich M, Vassilevska Williams V (2017) Dynamic parameterized problems. In: Proceedings of ICALP 2017, Leibniz Int Proc Informatics, vol 80, pp 41:1–41:16
4. Alon N, Yuster R, Zwick U (1995) Color-coding. J ACM 42(4):844–856
5. Arkin EM, Bender MA, Demaine ED, Fekete SP, Mitchell JSB, Sethia S (2005) Optimal covering tours with turn costs. SIAM J Comput 35(3):531–566
6. Arora S, Ge R, Kannan R, Moitra A (2016) Computing a nonnegative matrix factorization—provably. SIAM J Comput 45(4):1582–1611
7. Baswana S, Gupta M, Sen S (2015) Fully dynamic maximal matching in $O(\log n)$ update time. SIAM J Comput 44(1):88–113
8. Bernstein A, Roditty L (2011) Improved dynamic algorithms for maintaining approximate shortest paths under deletions. Proc SODA 2011:1355–1365
9. Bhattacharya S, Henzinger M, Italiano GF (2015) Deterministic fully dynamic data structures for vertex cover and matching. Proc SODA 2015:785–804
10. Bodlaender HL, Drange PG, Dregi MS, Fomin FV, Lokshtanov D, Pilipczuk M (2016) A $c^k n$ 5-approximation algorithm for treewidth. SIAM J Comput 45(2):317–378
11. Boria N, Paschos V (2011) A survey on combinatorial optimization in dynamic environments. RAIRO Oper Res 45:241–294
12. Cevher V, Becker S, Schmidt M (2014) Convex optimization for big data: scalable, randomized, and parallel algorithms for big data analytics. IEEE Signal Proc Mag 31(5):32–43
13. Charikar M, Guruswami V, Kumar R, Rajagopalan S, Sahai A (2000) Combinatorial feature selection problems (extended abstract). Proc FOCS 2000:631–640
14. Courcelle B (1990) The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf Comput 85(1):12–75
15. Cygan M, Fomin FV, Kowalik Ł, Lokshtanov D, Marx D, Pilipczuk M, Pilipczuk M, Saurabh S (2015) Parameterized algorithms. Springer, Cham
16. Czumaj A, Sohler C (2006) Sublinear-time algorithms. Bull Eur Assoc Theor Comput Sci EATCS 89:23–47
17. Czumaj A, Sohler C (2007) Sublinear-time approximation algorithms for clustering via random sampling. Random Struct Algorithms 30(1–2):226–256
18. Demetrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths. J ACM 51(6):968–992
19. Dorn F (2010) Planar subgraph isomorphism revisited. In: Proceedings of STACS 2010, Leibniz Int Proc Informatics, vol 5, pp 263–274
20. Etscheid M, Mnich M (2016) Linear kernels and linear time algorithms for finding large cuts. In: Proceedings of ISAAC 2016, Leibniz Int Proc Informatics, vol 64, pp 31:1–31:13
21. Fafianie S, Hols EMC, Kratsch S, Quyen VA (2016)Preprocessing under uncertainty: Matroid intersection. In: Proceedings of MFCS 2016, Leibniz Int Proc Informatics, vol 58, pp 35:1–35:14
22. Fafianie S, Kratsch S, Quyen VA (2016) Preprocessing under uncertainty. In: Proceedings of STACS 2016, Leibniz Int Proc Informatics, vol 47, pp 33:1–33:13
23. Feldman D, Schmidt M, Sohler C (2013) Turning big data into tiny data: constant-size coresets for k-means, PCA and projective clustering. In: Proceedings of SODA 2013
24. Fomin FV, Lokshtanov D, Saurabh S, Zehavi M (2017) Kernelization: theory of parameterized preprocessing. Cambridge University Press, Cambridge
25. Froese V, van Bevern R, Niedermeier R, Sorge M (2016) Exploiting hidden structure in selecting dimensions that distinguish vectors. J Comput System Sci 82(3):521–535
26. Grippo L, Palagi L, Piacentini M, Piccialli V, Rinaldi G (2012) SpeeDP: an algorithm to compute SDP bounds for very large Max-Cut instances. Math Program 136(2, Ser. B):353–373
27. Gupta M, Peng R (2013) Fully dynamic $(1+\varepsilon)$-approximate matchings. Proc FOCS 2013:548–557
28. Gyárfás A (1990) A simple lower bound on edge coverings by cliques. Discret Math 85(1):103–104
29. Hagerup T (2012) Simpler linear-time kernelization for planar dominating set. In: Proceedings of IPEC 2011, Lecture Notes Comput Sci, vol 7112, pp 181–193
30. Henzinger MR, King V (2001) Maintaining minimum spanning forests in dynamic graphs. SIAM J Comput 31(2):364–374
31. Henzinger M, Krinninger S, Nanongkai D (2016) Dynamic approximate all-pairs shortest paths: breaking the $O(mn)$ barrier and derandomization. SIAM J Comput 45(3):947–1006
32. Henzinger M, Krinninger S, Nanongkai D (2015) Improved algorithms for decremental single-source reachability on directed graphs. In: Proceedings of ICALP 2015, Lecture Notes Comput Sci, vol 9134, pp 725–736
33. Holm J, de Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48(4):723–760
34. Huang S, Huang D, Kopelowitz T, Pettie S (2016) Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. Tech Rep. http://arxiv.org/abs/1609.05867
35. Iwata Y (2017) Linear-time kernelization for feedback vertex set. In: Proceedings of ICALP 2017, Leibniz Int Proc Informatics, vol 80, pp 68:1–68:14
36. Iwata Y, Oka K, Yoshida Y (2014) Linear-time FPT algorithms via network flow. Proc SODA 2014:1749–1761
37. Jacob R, Lieber T, Mnich M (2014) Treewidth computation and kernelization in the parallel external memory model. In: Proceedings of TCS 2014, Lecture Notes Comput Sci, vol 8705, pp 78–89
38. Kumar A, Sabharwal Y, Sen S (2010) Linear-time approximation schemes for clustering problems in any dimensions. J ACM 57(2):5:1–5:32
39. Langerman S, Morin P (2005) Covering things with things. Discret Comput Geom 33(4):717–729
40. Lokshtanov D, Ramanujan MS, Saurabh S (2015) Linear time parameterized algorithms for subset feedback vertex set. In: Proceedings of ICALP 2015, Lecture Notes Comput Sci, vol 9134, pp 935–946
41. Maheshwari A, Zeh N (2009) I/O-efficient algorithms for graphs of bounded treewidth. Algorithmica 54(3):413–469
42. McGregor A (2014) Graph stream algorithms: a survey. SIGMOD Rec 43(1):9–20
43. Moitra A (2016) An almost optimal algorithm for computing nonnegative rank. SIAM J Comput 45(1):156–173
44. Nemhauser GL, Trotter LE Jr (1975) Vertex packings: structural properties and algorithms. Math Program 8:232–248
45. Nielsen FA (2008) Clustering of scientific citations in wikipedia. Tech. Rep. https://arxiv.org/abs/0805.1154
46. Onak K, Rubinfeld R (2010) Maintaining a large matching and a small vertex cover. Proc STOC 2010:457–464
47. Protti F, Dantas da Silva M, Szwarcfiter JL (2009) Applying modular decomposition to parameterized cluster editing problems. Theory Comput Syst 44(1):91–104

48. Qiu J, Wu Q, Ding G, Xu Y, Feng S (2016) A survey of machine learning for big data processing. EURASIP J Adv Signal Proc 2016(1):67
49. Solomon S (2016) Fully dynamic maximal matching in constant update time. Proc FOCS 2016:325–334
50. Stein C, Wagner DP (2001) Approximation algorithms for the minimum bends traveling salesman problem. In: Proceedings of IPCO 2001, Lecture Notes Comput Sci, vol 2081, pp 406–421
51. Strash D (2016) On the power of simple reductions for the maximum independent set problem. In: Proceedings of COCOON 2016, Lecture Notes Comput Sci, vol 9797, pp 345–356
52. Thorup M (2000) Near-optimal fully-dynamic graph connectivity. Proc STOC 2000:343–350
53. van Bevern R (2014) Towards optimal and expressive kernelization for d-hitting set. Algorithmica 70(1):129–147
54. van Bevern R, Hartung S, Kammer F, Niedermeier R, Weller M (2012) Linear-time computation of a linear problem kernel for dominating set on planar graphs. In: Proceedings of IPEC 2011, Lecture Notes Comput Sci, vol 7112, pp 194–206
55. Vitter JS (2008) Algorithms and data structures for external memory. Found Trends Theor Comput Sci 2(4):305–474
56. Wahlström M (2014) Half-integrality, LP-branching and FPT algorithms. Proc SODA 2014:1762–1781
57. Ward JS, Barker A (2013) Undefined by data: a survey of big data definitions. Tech. Rep. https://arxiv.org/abs/1309.5821
58. Zhou L, Pan S, Wang J, Vasilakos AV (2017) Machine learning on big data: opportunities and challenges. Neurocomputer 237:350–361