# Space Limited Graph Algorithms on Big Data

Jianer Chen[1,2(✉)], Zirui Chu[1], Ying Guo[1], and Wei Yang[1]

[1] Guangzhou University, Guangzhou 510006, People's Republic of China
chen@cse.tamu.edu
[2] Texas A&M University, College Station, TX 77843, USA

**Abstract.** We study algorithms for graph problems in which the graphs are of extremely large size $N$ so that super-linear time $\omega(N)$ or linear space $\Theta(N)$ would become impractical. We use a parameter $k$ to characterize the computational power of a normal computer that can provide additional time and space bounded by polynomials of $k$. In particular, we are interested in strict linear-time algorithms using space $O(k^{O(1)})$. In our case studies, as examples, we present a randomized algorithm of time $O(N)$ and space $O(k^2)$ that constructs a maximal matching of size upper bounded by $k$ in a graph of size $N$, and a randomized kernelization algorithm of time $O(N)$ and space $O(k^3)$ for the NP-hard Edge Dominating Set problem. Our kernelization algorithm for Edge Dominating Set has its kernel size match the best kernel size by known polynomial-time kernelization algorithms for the problem with no space complexity constraints. We also show that the techniques developed in our algorithms can be used to develop improved streaming algorithms.

**Keywords:** Big data · Maximal matching · Edge dominating set

## 1 Introduction

Recent progress in data science has shown that classical algorithmic techniques may become inadequate when dealing with data sets of enormous size. For example, it was estimated that Baidu stored 2,000 petabyte data, Google stored 15,000 petabyte data while NSA held 10,000 petabyte data [1]. Thus, a traditionally "efficient" algorithm of, say, quadratic running time, may turn out to be not practically feasible. There have been fast growing interests in the study of massive data sets. The research has included the study of structures of massive data and data queries (e.g., [10]), parallel and distributed processing of massive data (e.g., [15]), and preprocessing of massive data (e.g., [9]). The research has been driven directly by practical applications in massive data processing, and is essentially heuristic-based. There has also been very active research in the algorithmic community. The study of very fast (sublinear-time, linear-time, or nearly linear-time) algorithms in dealing with massive data sets has drawn extensive

Supported by National Natural Science Foundation of China under grant 61872097.


Y. Zhang et al. (Eds.): COCOON 2022, LNCS 13595, pp. 255–267, 2022.
https://doi.org/10.1007/978-3-031-22105-7_23

attention. A number of computation models for dealing with massive data sets have been proposed and studied. In particular, *data streaming models* [16] have been studied, where algorithms must process the input stream in the order it arrives while using a limited amount of memory. Recently, studies on streaming algorithms based on parameterized computation have appeared [5,6,8].

In the current paper, we study a new computational model for massive data processing with limited "local" computing resources. Our model is of a multivariate nature, which measures the complexity of reading the very large input data sets in terms of the size of the data sets and analyzes the computational cost in terms of a parameter that characterizes the computational power provided by limited local computing resources. Problems in our consideration have two parameters $N$ and $k$, where $N$ is the input size, which is assumed to be extremely large so that super-linear time (such as quadratic-time) algorithms would be considered impractical, while $k$ gives the "size" of feasibility such that the limited local computing resource (e.g., a normal computer) can handle problems with complexity (time and space) bounded by polynomials of $k$, in addition to the linear-time reading from the input. More specifically, we will study algorithms for graph problems in which the graphs are of extremely large size $N$, such that the algorithms run in *strict linear-time* using memory space polynomial in the parameter $k$, i.e., that the algorithms run in time $O(N)$ and space $O(k^{O(1)})$.

We argue that the proposed model is theoretically interesting and practically meaningful. Insisting on strict linear-time allows processing data sets of very large size. On the other hand, there seems no simple functional relations between the input size and the available computational power. In many cases, problems in massive data processing (such as aggregations) look for solutions of size manageable by local computational resources, where the solution size and the size of input data are not directly correlated. Therefore, it is meaningful to introduce another parameter $k$ to characterize the available computational resources. The constraint on the space complexity in terms of the parameter $k$ reflects the fact that although massive data are stored publicly so that users can read the data, the users do not own the space so cannot write over the data in the space. Limiting local resources to be bounded by polynomials of the parameter $k$ offers new challenges in algorithmic research. Also, optimizing the cost of local resources in terms of the parameter $k$ widens the applicability of the algorithms. For example, if $k$ is the solution size, then algorithms requiring less local resources can handle massive data problems with larger solutions.

As examples, we study well-known graph problems that have been extensively studied in algorithmic research, and demonstrate how they can be solved in the proposed model. In particular, we present a randomized algorithm of time $O(N)$ and space $O(k^2)$ that constructs a maximal matching of size upper bounded by $k$ in a graph of size $N$, and a randomized kernelization algorithm of time $O(N)$ and space $O(k^3)$ for the NP-hard Edge Dominating Set problem. Our kernelization algorithm for Edge Dominating Set has its kernel size match the best kernel size by known polynomial-time kernelization algorithms for the problem with no space complexity constraints. We also show that the techniques developed in our algorithms can be used to develop improved streaming algorithms.

Because of the space limit, proofs for the lemmas and theorems that are marked with ♢ are omitted and will be given in the full version of the paper.

## 2 Constructing a Maximal Matching

We start with the well-known Maximal Matching problem. For an integer $q > 0$, we will denote by $[q]$ the set $\{0, 1, 2, \ldots, q-1\}$. Let $G = (V, E)$ be the input graph. We define the *size* $N$ of the graph $G$ to be $N = |V| + |E|$. Without loss of generality, we will assume that $V = [|V|]$.

Recall that a *matching* in the graph $G$ is a set of edges in which no two edges share a common endpoint. The *size* of the matching $M$ is the number of edges in $M$. The matching $M$ in $G$ is *maximal* if for any edge $e \notin M$ in $G$, $M \cup \{e\}$ is not a matching in $G$. For a subgraph or an edge subset $Z$ of $G$, we will denote by $V(Z)$ the set of vertices in $Z$. In particular, for a matching $M$, $V(M)$ is the set of vertices that are endpoints of the edges in the matching $M$.

Maximal matching has many applications in the study of other graph problems. A large matching in a graph $G$ may directly lead to the nonexistence of a solution to $G$ for certain graph problems. On the other hand, a (small) maximal matching in $G$ may provide rich information for the solutions to the problems. For example, for the famous NP-hard Vertex Cover problem, a matching of size larger than $k$ in a graph directly implies that the graph has no vertex cover of size $k$, while the vertex set of a maximal matching in the graph is a good approximation to the minimum vertex cover of the graph [4]. Another example of applications of maximal matching will be given in the next section when we study kernelization algorithms for the NP-hard Edge Dominating Set problem. Motivated by these observations, we will study the following problem:

(parameterized) Maximal Matching (p-MaxMatch)
Given a graph $G$ and a parameter $k$, either construct a matching of size larger than $k$, or construct a maximal matching of size bounded by $k$.

Note that a solution to p-MaxMatch can be easily derived from a maximal matching of the graph $G$. Moreover, if we had space linear in $N$, then a maximal matching $M$ in $G$ can be constructed by the following greedy algorithm: first unmark all vertices, then scan the edges of the graph $G$, and add an edge $e$ to $M$ if the two endpoints of $e$ are unmarked then mark the two endpoints of $e$. However, this algorithm will need $\Omega(|V(G)|)$ space to record if each vertex in $G$ is marked or not, which becomes infeasible in the environment of big data. Alternatively, we can use a balanced search tree (e.g., a red-black tree [7]) that supports search and insertion in logarithmic time per operation to store the marked vertices. This takes space $O(k)$ for matchings of size $O(k)$. However, using this data structure, checking if a vertex is marked would take time $O(\log k)$, resulting in an $O(N \log k)$-time algorithm, which is a super-linear time algorithm.

We present an algorithm that runs in time $O(N)$ and space $O(k^2)$ for the p-MaxMatch problem. We assume that the edges of the graph $G$ are given in a fixed order. Thus, the maximal matching constructed by the greedy algorithm described in the previous paragraph is well-defined and unique. We denote this

maximal matching by $M_{\max}$. Note that we only use the greedy algorithm to *define* the maximal matching $M_{\max}$. Thus, the complexity of the greedy algorithm constructing the matching $M_{\max}$ is irrelevant to that of our algorithms.

Let the maximal matching $M_{\max}$ be $M_{\max} = \{e_1, e_2, \ldots, e_r\}$, where the edges $e_1$, $e_2$, ..., $e_r$ are given in the order constructed by the greedy algorithm. Based on the matching $M_{\max}$, we define another matching $M^*$ as follows.

$$M^* = \begin{cases} M_{\max} & \text{if } r \leq k, \\ \{e_1, e_2, \ldots, e_k, e_{k+1}\} & \text{if } r > k. \end{cases} \tag{1}$$

Thus, $M^*$ is a matching of at most $k+1$ edges. If $|M^*| \leq k$, then $M^*$ is the maximal matching $M_{\max}$. The vertex set $V(M^*)$ of the matching $M^*$ contains at most $2k+2$ vertices. In the discussions of this section, we will let $k_2 = 2k+2$.

We say that a hash function $h$ is *injective* from a vertex subset $V_0$ if for any $v, w \in V_0$, $v \neq w$, we have $h(v) \neq h(w)$. We say that an array $A[0..k_2^2 - 1]$ *represents the vertex set* $V_0$ *via a hash function* $h$ if (1) $h$ is a hash function from $V(G)$ to $[k_2^2]$ that is injective from $V_0$; and (2) for each $v \in V_0$, $A[h(v)] = v$, and for all other $j$'s (i.e., $j \neq h(v)$ for any $v \in V_0$), $A[j] = -1$.

Let $\mathcal{H}_0 = \{h_1, \ldots, h_b\}$ be a set of hash functions from $V(G)$ to $[k_2^2]$. Consider the algorithm **Match-0** in Fig. 1, where $M[1..b]$ is an array in which the element $M[i]$ is a matching in $G$ such that the hash function $h_i$ is injective from $V(M[i])$, and $H[1..b, 0..k_2^2 - 1]$ is a 2-dimensional array such that for each $i$, the 1-dimensional array $H[i, 0..k_2^2 - 1]$ represents the vertex set $V(M[i])$ via $h_i$.

**Algorithm Match-0**$(G, k, \mathcal{H}_0)$
INPUT: graph $G$, parameter $k$, a set $\mathcal{H}_0 = \{h_1, \ldots, h_b\}$ of hash functions from $V(G)$ to $[k_2^2]$
OUTPUT: $(M[i], h_i, H[i, 0..k_2^2 - 1])$, where $M[i]$ is a matching in $G$, $h_i$ is a hash function in $\mathcal{H}_0$ injective from $V(M[i])$, $H[i, 0..k_2^2 - 1]$ is the array that represents $V(M[i])$ via $h_i$.

1. **for** $(1 \leq i \leq b)$ $\{M[i] = \emptyset$; **for** $(0 \leq j \leq k_2^2 - 1)$ $H[i, j] = -1;\}$
2. $\mathcal{H} = \mathcal{H}_0$;
3. **for** (each edge $[v, w]$ in the graph $G$)
3.1 **for** (each hash function $h_i$ in $\mathcal{H}$)
3.1.1 **if** $(h_i(v) \neq h_i(w)$ and $H[i, h_i(v)] = -1$ and $H[i, h_i(w)] = -1)$
3.1.1.1 $\{H[i, h_i(v)] = v$; $H[i, h_i(w)] = w$; $M[i] = M[i] \cup \{[v, w]\};\}$
3.1.1.2 **if** $(|M[i]| > k)$ return$(M[i], h_i, H[i, 0..k_2^2 - 1])$;
3.1.2 **else if** $(H[i, h_i(v)] \neq v$ and $H[i, h_i(w)] \neq w)$
delete $h_i$ from the set $\mathcal{H}$;
3.2 **if** $(\mathcal{H} = \emptyset)$ return("no injective function from $V(M^*)$ in $\mathcal{H}_0$");
4. pick any $h_i$ in $\mathcal{H}$, and return$(M[i], h_i, H[i, 0..k_2^2 - 1])$.

**Fig. 1.** Constructing a maximal matching

**Theorem 1 (♢).** *The set* $\mathcal{H}_0$ *contains a hash function injective from* $V(M^*)$ *if and only if the algorithm* **Match-0**$(G, k, \mathcal{H}_0)$ *returns a triple* $(M[i], h_i, H[i, 0..k_2^2 - 1])$. *If this is the case, then* (1) $M[i]$ *is the matching* $M^*$*;* (2) $h_i \in \mathcal{H}_0$ *is injective from* $V(M[i])$*; and* (3) *the array* $H[i, 0..k_2^2 - 1]$ *represents* $V(M[i])$ *via* $h_i$.

We analyze the complexity of the algorithm **Match-0**, assuming that each hash function can be represented in space $O(1)$, and that for a vertex $v \in V(G)$, the value $h_i(v)$ for a hash function $h_i$ in $\mathcal{H}_0$ can be computed in constant time.

**Theorem 2 (♢).** *Algorithm* **Match-0**$(G, k, \mathcal{H}_0)$ *takes time* $O(N \cdot b)$ *and space* $O(k^2 b)$ *on a graph of size* $N$*, where* $b$ *is the number of hash functions in* $\mathcal{H}_0$*.*

**Remark 1.** For finding the matching $M^*$ and the hash function $h_i$ in $\mathcal{H}_0$ that is injective from $V(M^*)$, we can run the algorithm **Match-0** in an alternative way, which can reduce the space complexity to $O(k^2)$ while keeping the same time complexity. For this, we can call the algorithm **Match-0**$(G, k, \{h_i\})$ on each hash function $h_i$ in the set $\mathcal{H}_0$, and stop when **Match-0**$(G, k, \{h_i\})$ returns a triple $(M_i, h_i, H_i[0..k_2^2 - 1])$ on a hash function $h_i$. By Theorem 1, in this case, $M_i = M^*$, $h_i \in \mathcal{H}_0$ is injective from $V(M^*)$, and the array $H_i[0..k_2^2 - 1]$ represents the vertex set $V(M^*)$ via $h_i$. By Theorem 2, **Match-0**$(G, k, \{h_i\})$ takes time $O(N)$ and space $O(k^2)$. Thus, running **Match-0**$(G, k, \{h_i\})$ over all hash functions $h_i$ in $\mathcal{H}_0$ takes time $O(N \cdot b)$, which is the same as that given in Theorem 2. On the other hand, since space complexity is "re-useable", this approach takes space $O(k^2)$ instead of $O(k^2 b)$. The drawback of this approach is that we need to scan the input graph $G$ up to $b$ times. This, though keeping the same asymptotic time complexity as that of the algorithm **Match-0**, causes repeatedly scanning the input graph, which, in general, is undesirable in big data computations.

By Theorem 1, to solve the p-MaxMatch problem, i.e., to construct the matching $M^*$ for the instance $(G, k)$ of the problem, we need to have a set $\mathcal{H}_0$ of hash functions from $V(G)$ to $[k_2^2]$ in which at least one is injective from the vertex set $V(M^*)$. This can be achieved using universal hash functions [7].

**Proposition 1 ([7]).** *For any subset* $S$ *of* $V(G)$ *with* $|S| \leq s$*, if we randomly pick a hash function* $h$ *from a universal set* $\mathcal{H}_u$ *of hash functions from* $V(G)$ *to* $[s^2]$*, then the probability that* $h$ *is not injective from* $S$ *is upper bounded by* $1/2$*.*

Now we are ready to complete our algorithm for the p-MaxMatch problem. Let $M^*$ be the matching in the graph $G$ defined in (1). The matching $M^*$ contains at most $k + 1$ edges, so the number of vertices in $V(M^*)$ of the matching $M^*$ is bounded by $k_2 = 2k + 2$. Therefore, if we have a universal set $\mathcal{H}_u$ of hash functions from $V(G)$ to $[(2k + 2)^2] = [k_2^2]$, then for $r$ hash functions randomly picked from $\mathcal{H}_u$, the probability that all these $r$ hash functions are not injective from the set $V(M^*)$ is upper bounded by $1/2^r$. Thus, the probability that at least one of these $r$ hash functions is injective from $V(M^*)$ is at least $1 - 1/2^r$. In particular, for any given constant $\epsilon > 0$, if we let $r = \lceil \log(1/\epsilon) \rceil$, then the probability that at least one of the $r$ hash functions randomly picked from the universal set $\mathcal{H}_u$ of hash functions is injective from $V(M^*)$ is at least $1 - \epsilon$.

Note that by our assumption, $V(G) = [|V(G)|]$. To have a universal set of hash functions, let $p_0$ be a fixed prime number such that $p_0 \geq |V(G)|$.

**Proposition 2 ([7]).** *Let* $\mathcal{H}_u = \{h_{a,b} \mid 1 \leq a \leq p_0 - 1, 0 \leq b \leq p_0 - 1\}$*, where each hash function* $h_{a,b}$ *is defined as:* $h_{a,b}(v) = ((a \cdot v + b) \bmod p_0) \bmod k_2^2$*. Then the set* $\mathcal{H}_u$ *is a universal set of hash functions from* $V(G)$ *to* $[k_2^2]$*.*

Each hash function $h_{a,b}$ in $\mathcal{H}_u$ is given by the two integers $a$ and $b$, thus, can be represented in space $O(1)$. To randomly pick a hash function from $\mathcal{H}_u$ can

be implemented by randomly picking two integers $a$ and $b$ with $1 \leq a \leq p_0 - 1$ and $0 \leq b \leq p_0 - 1$, which takes (randomized) constant time. Moreover, given a hash function $h_{a,b}$ in $\mathcal{H}_u$ and a vertex $v \in V(G)$, we can compute the hash value $h_{a,b}(v)$ in constant time. Thus, all assumptions for the analysis of the complexity of the algorithm **Match-0**, as given in the paragraph before Theorem 2, can be satisfied if we use hash functions in the universal set $\mathcal{H}_u$. As a result, the complexity of the algorithm **Match-0** as given in Theorem 2 holds true.

We summarize the above discussions in the algorithm given in Fig. 2.

**Algorithm Match**$(G, k, \epsilon)$ \\\ with success probability at least $1 - \epsilon$
INPUT: a graph $G$, a parameter $k$, and a probability error bound $\epsilon > 0$
OUTPUT: the matching $M^*$ defined in (1).

1. randomly pick $b = \lceil \log(1/\epsilon) \rceil$ hash functions from the universal set $\mathcal{H}_u$ of hash functions mapping $V(G)$ to $[k_2^2]$, let $\mathcal{H}_0$ be the set of the picked hash functions;
2. $(M[i], h_i, H[i, 0..k_2^2 - 1]) =$ **Match-0**$(G, k, \mathcal{H}_0)$;
3. return$(M[i])$.

**Fig. 2.** An algorithm for the p-MaxMatch problem

When $\epsilon > 0$ is a fixed constant, the number $b = \lceil \log(1/\epsilon) \rceil$ of hash functions picked in step 1 of the algorithm **Match** is also a constant. Combining this with Theorems 1-2, we conclude with the following theorem (recall that $k_2 = 2k + 2$):

**Theorem 3.** *For any fixed error bound $\epsilon > 0$, there is a randomized algorithm that solves the* p-MaxMatch *problem with a probability $\geq 1 - \epsilon$, and runs in time $O(N)$ and space $O(k^2)$ on a graph $G$ of size $N$ and a parameter $k$.*

**Remark 2.** The triple $(M[i], h_i, H[i, 0..k_2^2 - 1])$ constructed in step 2 of the algorithm **Match**$(G, k, \epsilon)$ gives a good data structure for the matching $M^* = M[i]$, where the array $H[i, 0..k_2^2 - 1]$ represents the vertex set $V(M^*)$ of the matching $M^*$ via the hash function $h_i$ that is injective from $V(M^*)$. In particular, for any vertex $v$ in the graph $G$, by checking if $H[i, h_i(v)] = v$, we can test in constant time if the vertex $v$ is in the vertex set $V(M^*)$ of the matching $M^*$.

**Remark 3.** Algorithms **Match-0** and **Match** can actually be regarded as 1-pass streaming algorithms on the insert-only model [16] for the p-MaxMatch problem. The algorithm **Match-0** (thus also the algorithm **Match**) reads in step 3 the edges of $G$ in the given order, and never needs to re-read the edges. Thus, if the edges of the graph $G$ are given as a stream, then the algorithms still work fine. These stream algorithms take space $O(k^2)$ and update time $O(1)$ (the *update time* of a streaming algorithm is the time spent on processing each element in the stream, which, in the algorithm **Match-0**, is by steps 3.1–3.2).

## 3 Kernelization for Edge Dominating Set

In this section, we study kernelization algorithms for the NP-hard problem Edge Dominating Set on the proposed model.

Let $G$ be a graph. An edge $e$ in $G$ *dominates* another edge $e'$ if $e$ and $e'$ share a common endpoint. Note that an edge always dominates itself and that if $e$ dominates $e'$ then $e'$ also dominates $e$. An edge set $E_1$ *dominates* another edge set $E_2$ if every edge in $E_2$ is dominated by an edge in $E_1$. Note that the edge set $E_1$ dominating the edge set $E_2$ does not necessarily imply that $E_2$ dominates $E_1$. An *edge dominating set* of a graph $G$ is an edge set in $G$ that dominates all edges in $G$. The *size* of an edge dominating set $D$ is the number of edges in $D$.

The (parameterized) Edge Dominating Set problem (abbr. p-EDS) consists of instances of the form $(G, k)$, asking if the graph $G$ has an edge dominating set of size bounded by $k$. The p-EDS problem is NP-complete [13]. A *kernelization algorithm* for p-EDS is a polynomial-time algorithm that on an instance $(G, k)$ of p-EDS produces an "equivalent" instance $(G', k')$ of p-EDS such that; (1) $(G, k)$ is a yes-instance of p-EDS if and only if $(G', k')$ is a yes-instance of p-EDS; and (2) both the size of the new graph $G'$ and the new parameter $k'$ are upper bounded by a function $g(k)$ of the input parameter $k$ that is independent of the size of the graph $G$. The graph $G'$ in the output $(G', k')$ of the kernelization algorithm is called the *kernel* constructed by the kernelization algorithm.

There has been a series of research work on kernelization algorithms for the p-EDS problem. Fernau [11] proposed a kernelization algorithm for p-EDS that gives a kernel of at most $8k^2$ vertices. Rodriguez's kernelization algorithm for p-EDS [17] obtained an improved kernel with at most $4k^2 + 8k$ vertices. Xiao, Kloks, and Poon [18] proposed a linear-time kernelization algorithm for p-EDS yielding a kernel with at most $2k^2 + 2k$ vertices and $O(k^3)$ edges. Hagerup [14] presented a further improved linear-time kernelization algorithm for p-EDS that gives a kernel of at most $\max\{k^2/2 + 7k/2, 6k\}$ vertices and at most $8k^3/27 + O(k^2)$ edges. The linear-time algorithms in [14,18] require $O(N)$ space on input graphs of size $N$. Fafianie and Kratsch [8] studied parameterized streaming algorithms and presented a 2-pass streaming kernelization algorithm of space $O(k^3 \log k)$ for the p-EDS problem that gives a kernel of size $O(k^3 \log k)$.[1]

In this section, we will assume that the graph $G$ is given in an adjacency list.

### 3.1 The First Kernelization Algorithm for p-EDS

We first study a kernelization algorithm for the p-EDS problem on our model, following the ideas given in [18] that gave a kernerlization algorithm for p-EDS running in linear time and linear space. Note that it can be challenging to keep the linear time while limiting the space complexity: when we limit the space complexity, many constant time operations used in the linear-time algorithms in [14,18], such as checking if a vertex is in a given set, can no longer be done in constant time, which will cause the algorithms to run in super-linear time.

Our first kernelization algorithm for p-EDS is given in Fig. 3, which uses the algorithm **Match-0** developed in the previous section, where $k_2 = 4k + 2$.

[1] It is known [5] that (even randomized) 1-pass streaming algorithms for the p-EDS problem require $\Omega(N)$ space. As a result, 1-pass streaming algorithms for the problem have become less interesting.

By Theorems 1 and 3, with a probability at least $1-\epsilon$, step 2 of the algorithm **KerEDS1**$(G,k)$ obtains a triple $(M^*, h, H[0..k_2^2-1])$, where $M^*$ is a matching of at most $2k+1$ edges in $G$ that is a solution to the instance $(G,2k)$ of the P-MAXMATCH problem, $h$ is a hash function from $V(G)$ to $[k_2^2]$ (where $k_2 = 4k+2$) that is injective from $V(M^*)$, and the array $H[0..k_2^2-1]$ represents $V(M^*)$ via $h$. In particular, if $|M^*| \leq 2k$, then $M^*$ is a maximal matching in the graph $G$.

**Algorithm KerEDS1**$(G,k)$ \\ with success probability at least $1-\epsilon$
INPUT: an instance $(G,k)$ of P-EDS
OUTPUT: an instance $(G_0,k)$ of P-EDS that is equivalent to $(G,k)$

1. randomly pick $b = \lceil \log(1/\epsilon) \rceil$ hash functions from a universal set of hash functions from $V(G)$ to $[k_2^2]$, let $\mathcal{H}_0$ be the set of the picked hash functions; \\ $k_2 = 4k+2$
2. $(M^*, h, H[0..k_2^2-1]) =$ **Match-0**$(G, 2k, \mathcal{H}_0)$;
3. **if** $(|M^*| > 2k)$ **then** stop ("no edge dominating set of size $\leq k$") **else** $m = |M^*|$;
4. $V_l^+ = V_l \cup V_1$, where
   $V_l$ is the set of vertices $v$ in $V(M^*)$ such that $|\{[v,w] \mid w \notin V(M^*)\}| > 2k-m$, and
   $V_1$ is the set of vertices $v$ in $V(M^*)$ such that $v$ has a neighbor of degree 1;
5. let $G_0 = G(M^*)$; \\ $G(M^*)$ is the subgraph induced by $M^*$
6. **for** (each vertex $v$ in $V_l^+$)
   add an edge $[v,v']$ to the graph $G_0$, where $v'$ is a new vertex of degree 1;
7. **for** (each vertex $v \notin V(M^*)$)
   **if** ($v$ has a neighbor not in $V_l^+$) **then** add $v$ and all its incident edges to $G_0$;
8. return $(G_0,k)$.

**Fig. 3.** First kernelization algorithm for the P-EDS problem

**Proposition 3 ([18]).** *Let $(G_0,k)$ be the output of algorithm* **KerEDS1**$(G,k)$. *Then $(G,k)$ is a yes-instance of* P-EDS *if and only if $(G_0,k)$ is a yes-instance of* P-EDS, *where the graph $G_0$ has at most $2k^2+2k$ vertices and $O(k^3)$ edges.*

**Theorem 4 ($\diamondsuit$).** *For any fixed $\epsilon > 0$, the algorithm* **KerEDS1**$(G,k)$ *is a randomized kernelization algorithm for the* P-EDS *problem that produces the kernel $(G_0,k)$ in time $O(N)$ and space $O(k^3)$, with a success probability at least $1-\epsilon$.*

**Remark 4.** The algorithm **KerEDS1** can be implemented so that it only makes two linear-time scans on the edges of the graph $G$, taking constant time per edge. Therefore, the algorithm can be implemented as a 2-pass streaming kernelization algorithm for P-EDS. Fafianie and Kratsch [8] have studied streaming kernelization algorithms for P-EDS. A 2-pass streaming kernelization algorithm for P-EDS with update time $O(k^2)$ was presented (see [8], Theorem 5). It was mentioned, with no detailed descriptions (see [8], Corollary 4), that the update time of the algorithm could be improved to $O(\log k)$. In any case, the algorithms given in [8] run in super-linear time in terms of the input graph size. In comparison, our algorithm **KerEDS1**, if implemented as a streaming algorithm, uses the same number of passes and has the same space complexity as those in [8], but has better update time $O(1)$. On the other hand, we mention that the algorithms given in [8] are deterministic while our algorithm **KerEDS1** is randomized.

The running time of the algorithm **KerEDS1** depends on the probability error bound $\epsilon > 0$: by Theorem 2 and step 1 of the algorithm **KerEDS1**, to achieve a success probability $1 - \epsilon$, the algorithm **Match-0** will take time $O(N \log(1/\epsilon))$ and space $O(k^3)$. Therefore, the algorithm will no longer run in linear time if we want to achieve a success probability $1 - o(1)$.

### 3.2 A New Kernelization Algorithm for P-EDS

In this subsection, we present a new randomized kernelization algorithm for the P-EDS problem, which basically keeps the same time and space complexity as that of the algorithm **KerEDS1**, but has a much high success probability. Moreover, we will show how to use this algorithm to achieve kernels whose size matches the best known kernel size for P-EDS, without increasing the time and space complexity and decreasing the success probability.

A vertex $v$ in $G$ is a *large-vertex* if its degree is larger than $2k$. Otherwise, $v$ is a *small-vertex*. An edge is a *large-edge* if its both endpoints are large-vertices, and an edge is a *small-edge* if its both endpoints are small-vertices. Therefore, the vertices of the graph $G$ are classified into large-vertices and small-vertices, and there are three kinds of edges: large-edges, small-edges, and edges whose one endpoint is a large-vertex and the other endpoint is a small-vertex.

**Lemma 1 ($\diamondsuit$).** *Every large-vertex must be contained in every edge dominating set of size bounded by $k$. As a result, the number of large-vertices is upper bounded by $2k$ if the graph $G$ has an edge dominating set of size bounded by $k$.*

Our new kernelization algorithm, **KerEDS2**, for the P-EDS problem is given in Fig. 4. Lemma 1 ensures the correctness of step 2 of this algorithm. The major difference between this algorithm **KerEDS2** and the algorithm **KerEDS1** is that the vertex set $V(M^*)$ of the matching $M^*$ is unknown to the algorithm **KerEDS1** when it chooses hash functions in step 1, while the vertex set $V_L$ is known to the algorithm **KerEDS2** when it chooses the hash function in step 3. This difference enables us to significantly improve the success probability.

**Algorithm KerEDS2**$(G, k)$
INPUT: an instance $(G, k)$ of P-EDS
OUTPUT: an instance $(G_0, k)$ of P-EDS that is equivalent to $(G, k)$

1. collect all large-vertices in $G$, and store them in $V_L$;
2. **if** $(|V_L| > 2k)$ **then** stop ("no edge dominating set of size $\leq k$") **else** $k_L = |V_L|$;
3. construct a hash function $h$ from $V(G)$ to $[k_L^2]$ and an array $H[0..k_L^2 - 1]$ such that $h$ is injective from $V_L$ and $H[0..k_L^2 - 1]$ represents $V_L$ via $h$;
4. collect all small-edges and store them in $E_S$;
5. **if** $(|E_S| > k(4k - 1))$ **then** stop ("no edge dominating set of size $\leq k$");
6. delete all small-vertices that have all neighbors in $V_L$;
7. **for** (each vertex $v$ in $V_L$) add a new degree-1 vertex $v^*$ and a new edge $[v, v^*]$;
8. let the resulting graph be $G_0$, return $(G_0, k)$.

**Fig. 4.** A new kernelization algorithm for the P-EDS problem.

**Lemma 2 (♢).** *An edge can dominate at most $4k-1$ edges in the set $E_S$. Thus, if $|E_S| > k(4k-1)$, then the graph $G$ has no edge dominating set of size $\leq k$.*

**Lemma 3 (♢).** *The graph $G_0$ constructed by the algorithm* **KerEDS2**$(G,k)$ *has $O(k^2)$ vertices and $O(k^3)$ edges.*

**Theorem 5 (♢).** *The graph $G$ has an edge dominating set of size $\leq k$ if and only if the graph $G_0$ constructed by the algorithm* **KerEDS2**$(G,k)$ *has an edge dominating set of size $\leq k$.*

By Theorem 5, the algorithm **KerEDS2** is a kernelization algorithm for the P-EDS problem. In the following, we analyze the complexity of the algorithm.

**Theorem 6.** *The algorithm* **KerEDS2**$(G,k)$ *is a randomized kernelization algorithm for the* P-EDS *problem that runs in time $O(N)$ and space $O(k^3)$, and, with a success probability at least $1-1/2^{k+N/k} \geq 1-1/4^{\sqrt{N}}$, constructs a kernel of $O(k^2)$ vertices and $O(k^3)$ edges.*

*Proof.* After the algorithm obtained the vertex set $V_L$ in step 1 and verified that $k_L = |V_L| \leq 2k$ in step 2, its step 3 repeatedly picks a random hash function $h$ from a universal set $\mathcal{H}_u$ of hash functions that map $V(G)$ to $[k_L^2]$, and checks if $h$ is injective from $V_L$, until a hash function $h$ from $\mathcal{H}_u$ is obtained such that $h$ is injective from $V_L$. Note that picking a random hash function from $\mathcal{H}_u$ takes (randomized) $O(1)$ time. To check if a hash function $h$ from $V(G)$ to $[k_L^2]$ is injective from $V_L$, we can use an array $H[0..k_L^2-1]$ to record the values $h(v)$ for all $v \in V_L$ to see if two vertices in $V_L$ collide under the hash function $h$. Thus, checking if a hash function $h$ is injective from $V_L$ takes time $O(|V_L|) = O(k)$. Suppose that step 3 of the algorithm **KerEDS2** tries at most $b$ randomly picked hash functions from $\mathcal{H}_u$, then step 3 of the algorithm will take time $O(bk+k^2)$ and space $O(k^2)$, where the space complexity $O(k^2)$ is for storing the array $H[0..k_L^2-1]$ and the term $k^2$ in the time complexity is the time to initialize the array $H[0..k_L^2-1]$. By Proposition 1, the probability that step 3 of the algorithm finds a hash function $h$ from $\mathcal{H}_u$ that is injective from $V_L$ is at least $1-1/2^b$. Note that once such a hash function $h$ is found, the array $H[0..k_L^2-1]$ described above becomes the array that represents the vertex set $V_L$ via the hash function $h$. In consequence, checking the membership of the set $V_L$ now takes time $O(1)$.

With the data structure built in step 3 that supports checking the membership of the vertex set $V_L$ in constant time, now checking if an edge is in the set $E_S$ in step 4 takes constant time, and checking if a vertex $v$ is a small-vertex with all neighbors in $V_L$ in step 6 takes time $O(d_G(v))$, where $d_G(v) \leq 2k$ is the degree of the small-vertex $v$. Therefore, another linear-time scanning will be sufficient to complete all steps 4–8 of the algorithm **KerEDS2**.

Summarizing the above discussions combined with Lemma 3, we conclude that the algorithm **KerEDS2**$(G,k)$ runs in time $O(N+bk+k^2)$ and space $O(k^3)$ and has a success probability at least $1-1/2^b$. Setting $b=(N+k^2)/k$, the algorithm running time becomes $O(N+k^2)$, and the success probability

becomes $1 - 1/2^{(N+k^2)/k}$. Finally, by comparing $N$ and $k^2$, using a technique we also used in Theorem 2, we can simplify the running time to $O(N)$. □

The error bound $1/4^{\sqrt{N}}$ of the algorithm **KerEDS2** given in Theorem 6 is very small because $N$ is assumed to be very large. On the other hand, the size of the kernel constructed by the algorithm **KerEDS2** is not as good as that by the algorithm **KerEDS1**, as given in Proposition 3. This, however, can be easily overcome: once we obtain the kernel $(G_0, k)$ by the algorithm **KerEDS2**$(G, k)$, where the size $N_1$ of the graph $G_0$ is $O(k^3)$, we can run any known linear-time kernelization algorithms for p-EDS on $(G_0, k)$, such as those in [14,18], which take time $O(N_1) = O(k^3)$ (thus also in space $O(k^3)$), and produce a kernel of better size. In particular, if we use the algorithm in [14] that gives the currently best kernel size for p-EDS on the instance $(G_0, k)$, then with the additional $O(k^3)$ time and $O(k^3)$ space, we can get the kernel whose size matches the best known kernel size given in [14]. We summarize this discussion as follows:

**Theorem 7.** *There is a randomized kernelization algorithm for the* p-EDS *problem that on an instance* $(G, k)$ *of* p-EDS*, runs in time* $O(N)$ *and space* $O(k^3)$*, and, with a success probability at least* $1 - 1/4^{\sqrt{N}}$*, constructs a kernel of at most* $\max\{k^2/2 + 7k/2, 6k\}$ *vertices and at most* $8k^3/27 + O(k^2)$ *edges.*

**Remark 5.** Again the algorithm in Theorem 7 can be implemented as a 2-pass streaming kernelization algorithm for p-EDS, with space complexity $O(k^3)$ and update time $O(1)$. To achieve $O(1)$ update time, we need to be more careful in step 6 of the algorithm **KerEDS2**, using the technique of pipelines (see [3]). We point out that although our algorithm has constant update time, it does require non-constant time between the two passes to find an injective hash function (see step 3 of the algorithm **KerEDS2**), and, if we want to achieve a kernel size matching the best known one, also non-constant time after the second pass to apply the kernelization on the instance constructed by the algorithm **KerEDS2**.

## 4 Conclusion and Final Remarks

Motivated by recent research in massive data processing, we studied a computational model whose complexity is measured by the very large input size $N$ and a parameter $k$ that characterizes the limited power of local resources. We presented algorithms for well-known graph problems and studied new algorithmic techniques on the model. In particular, we developed a randomized algorithm of time $O(N)$ and space $O(k^2)$ that constructs a maximal matching of size bounded by $k$ in a graph of size $N$, and a randomized kernelization algorithm of time $O(N)$ and space $O(k^3)$ for the NP-hard Edge Dominating Set problem. Our kernelization algorithm for Edge Dominating Set has its kernel size match the best one by known polynomial-time kernelization algorithms without space constraints for the problem. Moreover, as we discussed in Remarks 4–5 (see also [3]), the techniques developed in our study can be applied to develop streaming algorithms with improved update time for some well-known graph problems.

The computational model we studied suggests reconsideration for many computational problems, including many classical ones such as Graph Matching [2], in the framework of massive data processing. A particular area where the model can be investigated is kernelization algorithms [12], for which here we studied a particular problem Edge Dominating Set. Most proposed kernelization algorithms run in polynomial time and were developed without much focus on detailed efficiency and with space complexity rarely considered. On the other hand, the approach of kernelization seems to fit very well in dealing with massive data, and provides pre-processing techniques to reduce large problem instances to much smaller (thus manageable) instances. Kernelization algorithms whose running time is linear or nearly linear in terms of the input size, with limited space complexity, are very interesting in this direction of research.

## References

1. https://followthedata.wordpress.com/2014/06/24/data-size-estimates/
2. Chen, J., Guo, Y., Huang, Q.: Linear-time parameterized algorithms with limited local resources. Inf. Comput. **289**, 104951 (2022). https://doi.org/10.1016/j.ic.2022.104951
3. Chen, J., Huang, Q., Kanj, I.A., Li, Q., Xia, G.: Streaming algorithms for graph $k$-matching with optimal or near-optimal update time. In: Proceedings of 32nd International Symposium on Algorithms and Computation (ISAAC 2021), Article No. 48, pp. 48:1–48:17 (2021)
4. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41–2**, 280–301 (2001)
5. Chitnis, R., Cormode, G., Esfandiari, H., Hajiaghayi, M., McGregor, A., Monemizadeh, M.: Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In: Proceedings of 27th ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), pp. 1326–1344 (2016)
6. Chitnis, R., Cormode, G., Hajiaghayi, M.T., Monemizadeh, M.: Parameterized streaming: maximal matching and vertex cover. In: Proceedings of 26th ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), pp. 1234–1251 (2015)
7. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge, MA (2009)
8. Fafianie, S., Kratsch, S.: Streaming kernelization. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014. LNCS, vol. 8635, pp. 275–286. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44465-8_24
9. Fan, W., Geerts, F., Neven, F.: Making queries tractable on big data with preprocessing. In: Proceedings of 39th International Conference on Very Large Data Bases, pp. 685–696 (2013)
10. Fan, W., Hu, C.: Big graph analysis: from queries to dependencies and association rules. Data Sci. Eng. **2**(1), 36–55 (2017)
11. Fernau, H.: Edge dominating set: efficient enumeration-based exact algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 142–153. Springer, Heidelberg (2006). https://doi.org/10.1007/11847250_13
12. Fomin, F., Lokshtanov, D., Saurabh, S., Zehavi, M.: Kernelization: Theory of Parameterized Preprocessing. Cambridge University Press, Cambridge (2019)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)

14. Hagerup, T.: Kernels for edge dominating set: simpler or smaller. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 491–502. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_44
15. Malewicz, G., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD 2010), pp. 135–145 (2010)
16. McGregor, A.: Graph stream algorithms: a survey. ACM SIGMOD Rec. **43**(1), 9–20 (2014)
17. Rodriguez, E.P.: Systematic kernelization in FPT algorithm Design, Ph.D. Dissertation, The University of Newcastle (2013)
18. Xiao, M., Kloks, T., Poon, S.-H.: New parameterized algorithms for the edge dominating set problem. Theor. Comput. Sci. **511**, 147–158 (2013)