Sublinear Algorithms for Processing Massive Datasets

Hamed Saleh
May 2020
Publications

- Externalities and Fairness [WWW’19]

- Streaming and Massively Parallel Algorithms for Edge Coloring [ESA’19]
  (also appeared in [DISC’19] as a brief announcement)

- Massively Parallel Algorithms for String Matching with Wildcards [arXiv]

- Computational Analyses of the Electoral College: Campaigning is Hard But Approximately Manageable [preprint]
Publications

- Externalities and Fairness [WWW’19]

- Streaming and Massively Parallel Algorithms for Edge Coloring [ESA’19]
  (also appeared in [DISC’19] as a brief announcement)

- Massively Parallel Algorithms for String Matching with Wildcards [arXiv]

- Computational Analyses of the Electoral College: Campaigning is Hard But Approximately Manageable [preprint]
Sublinear Space

Have you ever had a dataset so big that it doesn’t fit in the memory?

Sublinear Space Algorithms!

\( o(n) \)
Sublinear Space

Have you ever had a dataset so big that it doesn’t fit in the memory?

Sublinear Space Algorithms!

$\text{RAM model}$
Sublinear Space

Have you ever had a dataset so big that it doesn’t fit in the memory?

Sublinear Space Algorithms!

Alternative models
Alternative Models of Computation
Massively Parallel Computation

- Modern frameworks for Large-scale parallel/distributed data processing: MapReduce, Hadoop, Spark.
- Key Idea: distribute the workload among several machines.
- The MPC model: A theoretical model to abstract out the computational power of these frameworks.

[Karloff et. al 2010]
[Goodrich et. al 2011]
[Beame et. al 2013]
[Andoni et. al 2014]
Massively Parallel Computation

- There are $M$ machines each with memory $S$.
- The input of length $N$ is initially (randomly) distributed among the machines.
  - Sublinear space $S = o(N)$.
  - Usually $N = O(S \cdot M)$.
- The data is processed in several synchronous rounds.
Massively Parallel Computation

- There are $M$ machines each with memory $S$.
- The data is processed in several synchronous rounds. In each round,
  - Machines perform arbitrary computation on their local data.
  - Machines communicate with each other. Total incoming/outgoing messages of each machine is bounded by $O(S)$ words.
Massively Parallel Computation

- There are $M$ machines each with memory $S$.
- The data is processed in several synchronous rounds.
- Main bottleneck: Communication. We wish for algorithms with very small number of rounds.
  - Often sub-logarithmic rounds.
Related distributed/parallel models

- The PRAM model (shared memory)
  - Any PRAM algorithm running in time $t = t(n)$ can be simulated in $O(t)$ MPC rounds. [Karloff et. al 2010]

- The LOCAL, CONGEST, and congested-clique models
  - Similar techniques can be used for both MPC and these distributed models.
  - Congested-clique is almost equivalent to MPC (in terms of the number of rounds). [Behnezhad et. al 2018]
Streaming

- There is a single machine with memory $S$.
- The input of length $N$ is streamed into the machine.
  - Sublinear space $S = o(N)$.
- The data is processed in several passes.
There is a single machine with memory $S$.

The data is processed in several passes. In each pass:

- The input entries arrive sequentially and one by one in a specific order.
- The order can be either random or adversarial.
Streaming

- There is a single machine with memory $S$.
- The data is processed in several passes.
- There is a trade-off between the number of passes and the space of the machine.
There is a single machine with memory $S$.

What if the output also doesn’t fit in the memory?

- We use the W-streaming model.
- The output is also streamed.
Semi-streaming and Semi-MPC

- In many graph problems, we assume $S = O(|V|)$, where the input graph is given as $G = (V, E)$.

- The variant of the streaming model in which $S = O(|V|)$ is called Semi-streaming.

- This restriction is stricter on dense graphs and less strict on sparse graphs.

- The standard variants are either too trivial or too hard on sparse graphs.

- Similarly, the Semi-MPC model is also defined.
Edge Coloring

Streaming and Massively Parallel Algorithms for Edge Coloring [ESA’19]
Behnezhad, Derakhshan, Knittel, Hajiaghayi, me
The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a valid “edge-coloring” is a function $\text{COL}: E \rightarrow [\Psi]$ so that no two incident edges have a common color.

- We wish to minimize $\Psi$, i.e., the number of colors.
The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a valid “edge-coloring” is a function $\text{COL}: E \rightarrow [\Psi]$ so that no two incident edges have a common color.

- Let $\Delta$ be the maximum degree of graph $G$.

- Then, we know $\Delta \leq \Psi \leq \Delta + 1$.  

  [Vizing 1964]

- An odd cycle ($\Delta = 2$) needs 3 colors.
The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a valid “edge-coloring” is a function $\text{COL} : E \to [\Psi]$ so that no two incident edges have a common color.

- However, $(\Delta + 1)$-coloring algorithms are highly **sequential**.

- There is a greedy $(2\Delta - 1)$-coloring algorithm.
The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a valid “edge-coloring” is a function $\text{COL}: E \rightarrow [\Psi]$ so that no two incident edges have a common color.

- There is a greedy $(2\Delta - 1)$-coloring algorithm.
  - Process edges in an arbitrary order.
  - Color each edge with an available color.
Related work

- Vertex Coloring: Assadi et. Al
- Distributed Ghaffari et al
- Harvey et. Al
- Streaming?
Edge Coloring

Massively Parallel Computation
A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

There is a constant-round MPC algorithm, with $S = O(n)$ and $S \cdot M = O(m)$, which computes an edge-coloring so that $\Psi = \Delta + \tilde{O}(\Delta^{3/4})$. \[\text{BDKHS 2019}\]
**Semi-MPC Edge Coloring**

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

- There is a constant-round MPC algorithm, with $S = O(n)$ and $S \cdot M = O(m)$, which computes an edge-coloring so that $\Psi = \Delta + \tilde{O}(\Delta^{3/4})$. [BDKHS 2019]

- It is technically a Semi-MPC algorithm.
Semi-MPC Edge Coloring

- It is technically a Semi-MPC algorithm, but we can achieve $o(n)$ space in dense graphs.
- The exact space-per-machine is equal to
  $$S = O\left(\frac{n\Delta}{k^2} + \frac{n}{k} \sqrt{\frac{\Delta}{k} \log n}\right)$$
- Set $k = \sqrt{\Delta} + \log n$. 

![Diagram of Semi-MPC Edge Coloring](image)
**Vertex Partitioning**

- Set $k = \sqrt{\Delta} + \log n$.

- Random vertex partitioning: $V = V_1 \cup V_2 \cup \ldots \cup V_k$. 
**Vertex Partitioning**

- Set $k = \sqrt{\Delta} + \log n$.

- Random vertex partitioning: $V = V_1 \cup V_2 \cup \ldots \cup V_k$.

- $G_{i,i} = \{ (u, v) \mid u \in V_i \land v \in V_i \}$: the **induced** subgraph of each partition
Vertex Partitioning

- Random vertex partitioning: \( V = V_1 \cup V_2 \cup \ldots \cup V_k \).
- \( G_{i,i} = \{ (u, v) \mid u \in V_i \land v \in V_i \} \): the induced subgraph of each partition
- \( G_{i,j} = \{ (u, v) \mid u \in V_i \land v \in V_j \} \): the bipartite induced subgraph of pairs of partitions
Local Coloring

- $G_{i,i} = \{ (u, v) \mid u \in V_i \land v \in V_i \}$: the \textbf{induced} subgraph of each partition
- $G_{i,j} = \{ (u, v) \mid u \in V_i \land v \in V_j \}$: the \textbf{bipartite induced} subgraph of pairs of partitions
- Run Vizing’s $(\Delta + 1)$-coloring algorithm on each machine.
Merging Colored Subgraphs

- $k + 1$ disjoint color palettes are enough.

- The number of colors in each palette needs to be as large as the maximum degree in subgraphs.

- The maximum degree in each subgraphs is concentrated.

- For $k = \sqrt{\Delta} + \log n$, we have $\Psi = \Delta + \tilde{O}(\Delta^{3/4})$. 
Edge Coloring
Random Streaming
Random Streaming Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

- There is a one-pass $W$-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = (2e + \epsilon)\Delta$.

[BDKHS 2019]
Random Streaming Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

- There is a one-pass $W$-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = (2e + \epsilon)\Delta$.

- The algorithm is straight-forward.

- Maintain a counter variable $c_v$ for each vertex, initially set to 1.
Random Streaming Edge Coloring

- Maintain a counter variable $c_v$ for each vertex, initially set to 1.
- Upon arrival of $(u, v)$ color it $\max(c_v, c_u)$.
Random Streaming Edge Coloring

- Maintain a **counter** variable $c_v$ for each vertex, initially set to 1.
- Upon arrival of $(u, v)$ color it $\max(c_v, c_u)$. 
Random Streaming Edge Coloring

- Maintain a counter variable $c_v$ for each vertex, initially set to $1$.  
- Upon arrival of $(u, v)$ color it $\max(c_v, c_u)$.  
- Set both $c_v$ and $c_u$ to $\max(c_v, c_u) + 1$. 

![Diagram of edge coloring](image)
Longest **Monotone** Path

- **Lemma:** $\Psi$ is equal to the length of the longest *monotone* path in the *line graph* after the algorithm finishes.

- We can construct a *monotone* path of length $\Psi$ starting from an edge colored $\Psi$. 

![Diagram of a line graph with colored vertices and edges](image)
Longest **Monotone** Path

- **Lemma:** $\Psi$ is equal to the length of the longest monotone path in the line graph after the algorithm finishes.

\[
\Pr[\Psi \geq \alpha \Delta] \leq \frac{(2\Delta)^{\alpha \Delta}}{(\alpha \Delta)!}
\]
Longest **Monotone** Path

- **Lemma:** $\Psi$ is equal to the length of the longest monotone path in the line graph after the algorithm finishes.

$$\Pr[\Psi \geq (2e + \epsilon)\Delta] \leq n^{-c}$$
Edge Coloring

Adversarial Streaming
Adversarial Streaming Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

- There is a one-pass $W$-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = O(\Delta^2)$.

[BDKHS 2019]
A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.

There is a one-pass $W$-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = O(\Delta^2)$.

The algorithm is very similar to the random stream algorithm.

Maintain a counter variable $c_v$ for each vertex, initially set to 1.
Bipartite Edge Coloring

- Maintain a counter variable $c_v$ for each vertex, initially set to 1. Also assume the graph is bipartite.

- Upon arrival of $(u, v)$ color it $(c_u, c_v)$. 
Bipartite Edge Coloring

- Maintain a **counter** variable $c_v$ for each vertex, initially set to 1. Also assume the graph is **bipartite**.

- Upon arrival of $(u, v)$ color it $(c_u, c_v)$.
Bipartite Edge Coloring

- Maintain a **counter** variable $c_v$ for each vertex, initially set to 1. Also assume the graph is **bipartite**.

- Upon arrival of $(u, v)$ color it $(c_u, c_v)$.

- Increase both $c_u$ and $c_v$ by 1.
General Edge Coloring

- A bipartite graph is colored with $\Delta^2$ colors using this method.

- Decompose the graph into $O(\log n)$ random bipartite graphs, and color each with a different palette.
General Edge Coloring

- A bipartite graph is colored with $\Delta^2$ colors using this method.
- Decompose the graph into $O(\log n)$ random bipartite graphs, and color each with a different palette.
- In total, $\Psi = O(\Delta^2)$.
String Matching

Massively Parallel Algorithms for String Matching with Wildcards [arXiv]
Hajiaghayi, me, Seddighin, Sun
The **String Matching** problem

- An essential problem in bio-informatics and many other areas.
- Given a text $T$ and a pattern $P$, we wish to find all substrings of $T$ that match $P$. 

```
T: AGCATTTCGCAAG
P: CA  CA
```
The **String Matching** problem

- Given a **text** $T$ and a **pattern** $P$, we wish to find all substrings of $T$ that match $P$.
- In the simplest form both $T$ and $P$ are using the same alphabet $\Sigma$, and there is no special character.

<table>
<thead>
<tr>
<th>T</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The **String Matching** problem

- Given a **text** $T$ and a **pattern** $P$, we wish to find all substrings of $T$ that match $P$.

- We also study the case when $P$ can also have special characters known as **wildcards**. We are interested in {‘?’ , ‘+’, ‘*’}.
String Matching in MPC

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq \Sigma^m$ are given.

- There is a constant-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HSSS 2019]
String Matching in **MPC**

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq \Sigma^m$ are given.

- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

  \[\text{HSSS 2019}\]

- Easy when $m = O(n^{1-x})$: **Double**-covering.
String Matching in **MPC**

- A *text* $T \subseteq \Sigma^n$ and a *pattern* $P \subseteq \Sigma^m$ are given.

- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HSSS 2019]

- Also easy for general $m$: Partial **hashing**.
String Matching

with ‘?’ wildcard
String Matching with ‘?’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{'?\}')^m$ are given.
- The special character ‘?’ can be replaced with any arbitrary character.
String Matching with ‘?’ wildcard

- A **text** $T \subseteq \Sigma^n$ and a **pattern** $P \subseteq (\Sigma \cup \{?\})^m$ are given.

- There is a **constant**-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HöSS 2019]
String Matching with ‘?’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{?\})^m$ are given.

- There is a constant-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

  [HSS 2019]

- This can be improved to any constant $x < 1$ at the cost of $O(x^{-1})$ rounds.
‘?’ wildcard and convolution

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup ‘?’)^m$ are given.

- This variant of the string matching problem can be reduced to the convolution of two arrays.

  [Fischer et. al 1974]

\[
T^\dagger = \{mp_{T_1}, mp_{T_1}^{-1}, mp_{T_2}, mp_{T_2}^{-1}, \ldots, mp_{T_n}, mp_{T_n}^{-1}\}
\]

\[
P^\dagger = \{mp_{P_1}, mp_{P_1}^{-1}, mp_{P_2}, mp_{P_2}^{-1}, \ldots, mp_{P_n}, mp_{P_n}^{-1}\}
\]

\[
mp_{‘?’} = mp_{‘?’}^{-1} = 0, mp_{‘C’} = 3, mp_{‘C’}^{-1} = 1/3
\]
‘?’ wildcard and convolution

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{'?\}')^m$ are given.

- This variant of the string matching problem can be reduced to the convolution of two arrays. [Fischer et. al 1974]

\[
T^t = \langle 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1}, 3, \frac{1}{3}, 1, \frac{1}{1}, 4, \frac{1}{4}, 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1} \rangle
\]

\[
P^t = \langle 1, \frac{1}{1}, 0, 0, 1, \frac{1}{1} \rangle
\]
A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{?\})^m$ are given.

$$T^\dagger = \langle 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1}, 3, \frac{1}{3}, 1, \frac{1}{1}, 4, \frac{1}{4}, 1, \frac{1}{1}, 2, \frac{1}{2}, 18, \frac{1}{18}, 1, \frac{1}{1} \rangle$$

$$P^\dagger = \langle 1, \frac{1}{1}, 0, 0, 1, \frac{1}{1} \rangle$$

$$C = T^\dagger * \text{rev}(P^\dagger)$$
FFT in constant rounds

- Performing a **bit-reversal** operation makes the divide and conquer pattern clean.

- It is easy to decompose the **precedence** graph into the **Butterfly** graphs of different sizes.

- Cooley-Tukey with radix $R = n^{1-x}$.

- Further implications such as the **knapsack** problem.
String Matching

with ‘+’ wildcard
String Matching with ‘+’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{+\})^m$ are given.

- The special character ‘+’ means that the preceding character can be repeated arbitrary times.
String Matching with ‘+’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{+\})^m$ are given.

- There is a constant-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HSS 2019]
Run Length Encoding

- A text \( T \subseteq \Sigma^n \) and a pattern \( P \subseteq (\Sigma \cup \{+\})^m \) are given.
- Perform Run Length Encoding on both strings.

\[
T^o = \langle \langle b, 1 \rangle, \langle o, 2 \rangle, \langle k, 2 \rangle, \langle e, 2 \rangle, \langle p, 1 \rangle, \langle e, 1 \rangle, \langle r, 1 \rangle \rangle
\]
\[
P^o = \langle \langle o, 2+ \rangle, \langle k, 1+ \rangle, \langle e, 2+ \rangle, \langle p, 1 \rangle \rangle
\]
- Reduces to Greater-than matching.
Run Length Encoding

- Reduces to Greater-than matching.
- A special case of Subset matching.
- The Subset matching problem can be solved in $O(n \log^2 m)$ by a careful reduction to sparse convolution. [Cole et. al 2002]
- It’s possible to implement it in $O(1)$ MPC rounds.
String Matching

with ‘*’ wildcard
String Matching with ‘*’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{\ast\})^m$ are given.

- The special character ‘*’ can be replaced with any arbitrary string.
String Matching with ‘*’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{'\ast'\})^m$ are given.

- The special character ‘*’ can be replaced with any arbitrary string.

- Unlike ‘*’ and ‘∗’, we have no positive result for this wildcard even in $O(\log n)$ rounds…

- There is a conjecture that we can’t solve graph connectivity in $o(\log n)$ rounds.
String Matching with ‘*’ wildcard

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{\ast\})^m$ are given.

- The special character ‘*’ can be replaced with any arbitrary string.

- Unlike ‘∗’ and ‘∗’, we have no positive result for this wildcard even in $O(\log n)$ rounds...

- But we can solve it in special cases.
‘*’ wildcard in small patterns

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{\ast\})^m$ are given such that $m = O(n^{1-x})$.

- There is an $O(\log n)$-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HSSS 2019]
‘*’ wildcard in no common prefix case

- A text $T \subseteq \Sigma^n$ and a pattern $P \subseteq (\Sigma \cup \{\ast\})^m$ are given such that no two sub-patterns share a common prefix.

- There is an $O(\log n)$-round MPC algorithm, with $S = O(n^{1-x})$ and $M = O(n^x)$ for any $x < 0.5$, which finds all occurrences of $P$ in $T$.

[HSSS 2019]
Future work

- The Hypergraph matching problem in MPC.
- Improving the edge-coloring bound in adversarial streams.
- A constant round MPC algorithm for weighted exact knapsack.
Acknowledgements

Co-Authors:
S. Behnezhad
S. Dehghani
M. Derakhshan
M. Ghodsi
M. Hajiaghayi
M. Knittel
S. Seddighin
M. Seddighin
X. Sun
S. Teng

Committee Members:
Prof. Hajiaghayi
Prof. Dickerson
Prof. Mount
Thanks for watching!

Any questions?