A Functional Approach to External Graph Algorithms

J. ABELLO, A. L. BUCHSBAUM, AND J. R. WESTBROOK

Motivation

Large data that exceeds main memory storage limits

Algorithms that require data locality need to be adapted when implementing their external versions, e.g. graph problems

Current approaches do not completely address I/O implications

Key Contributions

Divide-and-conquer approach for designing external graph algorithms

- No need for sophisticated data structures

Produce purely functional algorithms without “side effects”

- Lack of “side effects” allows for standard check-pointing
- Functional approach amenable to general programming language transformations that can reduce runtime

Other Previous Approaches

PRAM (Parallel Random Access Machine) Simulation

- Simulate CRCW (concurrent read concurrent write)
- PRAM using processor and external disk
- **Weakness**: no algorithm has been implemented with this

Buffering Data Structures

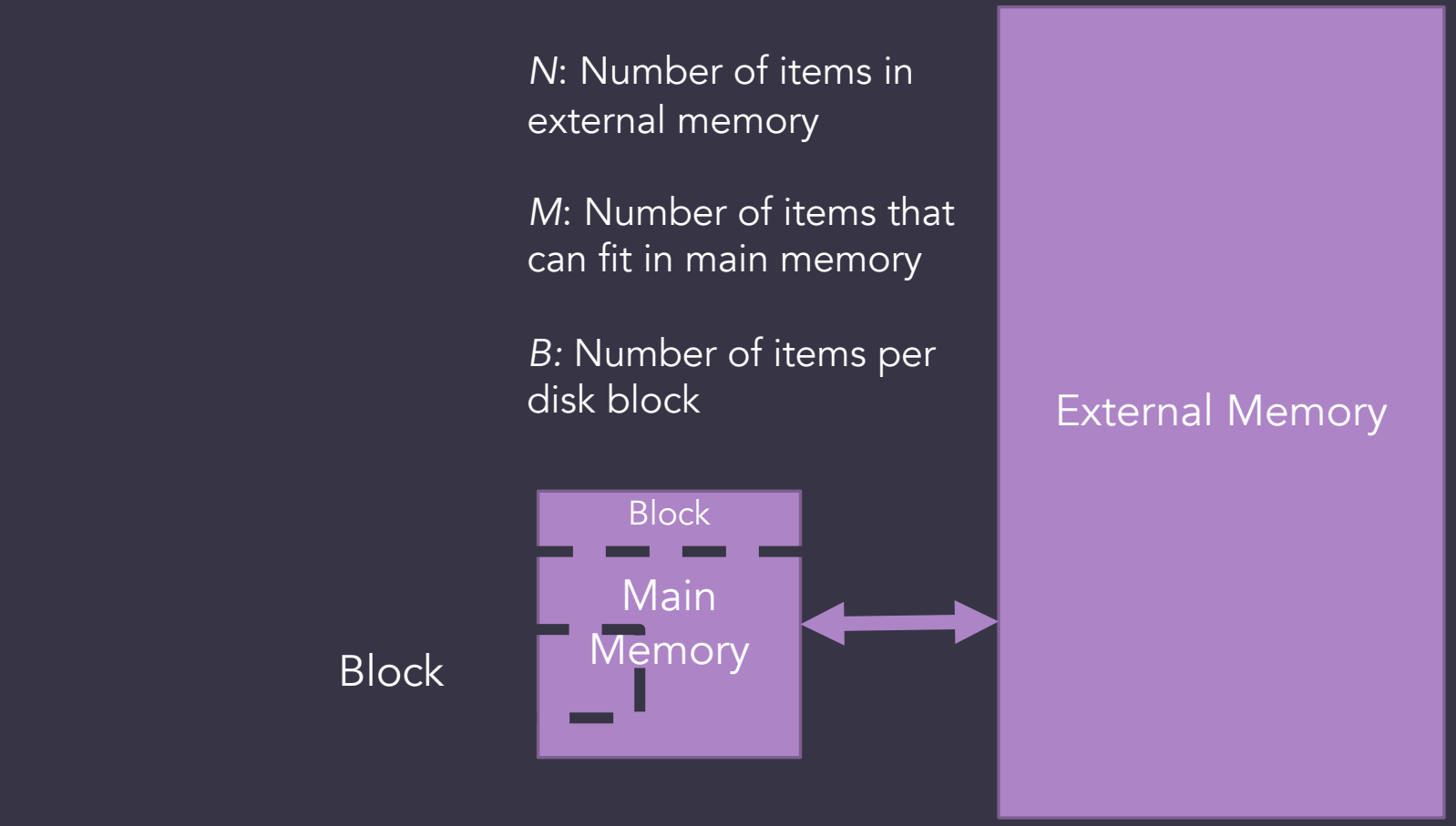
- External variants of classic data structures
- **Weakness**: Requires other structures to perform queries on-line, increasing I/O and implementation complexity

Definitions

N: Number of items in external memory

M: Number of items that can fit in main memory

B: Number of items per disk block



Functional I/O Model

Parameterized the same way as I/O model

Write-once property => enable easier recovery

Operations restricted to only make functional transformations to data => enables checkpointing

When intermediate results are no longer needed, space is reclaimed (e.g. garbage collect)

Problem Definitions

Connected components: edge set induced by maximal set of vertices such that each pair of vertices is connected by a path

Minimum spanning forest: edge set of minimal total weight that connects all vertices

Bottleneck minimum spanning forest: edge set of minimal maximum weight that connects all vertices

Maximal matching: maximal set of edges such that no two edges share a vertex

Maximal independent set: maximal set of vertices such that no two vertices share an edge

Overview of Results

$sort(N) = \Theta((N/B) \log_{M/B}(N/B))$

Table 1. I/O bounds for our functional external algorithms.

Problem	Deterministic	Randomized	
	I/O bound	I/O Bound	With probability
Connected components	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
MSFs	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
BMSFs	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
Maximal matchings	$O(\frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - \varepsilon$ for any fixed ε
Maximal independent sets		$O(sort(E))$	$1 - \varepsilon$ for any fixed ε

The Problems with Graph Problems

Need for data locality

On-line nature makes it difficult to utilize full disk block of edges once read

Difficult to divide a problem into independent sub-problems

Functional Graph Transformations

Contraction

Functional Graph Transformations

LEMMA 3.1. *If each pair in E' contains two vertices in the same connected component in G, then, for any $u, v \in V$, u and v are in the same connected component in G if and only if $s(u)$ and $s(v)$ are in the same connected component in G'.*

Functional Graph Transformations

Relabeling

Functional Graph Transformations

Connected Components

Connected Components Algorithm

Algorithm **CC**

1. Let E_1 be any half of the edges of G; let $G_1 = (V, E_1)$.
2. Compute $CC(G_1)$ recursively.
3. Let $G' = G/CC(G_1)$.
4. Compute $CC(G')$ recursively.
5. $CC(G) = CC(G') \cup RL(CC(G'), CC(G_1))$.

Generalization of Method as Functional Approach to External Graph Algorithms

$f_P(G)$: solution to the graph problem P on input graph $G = (V,E)$

$G_1 = S(G) \subseteq E$

T_1: transformation that combines G and the solution $f_P(G_1)$ to form subgraph G_2

T_2: transformation the maps $f_P(G_1)$ and $f_P(G_2)$ to $f_P(G)$

1. $G_1 \leftarrow S(G)$;
2. $G_2 \leftarrow T_1(G, f_{\mathcal{P}}(G_1))$;
3. $f_{\mathcal{P}}(G) = T_2(G, G_1, G_2, f_{\mathcal{P}}(G_1), f_{\mathcal{P}}(G_2))$.

Functional Implementations of Basic Methods

Selection

Relabeling

Contraction

Deletion

Functional Implementations of Selection

Select the *k*th largest element and it's multiplicity in *I*

1. Partition I into cM-element subsets, for some $0 < c < 1$.
2. Determine the median of each subset in main memory. Let S be the set of medians of the subsets.
3. $m \leftarrow$ `Select`$(S, \lceil S/2 \rceil)$.
4. Let I_1, I_2, I_3 be the sets of elements less than, equal to, and greater than m, respectively.
5. If $|I_1| \geq k$, then return `Select`(I_1, k).
6. Else if $|I_1| + |I_2| \geq k$, then return m.
7. Else return `Select`$(I_3, k - |I_1| - |I_2|)$.

LEMMA 3.3. `Select`(I,k) *can be performed in* $O(scan(|I|))$ *I/Os in the FIO model.*

Functional Implementations of Relabeling

1. Sort F by source vertex, v.
2. Sort I by second component.
3. Process F and I in tandem.
   (a) Let $\{s, h\} \in I$ be the current edge to be relabeled.
   (b) Scan F starting from the current edge until finding $(p(v), v)$ such that $v \geq h$.
   (c) If $v = h$, then add $\{s, p(v)\}$ to I''; otherwise, add $\{s, h\}$ to I''.
4. Repeat steps 2 and 3, relabeling first components of edges in I'' to construct I'.

LEMMA 3.4. *Relabeling an edge list I by a forest F can be performed in $O(sort(|I|) + sort(|F|))$ I/Os in the FIO model.*

Functional Implementations of Contraction

1. For each $C_i = \{\{u_1, v_1\}, \ldots\}$:
   (a) $R_i \leftarrow \emptyset$.
   (b) Pick u_1 to be the canonical vertex.
   (c) For each $\{x, y\} \in C_i$, add (u_1, x) and (u_1, y) to relabeling R_i.
2. Apply relabeling $\bigcup_i R_i$ to I, yielding the contracted edge list I'.

LEMMA 3.5. *Contracting an edge list I by a list of delineated subcomponents $C = \{C_1, C_2, \ldots\}$ can be performed in $O(sort(|I|) + sort(\sum_i |C_i|))$ I/Os in the FIO model.*

Functional Implementations of Deletion

Given edge lists *I* and *D*:

1. sort *I* and *D*
2. process in tandem to delete

Given edge list *I* and vertex list *U*:

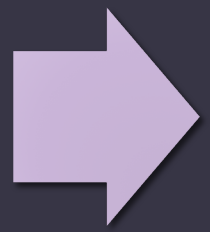
1. sort *U*
2. sort *I* by 1st component and process in tandem to delete
3. sort *I* by 2nd component and process in tandem to delete

LEMMA 3.6. *Deleting a vertex or edge set of cardinality N from an edge set I can be performed in $O(sort(|I|) + sort(N))$ I/Os in the FIO model.*

Deterministic Algorithms

Functional Implementations of

Selection, Relabeling, Contraction, Deletion



Functional Forms of Deterministic

CC, MSF, BMSF, MM

Functional Connected Components

Algorithm CC

1. Let E_1 be any half of the edges of G; let $G_1 = (V, E_1)$.
2. Compute $CC(G_1)$ recursively.
3. Let $G' = G/CC(G_1)$.
4. Compute $CC(G')$ recursively.
5. $CC(G) = CC(G') \cup RL(CC(G'), CC(G_1))$.

LEMMA 4.1. *Algorithm* **CC** *runs in* $O(sort(E)\log_2(E/M))$ *I/Os in the FIO model.*

Functional Connected Components

THEOREM 4.2. *The delineated edge list of components of a graph can be computed in* $O(sort(E) + (E/V)sort(V)\log_2(V/M))$ *I/Os in the FIO model.*

- Suffices to compute forest *F* of rooted stars
- Sort *E* by 1st vertex, sort *F* by source vertex, process in tandem, assign component labels to edges
- If *E* < *V*, we are done by Lemma 4.1 (prev slide)
- Otherwise,
  - Partition *E* into ceil(*E/V)* lists
  - Compute forest of rooted stars for each sublist
  - Iteratively merge pairs of forests (replace pairs of forests with their union)
  - Runtime is same as Lemma 4.1

Functional Minimum Spanning Forest

Algorithm MSF

1. Let E_1 be any lowest-cost half of the edges of G; i.e., every edge in $E \setminus E_1$ has weight at least that of the edge of greatest weight in E_1. Let $G_1 = (V, E_1)$.
2. Compute $MSF(G_1)$ recursively.
3. Let $G' = G/MSF(G_1)$.
4. Compute $CC(G')$ recursively.
5. $MSF(G) = EX(MSF(G')) \cup MSF(G_1)$.

Note: MSF is a BMSF!

THEOREM 4.3. *An MSF of a graph can be computed in* $O(sort(E)+(E/V)sort(V)\log_2(V/M))$ *I/Os in the FIO model.*

Functional Maximal Matching

Algorithm **MM**

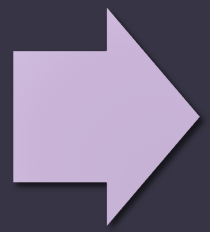
1. Let E_1 be any non-empty, proper subset of edges of G; let $G_1 = (V, E_1)$.
2. Compute $MM(G_1)$ recursively.
3. Let $E' = E \setminus V(MM(G_1))$; let $G' = (V, E')$.
4. Compute $MM(G')$ recursively.
5. $MM(G) = MM(G') \cup MM(G_1)$.

THEOREM 4.4. *A maximal matching of a graph can be computed in* $O((E/V)sort(V) \log_2(V/M))$ *I/Os in the FIO model.*

Randomized Algorithms

Functional Implementations of

Selection, Relabeling, Contraction, Deletion, Boruvka Step



Functional Forms of Randomized

CC, MSF, BMSF, MM

Randomized Algorithms: Boruvka Step

Boruvka step: selects and contracts the edge of minimum weight incident on each vertex.

=> Halves number of vertices in graph

=> Preserves MSF of contracted graph

Let *G* be a graph, let *F* be subgraph of *G* after a Boruvka step, let *G'* be resulting graph

=> MSF of *G* is the MSF of *G'* plus edges in *F*

Randomized Algorithms: Boruvka Step

LEMMA 5.1. *If $B = O(N/\log^{(i)} N)$ for some fixed integer $i > 0$, then a Borůvka step can be performed in $O(sort(E))$ I/Os in the FIO model.*

Relies on results from Chiang et al. [13]
Note: Arge [4] + Kumar and Schwabe [26] produce similar but not functional results

Sort by 1st and 2nd components of each edge and scan to select minimum weight edge incident on each vertex

O(sort(E)) I/Os

Let *F* be the set of chosen edges => *F* is a forest

Find connected components in *O(sort(V)) I/Os* [13]

Label each edge in *F* by connected component to rearrange edges into list of delineated components in *O(sort(E))* I/Os

Contract *E* by *F* in *O(sort(E))* I/Os by Lemma 3.5

Randomized Linear MSF Algorithm

Karger et al [26]

1. Perform two Borůvka steps, which reduces the number of vertices by at least a factor of four. Call the contracted graph G'.
2. Choose a subgraph H of G' by selecting each edge independently with probability $1/2$.
3. Apply the algorithm recursively to find the MSF F of H.
4. Delete from G' each edge $\{u, v\}$ such that (1) there is a path, $P(u, v)$, from u to v in F and (2) the weight of $\{u, v\}$ exceeds that of the maximum-weight edge on $P(u, v)$. Call the resulting graph G''.
5. Apply the algorithm recursively to G'', yielding MSF F'.
6. Return the edges contracted in step 1 together with those in F'.

THEOREM 5.2. *If $B = O(N/\log^{(i)} N)$ for some fixed integer $i > 0$, then an MSF of a graph can be computed in $O(sort(E))$ I/Os with probability $1 - e^{-\Omega(E)}$ in the FIO model.*

Randomized Connected Components

COROLLARY 5.3. *If* $B = O(N/\log^{(i)} N)$ *for some fixed integer* $i > 0$, *then the delineated edge list of components of a graph can be computed in* $O(sort(E))$ *I/Os with probability* $1 - e^{-\Omega(E)}$ *in the FIO model.*

PROOF. The MSF of the graph yields its connected components. □

Randomized BMSF

COROLLARY 5.4. *If* $B = O(N/\log^{(i)} N)$ *for some fixed integer* $i > 0$, *then a BMSF of a graph can be computed in* $O(sort(E))$ *I/Os with probability* $1 - e^{-\Omega(E)}$ *in the FIO model.*

PROOF. Any MSF is also a BMSF, so we can apply Theorem 5.2. □

Randomized MM

Luby [28]

1. $I \leftarrow \emptyset$; $V' \leftarrow V$; $E' \leftarrow E$.
2. Set $d(v)$ to the degree of vertex v, $\forall v \in V'$.
3. Mark v with probability $1/2d(v)$ if $d(v) > 0$ and probability 1 if $d(v) = 0$, $\forall v \in V'$.
4. For each $\{u, v\} \in E'$ such that both u and v are marked, if $d(u) \leq d(v)$, then unmark u; otherwise unmark v.
5. $I' \leftarrow \{v \in V' : v \text{ is marked}\}$; $I \leftarrow I \cup I'$; $Y \leftarrow I' \cup \{v \in V' : \exists \{u, v\} \in E' \ni u \in I'\}$.
6. $V' \leftarrow V' \setminus Y$; $E' \leftarrow E' \setminus Y$.
7. If $V' \neq \emptyset$, repeat from step 2.

THEOREM 5.5. *For any fixed ε, a maximal independent set of a graph can be computed in $O(sort(E))$ I/Os with probability at least $1 - \varepsilon$ in the FIO model.*

Semi-External Algorithms

When $V <= M$ but $E > M$

CC: Can compute forest of rooted stars in one scan

MSF: Sort edge list by weight, then maintain forest internally.

- Can also use dynamic trees to maintain internal forest. For each edge, delete maximum weight edge on any cycle created. => *O(E log V)* internal computation

BMSF: Same applies for BMSF

MM: Maintain internal matching and can compute in one scan of edge-list

Concerns: 1) Sorting, 2) Grouping by Key

External Sorting

Partition items into M/B buckets, each with distinct range. Each block in memory holds a single bucket

Assign items to appropriate block. If block is full, write to disk

Sort buckets recursively

Chain disk blocks into a linked list without affecting the asymptotic space usage

After reading input:

- Non-empty memory blocks with non-null pointers are emptied
- Remaining memory blocks partitioned into contiguous ranges of keys

Final output produced by scanning through the chains on disk, in order, to produce bucketed list that only requires N/B disk blocks.

THEOREM 6.1. *Sorting N records with keys in the integral range $[1, K]$ takes $\Theta(scan(N) \log_{M/B} K)$ I/Os in the FIO model.*

External Grouping

Given N records have K distinct keys, use the previous scheme to group them

During each recursive phase, pick hash function, h, independently and uniformly at random from a family of universal hash function that map the K keys to the range [1, floor(M/B)]

Use h to assign keys to buckets

When bucket contains M/B distinct keys, group records in one additional scan

THEOREM 6.3. *With probability at least* $1 - 1/K^c$ *for any fixed constant* c, $O(scan(N) \log_{M/B} K)$ *I/Os suffice to group* N *items with* K *distinct keys from an arbitrary universe in the FIO model.*

My Thoughts

Strengths:

- comprehensive theoretical analysis about complexity

- builds off of a lot of previous work in the topic

- justification about the benefits of a functional approach

Weaknesses:

- no experiments, only theory

- unable to address other fundamental graph algorithms like BFS, DFS, topological sorting, single source shortest paths, transitive closure

Discussion

How feasible do you think it is to implement the proposed functional approaches?

What are your thoughts on the novelty of the authors' functionalization of other algorithms, e.g. Karger et al's randomized linear MSF and Luby's randomized MM algorithms?

How do you think FIO compares to other approaches? (PRAM simulation and buffering data structures)