Theoretical Models of MapReduce-type Computation

张晓达

Paper Presentations

- Howard Karloff, Siddharth Suri, Sergei Vassilvitskii. **A Model of Computation for MapReduce. *SODA '10*.**
- Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. **Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). *SPAA '16*.** [Best Paper Award]
- Sungjin Im, Benjamin Moseley, and Xiaorui Sun. **Efficient Massively Parallel Methods for Dynamic Programming. *STOC '17*.**

Outline

- Background
  - Traditional PRAM and **NC**
  - MapReduce
- The fundamental problems introduced by MapReduce
- The model and a particular problem [KSV *SODA '10*]
- The model and its ***lower bound*** [RVW *SPAA '16*]
- The model and a specific ***algorithm design technique*** [IMS *STOC '17*]

Background

- The synchronous parallel random assess machine (**PRAM**) model
  - Global memory (M locations), P processors (each has a constant number of local registers)
  - Computation proceeds in a series of *synchronous* parallel steps, each consisting:
    - each processor chooses a mem. location and reads;
    - each does some computation;
    - each chooses a mem. location and writes;
  - By ***synchrony***, every processor finishes executing step *i* before any processor begins executing step *i+1*
- Whether or not allowing concurrent read/write
  - EREW, CREW, CRCW

Background

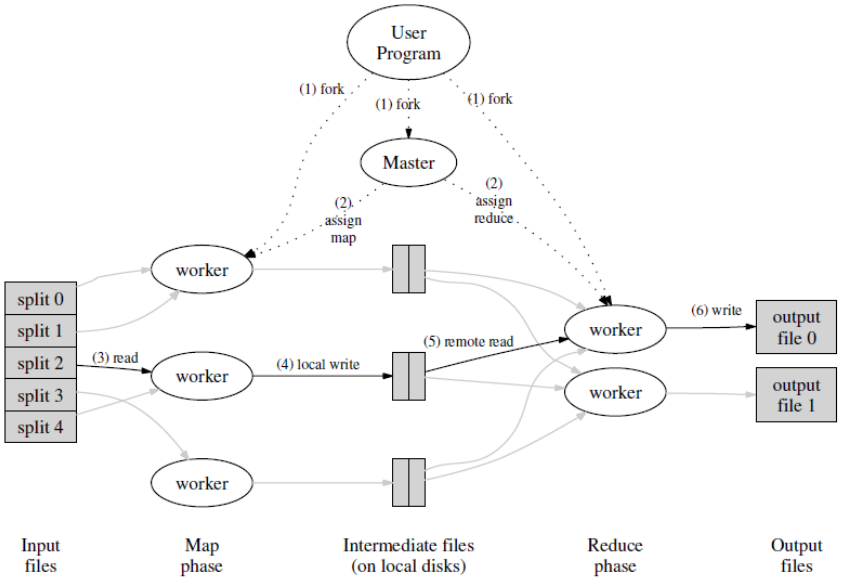
- What problems can be solved in ***polylog*** time on a PRAM with a ***polynomial*** number of processors?
  - Polylog time serves as *gold-standard* for parallel running time;
  - Polynomial number of processors provides a necessary condition for *efficiency*.
- The class **NC** (Nick's class) is defined as the set of such problems

► **Definition 12.1:** The class *NC* consists of languages L that have a PRAM algorithm A such that for any $x \in \Sigma^*$

- $x \in L \Rightarrow A(x)$ accepts;
- $x \notin L \Rightarrow A(x)$ rejects;
- the number of processors used by A on x is polynomial in $|x|$;
- the number of steps used by A on x is polylogarithmic in $|x|$.

Background

- MapReduce
  - Map phase, (shuffle), Reduce phase
  - Tasks in Map phase compute over disjoint input splits and output intermediate data *locally*
  - Tasks in Reduce phase take as input the intermediate data, and contribute the parts of the final output



The fundamental problems

- What is *efficiently* (*e.g.* in polylog rounds) computable in the MapReduce paradigm?

- What is the fundamental *limitations* on how efficiently algorithms implemented on MapReduce?

- Is there a *meta-algorithm* that task as input a sequential algorithm and outputs an efficient distributed (MapReduce) algorithm?

The **MRC**

The input to a **MRC** problem is a finite sequence of pairs <kj, vj>, for j = 1, 2, 3 …. The length of the input is $n = \sum_j (|k_j| + |v_j|)$.

randomized version

DEFINITION *Fix an* $\epsilon > 0$. *An algorithm in* $\mathcal{MRC}^i$ *consists of a sequence* $\langle \mu_1, \rho_1, \mu_2, \rho_2, \ldots, \mu_R, \rho_R \rangle$ *of operations which outputs the correct answer with probability at least* $3/4$ *where:*

- *Each* μ_r *is a randomized mapper implemented by a RAM with* $O(\log n)$*-length words, that uses* $O(n^{1-\epsilon})$ *space and time polynomial in* n.
- *Each* ρ_r *is a randomized reducer implemented by a RAM with* $O(\log n)$*-length words, that uses* $O(n^{1-\epsilon})$ *space and time polynomial in* n.
- *The total space* $\sum_{\langle k;v \rangle \in U'_r} (|k| + |v|)$ *used by* $\langle key; value \rangle$ *pairs output by* μ_r *is* $O(n^{2-2\epsilon})$.
- *The number of rounds* $R = O(\log^i n)$.

number of machines is sublinear $O(n^{1-\epsilon})$

sublinear space in each machine

允许mapper阶段的输出有duplication, 但不是任意duplication

Restriction is only applied to mapper, not reducer.

- (deterministic **MRC**) **DMRC**'s relationship with **P/NC**
  - **DMRC** is in **P**

    THEOREM $\quad If\ \mathcal{P} \neq \mathcal{NC}\ then\ \mathcal{DMRC} \not\subseteq \mathcal{NC}.$

    PROOF: From a **P-complete** language (Circuit Value), construct another language which is solvable in **DMRC**. Since **P-complete** language is not in **NC**, **DMRC** is not in **NC**.
  - **DMRC** ?= **P**

    DMRC实际定义的language是solvable by Turing machine simultaneously in space O($n^2 \log n$) and polynomial time。

    若证明**P ≠ MRC**，需找到一个language在**P - MRC**中，这样的language目前还不知道是否存在。

MRC-parallelizable function

- **MRC**-parallelizable function as a building block:

applying *g*, and then *h*

DEFINITION *Let S be a set. Call a function f on S $\mathcal{MRC}$-parallelizable if there are functions g and h so that:*

1. *For any partition $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of S, where $\cup_i T_i = S$ and $T_i \cap T_j = \emptyset$ for $i \neq j$ (of course), f can be expressed as: $f(S) = h(g(T_1), g(T_2), \ldots, g(T_k))$.*
2. *g and h can be expressed in $O(\log n)$ bits.*
3. *g and h can be computed in time polynomial in $|S|$ and every output of g can be expressed in $O(\log n)$ bits.*

  - If one wants to compute *f* over S, one could do so by partitioning S arbitrarily, applying *g* to each part of the partition, and then applying *h* to the results.

The power of **MRC**-parallelizable function

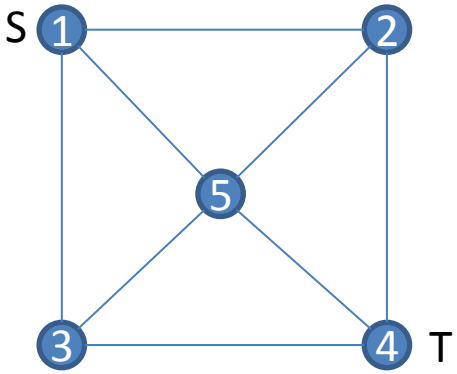
LEMMA *Consider a universe $\mathcal{U}$ of size n and a collection $\mathcal{S} = \{S_1, \ldots, S_k\}$ of subsets of $\mathcal{U}$, where $S_i \subseteq \mathcal{U}$, $\sum_{i=1}^{k} |S_i| \leq n^{2-2\epsilon}$, and $k \leq n^{2-3\epsilon}$. Let $\mathcal{F} = \{f_1, \ldots, f_k\}$ be a collection of $\mathcal{MRC}$-parallelizable functions. Then the output $f_1(S_1), \ldots, f_k(S_k)$ can be computed using $O(n^{1-\epsilon})$ reducers each with $O(n^{1-\epsilon})$ space.*

- 构造性证明: (两轮MapReduce)需要split 每个Si，让g先对每个partition进行计算，而后将用h再进行一次计算可得最后结果。
- The power of this lemma: focus on the structure of the problem and the input; the lemma will handle *how to distribute* the input across the reducers

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

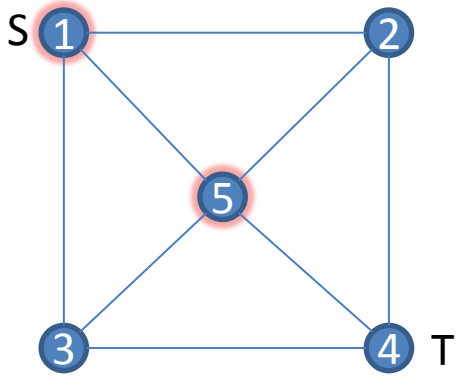


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

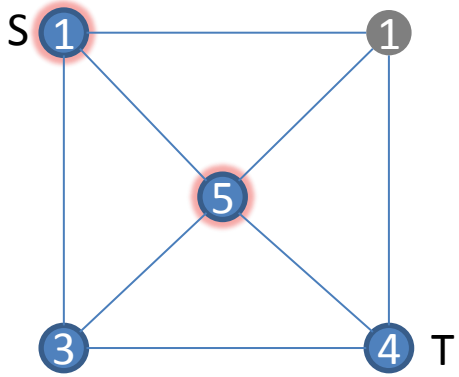


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

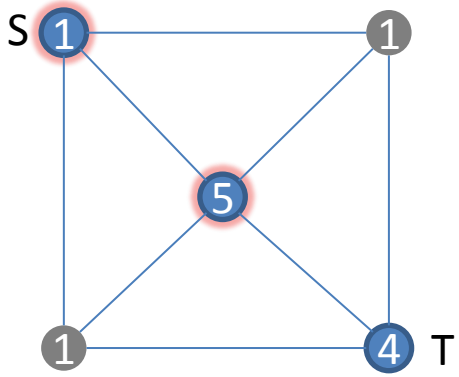


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

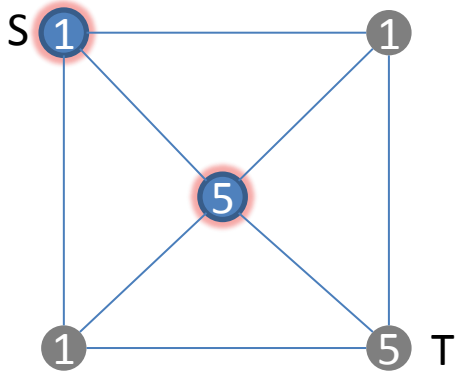


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

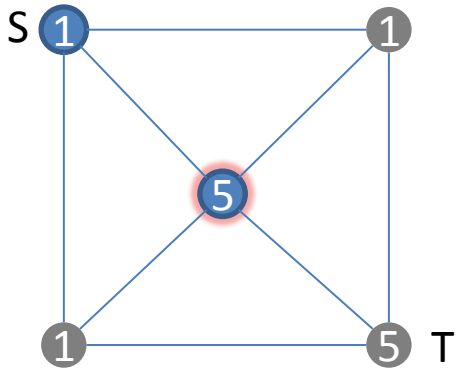


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.

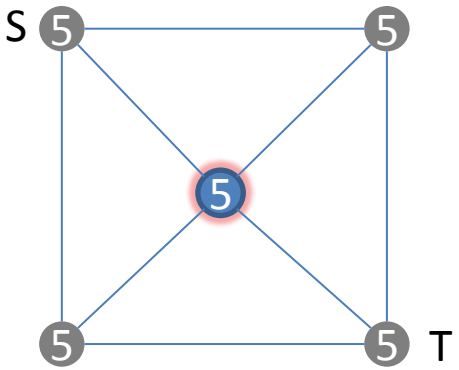


- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

Application of the Function lemma

- To solve the USTCON problem:

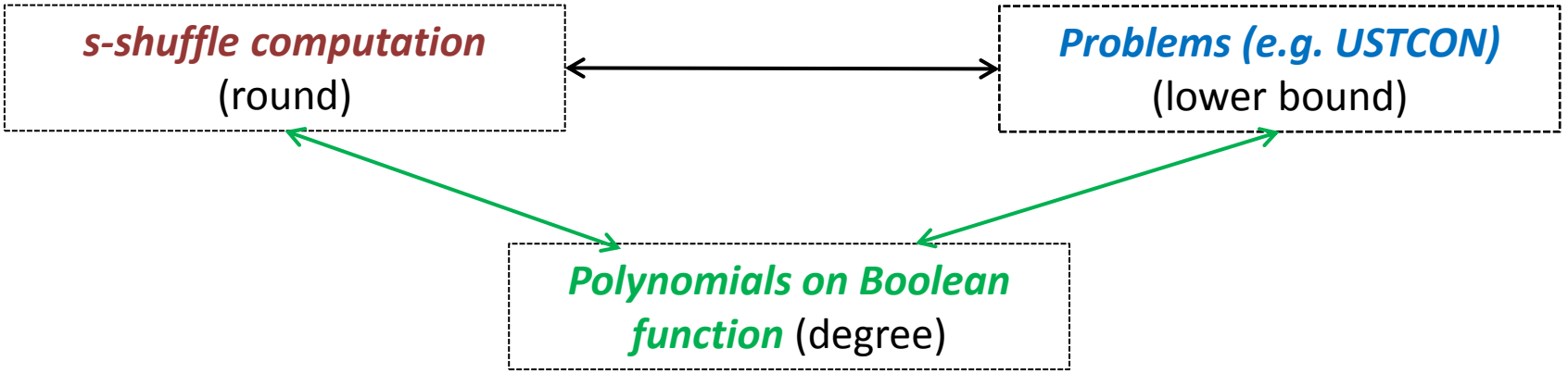
1. Begin with every node $v \in V$ being active with label $\ell(v) = v$.
2. For $i = 1, 2, 3, ..., O(\log N)$ do:
   (a) Call each active node a leader with probability $1/2$.
   (b) For every active non-leader node w, find the smallest (according to π) node $w^* \in \Gamma'(w)$.
   (c) If w^* is not empty, mark w passive and relabel each node with label w by w^*.
3. Output *true* if s and t have the same labels, false otherwise.



- This Monte Carlo algorithm is correct
- It needs O(log N) rounds to terminate, w.h.p.
- The (a)(b)(c) step *satisfies* the conditions of the Function Lemma

The s-shuffle model and lower bound

- **s-shuffle model**
  - few rounds s-shuffle computation and its connection to low-degree polynomials
    - translate a lower bound on the *polynomial degree* of a Boolean function to a lower bound of *s-shuffle computation*
  - prove lower bounds on the polynomial degree (Boolean)



The s-shuffle model

Capturing core properties of massively parallel computation

- Computation proceeds in ***synchronous*** rounds
- In each round, each machine performs an arbitrary computation on its input, and send arbitrary information to arbitrary machines in the next round (***shuffle***)
- The only constraint is that each machine receives ***at most s bits*** each round

Definition (**s-SHUFFLE Computation**) An *R-round* s-SHUFFLE *computation with inputs* $x_1, \ldots, x_n$ *and outputs* $y_1, \ldots, y_k$ has the following ingredients:

1. A set V of *machines*, which includes one machine for each input bit x_i and each output bit y_i.
2. An assignment of a *round* $r(v)$ to each machine $v \in V$. Machines corresponding to input bits have round 0. Machines corresponding to output bits have round $R + 1$. All other machines have a round in $\{1, 2, \ldots, R\}$.
3. For each pair (u, v) of machines with $r(u) < r(v)$, a function α_{uv} from $\{0, 1, \perp\}^s$ to $\{0, 1, \perp\}^s$.

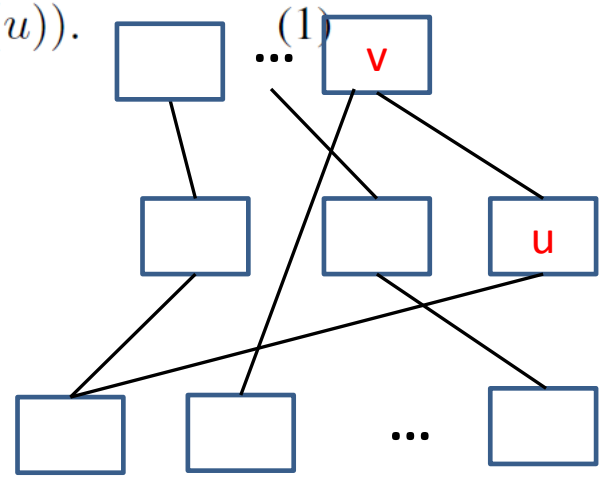
The s-shuffle model

Definition (Result of an s-SHUFFLE Computation) The *result* of an s-SHUFFLE computation assigns a value $g(v) \in \{0, 1, \perp\}^s$ to every machine $v \in V$, and is defined inductively as follows.

1. For a round-0 machine v, corresponding to an input bit x_i, the value $g(v)$ is the s-tuple $(x_i, \perp, \perp, \ldots, \perp)$.

2. Given the value $g(u)$ assigned to every machine u with $r(u) < q$, the value assigned to a machine v with $r(v) = q$ is the $\perp$-sum, over all machines u with $r(u) < r(v)$, of the message $\alpha_{uv}(g(u))$ sent to v by u:

$$g(v) := \odot_{u\,:\,r(u)<r(v)} \alpha_{uv}(g(u)). \qquad (1)$$



The main theorem

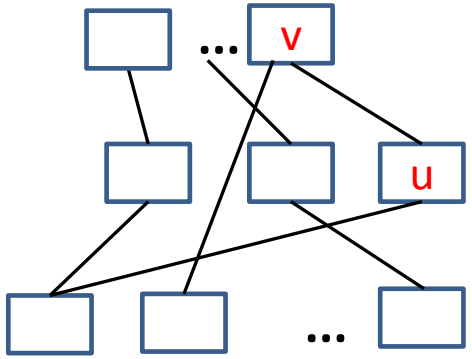
Theorem 3.1 *Suppose that an s-SHUFFLE computation computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then there are k polynomials $\{p_i(x_1, \ldots, x_n)\}_{i=1}^k$ of degree at most s^r such that $p_i(\mathbf{x}) = f(\mathbf{x})_i$ for all $i \in \{1, 2, \ldots, k\}$ and $\mathbf{x} \in \{0,1\}^n$.*

PROOF: Induction on the number of rounds.

We claim that for every non-output machine $v \in V$, value $\mathbf{z} \in \{0,1,\perp\}^s$, there is a polynomial $p_{v,z}(x_1, \ldots, x_n)$ that evaluates to 1 on point **x** for which the computation's assigned value *g(v)* to *v* is z and to 0 on all other points **x** $\in \{0,1,\perp\}^n$. $p_{v,z}$ has degree at most $s^{r(v)}$.

Base case: for a machine *v* in round 0 (*r(v)* = 0). Each such machine corresponds to an input bit x_i and its value *g(v)* is (x_i, ⊥, ⊥,…, ⊥). The (degree-1) polynomial for **z** = (0, ⊥, …, ⊥) and **z** = (1, ⊥, …, ⊥) are $p_{v,z}(x_1, \ldots, x_n)$ = 1- x_i and $p_{v,z}(x_1, \ldots, x_n) = x_i$, respectively. All other values of **z** are impossible for such a machine, so they have polynomials $p_{v,z}(x_1, \ldots, x_n)$ = 0.

The main theorem



Theorem 3.1 *Suppose that an s-SHUFFLE computation computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then there are k polynomials $\{p_i(x_1, \ldots, x_n)\}_{i=1}^k$ of degree at most s^r such that $p_i(\mathbf{x}) = f(\mathbf{x})_i$ for all $i \in \{1, 2, \ldots, k\}$ and $\mathbf{x} \in \{0,1\}^n$.*

Consider a machine *v* (neither an input bit nor an output bit).

Fix some potential value $\mathbf{z} \in \{0,1,\perp\}^s$ of *g(v),* and focus first on the *ith* coordinate of g(v). Consider some machine *u* of previous round. It sends z_i on port *i* to machine *v* when $\alpha_{\{uv\}}(g(u))$ has an ith entry of z_i.

By inductive hypothesis, for particular value $g(u)$, there is a polynomial of degree at most $s^{r(v)-1}$ that indicates the inputs **x** for which *u* receives this value.

To obtain a polynomial that represents the inputs x for which g(v) = **z**, just take the product, *g(v)1 = **z**1, g(v)2 = **z**2, …, g(v)s= **z**s*. This produces a polynomial of degree $s^{r(v)}$.

The main theorem

Theorem *Suppose that an s-SHUFFLE computation computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then there are k polynomials $\{p_i(x_1, \ldots, x_n)\}_{i=1}^k$ of degree at most s^r such that $p_i(\mathbf{x}) = f(\mathbf{x})_i$ for all $i \in \{1, 2, \ldots, k\}$ and $\mathbf{x} \in \{0,1\}^n$.*

逆否命题

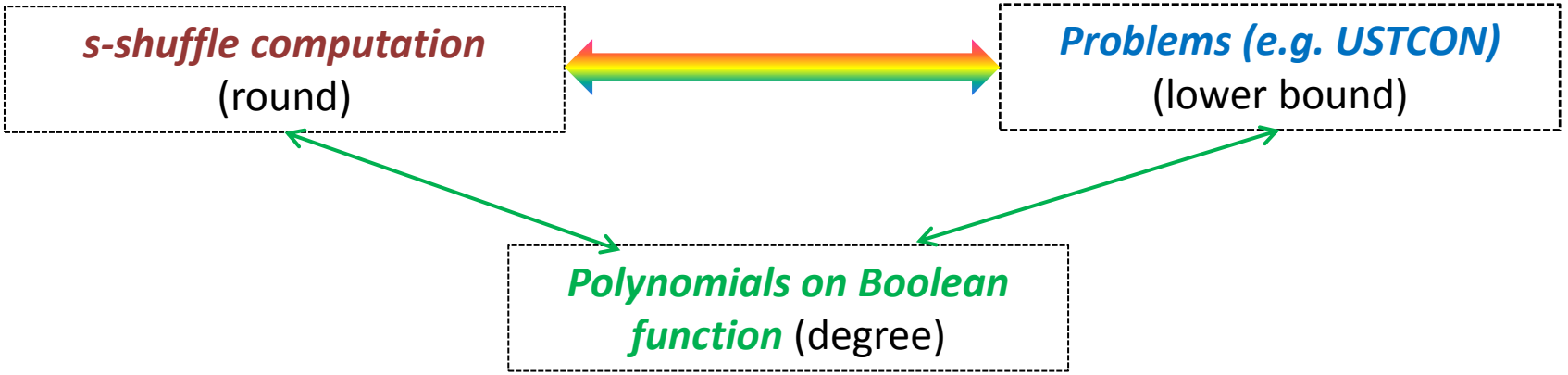
Corollary *If some output bit of the function $f : \{0,1\}^n \to \{0,1\}^k$ cannot be represented by a polynomial with degree less than d, then every s-SHUFFLE computation that computes f uses at least $\lceil \log_s d \rceil$ rounds.*

s-shuffle computation (round)

Polynomials on Boolean function (degree)

The lower bound

Theorem *The degree of the Boolean function for undirected* ST-CONNECTIVITY *on n vertices is $\binom{n}{2}$.*



The model and an algorithm design technique (DP)

- The model
  - Let *n* be the input size, *m* be the number of machines.
  - *local* memory on each machine should be $\tilde{\Theta}(n/m)$, and the algorithm should allow for $n^{\epsilon} \leq m \leq n^{1-\epsilon}$, for some constant $\epsilon > 0$

Tilde hides polylogarithmic factors

- *Are there principled guidelines for converting a sequential DP into an efficient (e.g. O(1) rounds) distributed DP?*

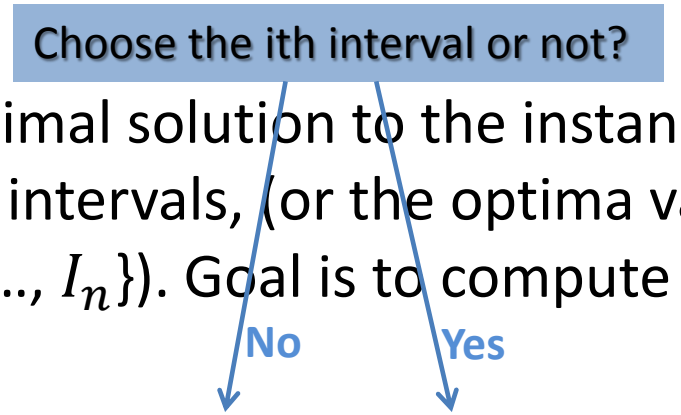
Key Properties

(1) **Monotonicity**: A sub-problem has no greater (smaller, resp.) optimum if the objective is to be maximized (minimized, resp.). (2) **Decomposibility**: The input can be decomposed into a two-level laminar family of partial input, where an upper level partial input is called a group and a lower level partial input is called a block, such that:

- A nearly optimal solution (for the whole input) can be constructed by concatenating solutions for groups.
- A nearly optimal solution for each group can be constructed from $O(1)$ blocks.

Weighted Interval Selection (WIS)

- Definition
  - **Input**: a collection of intervals $\{I_i = (a_i, b_i)\}_{i \in [n]}$, with their respective weights $\{w_i\}_{i \in [n]}$
  - **Goal**: choose a subset of disjoint intervals with the maximum weight.
- Standard sequential DP
  - Assume that intervals are ordered in increasing order of their start points
  - Let ***OPT(S)*** denote the optimal solution to the instance consisting of a subset *S* of intervals, (or the optima value). Let ***A***(i) denote ***OPT***($\{I_i, I_{i+1}, \ldots, I_n\}$). Goal is to compute ***A***(1) using the following recursion:



$$A(i) = \max\{A(i+1), w_i + A(j)\}$$

where $j = \arg\min_{j'} (b_i < a_{j'})$ when $i \leq n$; and $A(n+1) = 0$.

Preliminaries

- Goal

> THEOREM 4.1. *For any* $\epsilon, \delta > 0$, *there exists a* $(1-\epsilon)$-*approximation for the Weighted Interval Selection problem running in* $O(\frac{1}{\delta}(\log\frac{1}{\epsilon} + \log\frac{1}{\delta}))$ *rounds when each machine has memory* $\tilde{O}(n^{\delta})$.

- Preprocessing
  - 因为最后的结果只取决于interval开始和结束的相对位置，所以可以调整intervals, 使其starting and ending time are all distinct and not integers.
  - 将输入均分到m machines上. The first *n/m* intervals go to machine 1, the next go to machine 2, and so on. Assume that machine *k* can only see intervals starting at time between *k* and *k+1*.

The beginning lemma

- Definition
  - block: a subset of disjoint intervals
  - pairwise independent blocks: span disjoint sets of machines
  - *L*-block: a block contains at most *L* crossing intervals

> LEMMA 4.2. *For any even integer $L \geq 2$, there exists a $(1 - 2/L)$-approximate solution consisting of (pair-wise) independent L-blocks.*

PROOF: Consider a fixed optimal solution. Partition the crossing intervals into ***groups*** so that each group contains *L* consecutive crossing intervals; (the last may contain less).
We remove the lightest interval from every group except the last. Clearly, we lose at most *1/L* times the optimum. It is easy to see that the resulting solution consists of *2L*-blocks that are pairwise independent.

Overview of the algorithm

- *如果*a nearly optimal solution consisting only of 0-blocks *就好了*
  - *那么只需要*each machine compute the **best 'local'** solution from the local intervals on the machine, and simply **aggregate** the weights of the local solutions from all machines.
  - The aggregation can be done in $O(1/\delta)$, machine has $\tilde{O}(n^{\delta})$ memory
  - **However**, the optimal solution may have lots of heavy crossing intervals, which can not be ignored.
- By the above lemma
  - We can construct a $(1-\epsilon)$-approximate solution from $2/\epsilon$-blocks

  that are pairwise independent.

  We compute such blocks approximately and memory-efficiently in O(1) rounds. 一旦我们有了这些blocks, we view each block as an ***consolidated*** **interval**. 这样, the new instance whose size is almost linear in the number of machines. 至多只需递归$O(1/\delta)$ 次

Formally state the idea

Definition 4.3. For each interval I_i and a target weight μ, let $\mathrm{OPT}(i, \mu, L)$ be the smallest interval index $j \geq i$ such that there is a L-block of weight at least μ that is a subset of $\{I_i, I_{i+1}, I_{i+2}, \cdots I_n\}$ and is disjoint from $\{I_j, I_{j+1}, \cdots, I_n\}$. Let $W := \{0, \frac{1}{n^2}, \frac{(1+\eta)}{n^2}, \frac{(1+\eta)^2}{n^2}, \cdots, n\}$ for η to be determined. A family $\mathcal{F} = \{D(i, \mu, L)\}_{i \in [n], \mu \in W}$ of blocks is said to be a $1 - \gamma$-approximate compact family of L-blocks if $D(i, \mu, L)$ has weight at least $(1 - \gamma)\mu$ and $D(i, \mu, L) \leq \mathrm{OPT}(i, \mu, L)$. Here, $D(i, \mu, L)$ is analogously defined as $\mathrm{OPT}(i, \mu, L)$.

- Idea
  - there is a (1-2/L)-approximate solution consisting only of L-blocks. 对于each of those blocks, D*, t**he compact family** contains 几乎和D*一样好的D, 其中**D的weight is within (1-γ) of D*'s weight and ends no later than D***. 因此，如果我们将nearly optimal solution中的block换成相应的the compact family中的block。 这样得到的solution是可行解且近似度是(1-2/L)(1-γ).

Formally state the idea

LEMMA 4.4. *If we can construct a* $(1-O(\epsilon\delta))$*-approximate compact family of* L*-blocks in* $O(\log\frac{1}{\epsilon}+\log\frac{1}{\delta})$ *rounds where* $L=\frac{2}{\epsilon\delta}$ *and* $\eta=\frac{\epsilon\delta}{10(\log(1/\epsilon)+\log(1/\delta))}$*, then we can obtain a* $(1-O(\epsilon))$*-approximate solution for the IS problem in* $O(\frac{1}{\delta}\cdot(\log\frac{1}{\epsilon}+\log\frac{1}{\delta}))$ *rounds.*

- It now remains to show how to construct the Desired Compact Family using DP.

find a desired compact family, $\mathcal{F}(\ell) := \{D(i,\mu,2^\ell-1)\}_{i\in[n],\mu\in W}$, as stated in Lemma 4.4. Towards this end, we find such a family for $\ell = 0,1,2,\cdots,\ell_1 = \lceil\log 2/(\epsilon\delta)\rceil$ in this order until we have $2^\ell - 1 \geq L = 2/(\epsilon\delta)$. Since we will use an induction on the value of ℓ, we refine and strengthen the property that $\mathcal{F}(\ell)$ satisfies: our goal is to find $\mathcal{F}(\ell)$ that is $(1-\eta)^\ell$-approximate in increasing order of $\ell = 0,1,2,\cdots\ell_1$ in $O(\ell)$ rounds. Since $L = \frac{2}{\epsilon\delta}$, we only need $O(\log 1/\epsilon + \log 1/\delta)$ rounds, and we have $(1-\eta)^{\ell_1} = 1 - O(\epsilon\delta)$ since $\eta = \frac{\epsilon\delta}{10(\log(1/\epsilon)+\log(1/\delta))}$.

Our remaining goal is to find $\mathcal{F}(\ell)$ that is $(1-\eta)^{\ell}$-approximate in increasing order of $\ell = 0, 1, 2, 3, \cdots \ell_1$ in $O(\ell)$ rounds.

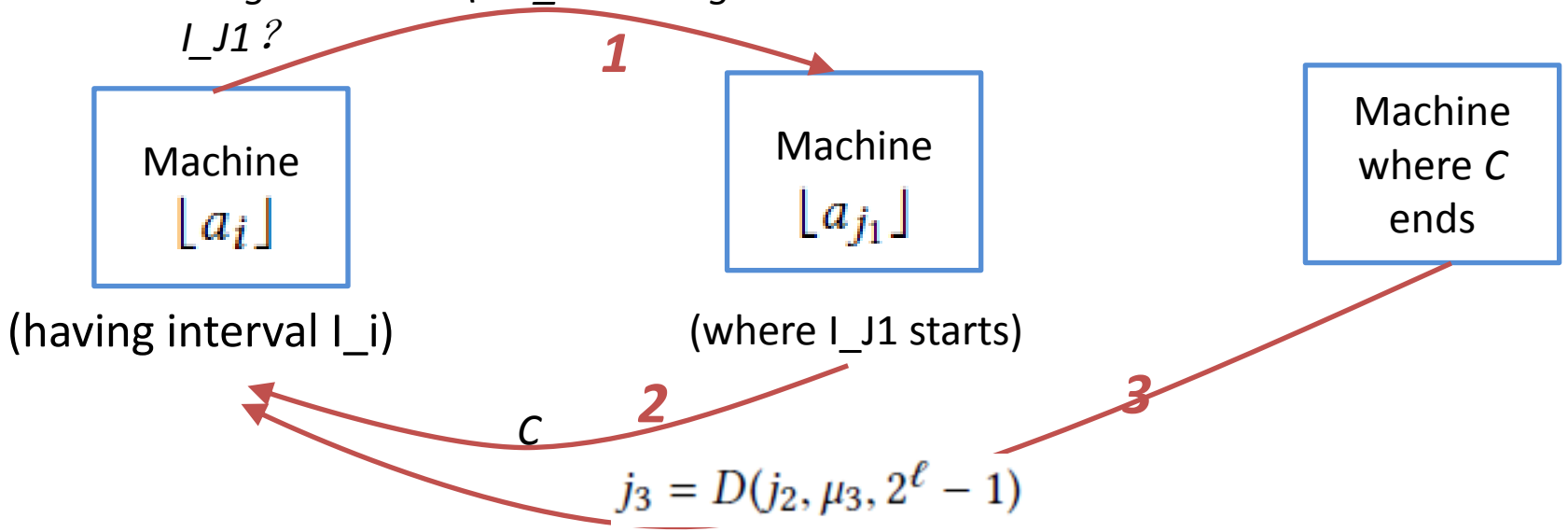
find *F(l+1)* from *F(l)* in O(1) rounds.

Show how to compute $D(i, \mu, 2^{\ell+1} - 1)$

Consider any triplet $\mu_1, \mu_2, \mu_3 \in W$ such that

$$(1-\eta)\mu \leq \mu_1 + \mu_2 + \mu_3 \leq \mu \qquad j_1 = D(i, \mu_1, 2^{\ell} - 1)$$

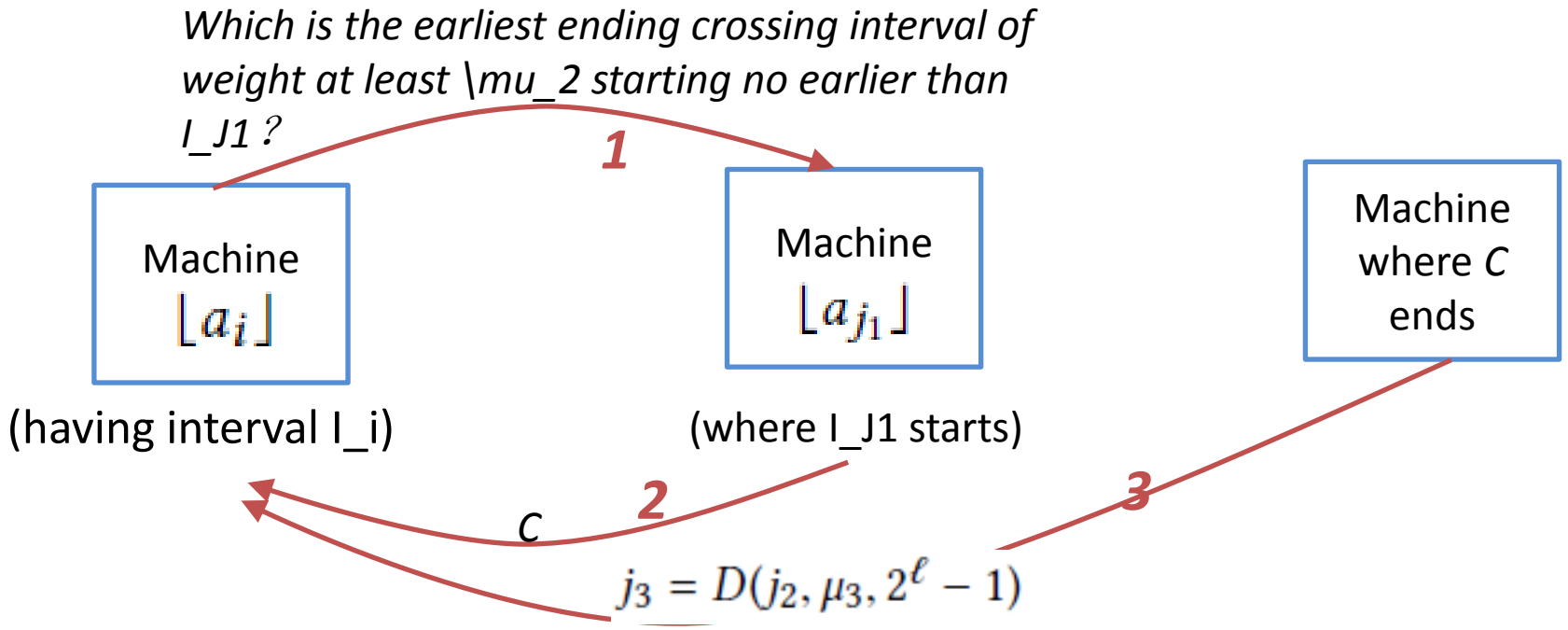
Which is the earliest ending crossing interval of weight at least \mu_2 starting no earlier than I_J1 ?



I_{j_2} is the earliest starting interval after the interval C ends.

Consider any triplet $\mu_1, \mu_2, \mu_3 \in W$ such that

$$(1-\eta)\mu \le \mu_1 + \mu_2 + \mu_3 \le \mu \qquad j_1 = D(i, \mu_1, 2^\ell - 1)$$



I_{j_2} is the earliest starting interval after the interval C ends.

2, 3 requires only O(1) rounds.

We take the minimum j_3 over all possible triplets in parallel.

There are at most

$|W|^3 = O(\log^3_{1+\eta} n)$ triplets to be considered for each i. Thus, each machine uses memory at most $(n/m)O(\log^3_{1+\eta} n) = \tilde{O}(n/m)$.

Prove the approximation radio by induction.