Randomization in Parallel and Distributed Computing

Paris Koutris

June 4, 2010

Outline

The Liar Game

Parallel Maximal Independent Set

Parallel Perfect Matching

Hot-Potato Routing

The Liar Game

- We have 2 players, Alice and Bob and 3 integers: N, Q, K
- Alice chooses a number m from $1, 2, \ldots, N$
- Bob must find m with Q questions of the form: *is* m *in set* S*?*
- Alice may lie at most K times
- Bob *wins* when there exists exactly one possible answer according to Alice's answers
- Either Bob or Alice has a perfect strategy!

Example

$N = 4$, $K = 1$, $Q = 5$, Alice chooses $m = 1$

Bob: Is the number in $\{1, 2\}$?

Alice: NO

Bob: Is the number in $\{1, 2\}$?

Alice: YES (Alice lies only once)

Bob: Is the number in $\{2, 3\}$?

Alice: NO (Bob knows that Alice tells the truth, and thus $1, 4$ are the only candidates)

Bob: Is the number in $\{4\}$?

Alice: NO (BINGO! Bob has won)

Example

$N = 4$, $K = 1$, $Q = 5$, Alice chooses $m = 1$

Bob: Is the number in $\{1, 2\}$?

Alice: NO

Bob: Is the number in $\{1, 2\}$?

Alice: YES (Alice lies only once)

Bob: Is the number in $\{2, 3\}$?

Alice: NO (Bob knows that Alice tells the truth, and thus $1, 4$ are the only candidates)

Bob: Is the number in $\{4\}$?

Alice: NO (BINGO! Bob has won)

Example

$N = 4$, $K = 1$, $Q = 5$, Alice chooses $m = 1$

Bob: Is the number in $\{1, 2\}$?

Alice: NO

Bob: Is the number in $\{1, 2\}$?

Alice: YES (Alice lies only once)

Bob: Is the number in $\{2, 3\}$?

Alice: NO (Bob knows that Alice tells the truth, and thus $1, 4$ are the only candidates)

Bob: Is the number in $\{4\}$?

Alice: NO (BINGO! Bob has won)

Example

$N = 4$, $K = 1$, $Q = 5$, Alice chooses $m = 1$

Bob: Is the number in $\{1, 2\}$?

Alice: NO

Bob: Is the number in $\{1, 2\}$?

Alice: YES (Alice lies only once)

Bob: Is the number in $\{2, 3\}$?

Alice: NO (Bob knows that Alice tells the truth, and thus $1, 4$ are the only candidates)

Bob: Is the number in $\{4\}$?

Alice: NO (BINGO! Bob has won)

Example

$N = 4$, $K = 1$, $Q = 5$, Alice chooses $m = 1$

Bob: Is the number in $\{1, 2\}$?

Alice: NO

Bob: Is the number in $\{1, 2\}$?

Alice: YES (Alice lies only once)

Bob: Is the number in $\{2, 3\}$?

Alice: NO (Bob knows that Alice tells the truth, and thus $1, 4$ are the only candidates)

Bob: Is the number in $\{4\}$?

Alice: NO (BINGO! Bob has won)

Strategies

Notice that if $K = 0$, the best Bob can do is a *binary search*. If we fix K, Q, what is the largest N such that Bob always has a winning strategy?

Theorem

If $2^Q < N\left(1 + Q + \ldots + \binom{Q}{K}\right)$, *then Alice always wins*

For $K = 1$, we get that $N > \frac{2^Q}{1+Q}$

Proof (1)

- Let Alice play the following *dummy* strategy: flip a coin to decide whether to lie or not
- If Alice lies more than K times, we declare Bob as the winner
- For $1 \leqslant i \leqslant N$ define the indicator variable

$$X_i = \begin{cases} 1 & \text{if } i \text{ is a candidate at the end,} \\ 0 & \text{otherwise.} \end{cases}$$

- Let $X = \sum_{i=1}^{N} X_i$. Bob wins $\Leftrightarrow X \leqslant 1$
- Fix i. For every question, Bob gets an indication about whether i is the number or not
- i is a *candidate* at the end only if there have been at most K "NO" answers

Proof (1)

- Let Alice play the following *dummy* strategy: flip a coin to decide whether to lie or not
- If Alice lies more than K times, we declare Bob as the winner
- For $1 \leqslant i \leqslant N$ define the indicator variable

$$X_i = \begin{cases} 1 & \text{if } i \text{ is a candidate at the end,} \\ 0 & \text{otherwise.} \end{cases}$$

- Let $X = \sum_{i=1}^{N} X_i$. Bob wins $\Leftrightarrow X \leqslant 1$
- Fix i. For every question, Bob gets an indication about whether i is the number or not
- i is a *candidate* at the end only if there have been at most K "NO" answers

Proof (2)

- What is the probability of that?

$$\mathbb{P}\mathrm{r}[\#\mathrm{NO} \leqslant K] = \sum_{i=1}^{K} \mathbb{P}\mathrm{r}[\#\mathrm{NO} = i] = \sum_{i=1}^{K} \binom{Q}{i} \frac{1}{2^Q}$$

- Linearity of Expectation: $\mathbb{E}[X] = N \cdot \sum_{i=1}^{K} \binom{Q}{i} \frac{1}{2^Q} > 1$
- $\mathbb{P}\mathrm{r}[\text{Bob wins}] = \mathbb{P}\mathrm{r}[X \leqslant 1] < 1$
- Thus, whatever strategy Bob plays, there exists a sequence of choices such that Alice wins $\Rightarrow$ Alice has a winning strategy!

Proof (2)

- What is the probability of that?

$$\mathbb{P}r[\#NO \leqslant K] = \sum_{i=1}^{K} \mathbb{P}r[\#NO = i] = \sum_{i=1}^{K} \binom{Q}{i} \frac{1}{2^Q}$$

- Linearity of Expectation: $\mathbb{E}[X] = N \cdot \sum_{i=1}^{K} \binom{Q}{i} \frac{1}{2^Q} > 1$
- $\mathbb{P}r[\text{Bob wins}] = \mathbb{P}r[X \leqslant 1] < 1$
- Thus, whatever strategy Bob plays, there exists a sequence of choices such that Alice wins $\Rightarrow$ Alice has a winning strategy!

But how Alice actually wins?

- We analyze only the case $K = 1$
- Let $S_{x,i}$ be the *ministrategy*: Alice chooses x and lies at question i (if i=0, Alice does not lie)
- Alice has $N \cdot (Q + 1) > 2^Q$ ministrategies
- After each question of Bob, some ministrategies are valid, other not
- STRATEGY: Alice chooses the answer which maximizes the number of *valid* ministrategies
- After each question, Alice has at least half ministrategies left!
- After Q questions, Alice has at least 2 ministrategies (each with different x)
- OBSERVE: The numbers of the ministrategies are candidates

Outline

The Liar Game

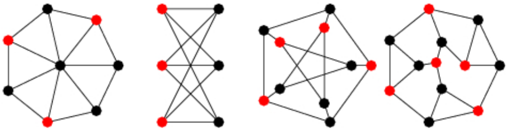
Parallel Maximal Independent Set

Parallel Perfect Matching

Hot-Potato Routing

Independent Set

- Given a graph $G = (V, E)$, find a subset $\mathcal{I} \subseteq V$ such that for all $(u, v) \in E$: $u \notin \mathcal{I}$ or $v \notin \mathcal{I}$
- An independent set $\mathcal{I}$ is *maximal* if $\mathcal{I}$ can not be augmented to a larger independent set
- An independent set $\mathcal{I}$ is *maximum* if for all independent sets $\mathcal{I}'$, we have that $|\mathcal{I}| \geqslant |\mathcal{I}'|$



How do we find a MIS ?

- Finding a maximum Independent Set is NP-hard
- Finding a maximal Independent Set is simple
- Sequential Algorithm
  1. Start with $\mathcal{I} = \emptyset$ and $\mathcal{Q} = V$
  2. While $\mathcal{Q}$ is not empty, choose any $v \in \mathcal{Q}$
  3. Set $\mathcal{I} = \mathcal{I} \cup \{v\}$ and $\mathcal{Q} = \mathcal{Q} \setminus (v \cup N(v))$
  4. Output $\mathcal{I}$
- If at step 2 we choose the *lexicographically* first v, we get the Lexicographically First MIS (LFMIS)

What about Parallelization ?

- If the problem of finding the LFMIS is in NC, then $\mathrm{P} = \mathrm{NC}$!!
- But we can find fast any arbitrary MIS (and not necessary the LFMIS)
- *A Simple Parallel Algorithm for the Maximal Independent Set Problem* [Luby '85]

Some Ideas

- At each step, find an independent set S in parallel. Add S to $\mathcal{I}$ and remove $S \cup \mathrm{N}(S)$
- We must guarantee a *small* number of steps
- At each step, guarantee that a constant fraction of remaining vertices is removed $\Rightarrow$ Difficult!
- What if we guarantee instead that during each step, the number of edges incident to $S \cup \mathrm{N}(S)$ is large?

Sketch of the algorithm

- Mark each node independently with some probability
- Mark with a bias towards vertices of *low* degree $\Rightarrow$ few edges with both nodes marked
- Drop the node with the lowest degree so as to get an Independent Set

The Parallel Algorithm

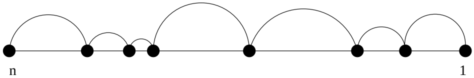
- $I \leftarrow \emptyset$, G the graph
- While G not empty do IN PARALLEL
  - Mark each vertex v independently with probability $\frac{1}{2d(v)}$ (always mark *isolated* nodes)
  - For every edge with both nodes *marked*, unmark the node with the lowest degree (break ties arbitrarily)
  - Let S be the set of all marked nodes, $I \leftarrow I \cup S$
  - Remove from G the vertices $S \cup N(S)$ and all incident edges

Outline of the Analysis

- The algorithm always terminates with a valid Maximal Independent Set
- We have to show that a constant fraction of the remaining edges is removed during each step
- This is enough to give an expected $O(\log n)$ number of steps for the parallel algorithm. Why?

A Random Particle Walk

- Consider a particle on an integer line at position m
- At each step, the particle moves to position $m - X$, where X is a random variable in $[1, m-1]$



- We know that $\mathbb{E}[X] \geqslant g(m)$, where g is a non-decreasing function
- How much does it take for the particle to reach position 1?

Theorem

Let T be the number of steps needed so that the particle reaches position 1 starting from n. Then, $\mathbb{E}[T] \leqslant \int_1^n \frac{dx}{g(x)}$

Good and Bad

Definition

A vertex v is *good* if it has at least $d(v)/3$ neighbors with degree no more than $d(v)$, otherwise, it is *bad*.

Definition

An edge (u, v) is *bad* if both u and v are bad. If at least one of u, v is *good*, then it is good.

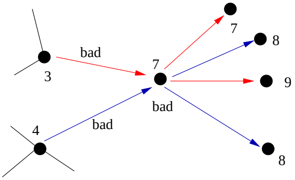
We will show that:

- The number of good edges is a constant fraction of the edges
- A good edge is deleted with constant probability

How many are the good edges?

Lemma

The number of good edges is at least $|E|/2$



- Direct each edge to the higher degree vertex
- If (u, v) is bad, both u, v are bad and let (u, v) directed towards v
- v has at least twice as many outgoing edges as incoming
- We can thus map each incoming bad edge to v to a pair of outgoing edges
- The number of bad edges can not be more than $|E|/2$

Why being a good vertex is good? (1)

Lemma

If v is a good vertex and $\mathrm{d}(v) > 0$, the probability that a vertex in $\mathrm{N}(v)$ gets marked is at least $1 - e^{-1/6}$

Proof.

- $w \in \mathrm{N}(v)$ gets marked with probability $\frac{1}{2\mathrm{d}(w)}$
- v has at least $\mathrm{d}(v)/3$ neighbors with degree at most $\mathrm{d}(v)$, which are marked with probability at least $\frac{1}{2\mathrm{d}(v)}$
- *Full independence of marking* $\Rightarrow$ probability that none of these neighbors is marked at most $\left(1 - \frac{1}{2\mathrm{d}(v)}\right)^{\mathrm{d}(v)/3} \leqslant e^{-1/6}$

$\square$

Why being a good vertex is good? (2)

Lemma

If w is marked, it is chosen in S with probability at least $1/2$

Proof.

- Let $H(w) = \{v \mid v \in N(w), d(v) \geqslant d(w)\}$, the neighbors of w with degree greater than $d(w)$
- w is unmarked only if a vertex in $H(w)$ is marked
- $\Pr[w \notin S \mid w \text{ marked}] \leqslant \sum_{v \in H(w)} \Pr[v \text{ marked} \mid w \text{ marked}]$
- *Pairwise independence* $\Rightarrow \sum_{v \in H(w)} \Pr[v \text{ marked}]$
- $\Pr[w \notin S \mid w \text{ marked}] \leqslant \sum_{v \in H(w)} \frac{1}{2d(w)} \leqslant \frac{1}{2}$

$\square$

Why being a good vertex is good? (3)

Lemma

If v is a good vertex, it is removed with probability at least $\frac{1-e^{-1/6}}{2}$

- For v to be removed, it is enough that a neighbor gets marked and then is chosen in S
- A neighbor is marked with probability $\geqslant 1 - e^{-1/6}$
- If a vertex is marked, it is chosen in S with probability at least $1/2$

Lemma

If an edge is good, it is deleted with probability at least $\frac{1-e^{-1/6}}{2}$

Pairwise vs Mutual Independence

- Consider the set of events $A_1, A_2, \ldots, A_n$
- The events $A_1, A_2, \ldots, A_n$ are *mutually independent* if $\Pr[A_1 \cap A_2 \ldots \cap A_n] = \Pr[A_1] \cdot \Pr[A_2] \ldots \Pr[A_n]$
- The events $A_1, A_2, \ldots, A_n$ are *pairwise independent* if for every $i, j : \Pr[A_i \cap A_j] = \Pr[A_i] \cdot \Pr[A_j]$
- Pairwise independence is weaker than mutual independence

Parallel MIS Revisited

- The analysis involved only one inequality where mutual independence of events is used
- We can provide a similar inequality and prove a constant probability with pairwise independence
- Thus the algorithm needs only pairwise independent random bits
- Why does this help?

Derandomization using Pairwise Independence

- Consider a probability space where the sample space consists of all binary vectors of length n (e.g. $\{00, 01, 01, 11\}$)
- For any binary vector $\langle b_0, \ldots, b_{n-1} \rangle$ we define the event $E_i : b_i = 1$
- Denote $p_i = Pr[E_i]$
- If the events E_i are mutually independent, we need $\Omega(n)$ random bits: one for each bit of the binary vector
- But we want pairwise independence of the events E_i

Derandomization using Pairwise Independence

- We define a new sample space
- Consider the $n \times q$ matrix A (q is a prime between n and $2n$)

$$A[i][j] = \begin{cases} 1 & \text{if } 0 \leqslant j \leqslant \lfloor p_i \cdot q \rfloor - 1 \text{ ,} \\ 0 & \text{otherwise.} \end{cases}$$

- Choose x, y uniformly at random from $0, 1, \ldots, q-1$
- Define a random binary vector as $b_{x,y} = \langle b^0_{x,y}, \ldots, b^{n-1}_{x,y} \rangle$ where

$$b^i_{x,y} = A[i][(x + y \cdot i) \bmod q]$$

- This creates a sample space of q^2 binary vectors, where each vector has probability $1/q^2$

Derandomization using Pairwise Independence

Lemma

$\Pr[E_i] = p_i' = \lfloor p_i \cdot q \rfloor / q$

There are exactly q pairs of x, y such that $(x + y \cdot i) \equiv l \pmod{q}$ for fixed l. E_i occurs when $(x + y \cdot i) \pmod{q}$ is between 0 and $\lfloor p_i \cdot q \rfloor - 1$. Thus, we have $p_i' \cdot q^2$ binary vectors where E_i occurs.

Lemma

$\Pr[E_i \cap E_j] = p_i' \cdot p_j'$

For fixed l_i, l_j, there exists exactly one pair x, y such that $(x + y \cdot i) \equiv l_i \pmod{q}$ and $(x + y \cdot j) \equiv l_j \pmod{q}$. The events E_i and E_j occur both for $(p_i' q) \cdot (p_j' q)$ pairs of l_i, l_j

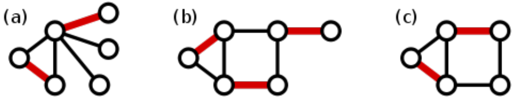
Putting all pieces together

- The new sample space has only q^2 samples, which is $O(n^2)$
- We can try run all these samples in parallel by using only polynomially more processors
- We only have to handle the problem that the new probabilities are not exactly the same (omitted)
- MIS belongs in NC!

Outline

Matchings

- Let $G = (V, E)$ be a graph
- A *matching* in G is a set of edges $M \subset E$ such that no two edges are incident
- A *maximum* matching is a matching with maximum number of edges [c]
- A *perfect matching* is a matching containing an edge incident to every vertex of G [b]



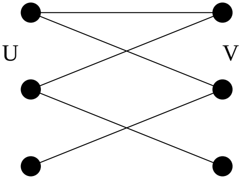
The Tutte Matrix

For simplicity, we will deal with bipartite graphs such that $G = (U, V, E)$ and $U = \{u_1, \dots u_n\}$, $V = \{v_1, \dots, v_n\}$

Definition

The Tutte Matrix A of a bipartite graph G is a $n \times n$ matrix such that

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (u_i, v_j) \in E \text{ ,} \\ 0 & \text{otherwise.} \end{cases}$$



$$A = \begin{pmatrix} x_{11} & x_{12} & 0 \\ x_{21} & 0 & x_{23} \\ 0 & x_{32} & 0 \end{pmatrix}$$

Determinant of the Tutte Matrix

Theorem

$\det(A) \neq 0 \Leftrightarrow G$ *has a perfect matching*

Proof.

- $\det(A) = \sum_\pi \mathrm{sgn}(\pi) \cdot \prod_{i=1}^{n} A_{i\pi(i)}$, where the sum is over all permutations π of $\{1, 2, \ldots, n\}$
- Each monomial corresponds to a unique possible perfect matching in G
- The monomial is non-zero off the matching exists in G
- Every pair of monomials differs in at least two variables

$\square$

Decision version of Perfect Matching

- $\det(A)$ is a polynomial with n^2 variables
- Use the Schwartz-Zippel algorithm for Polynomial Identity Testing to check whether $\det(A) = 0$
- Computing the determinant is used as a subroutine
- A $n \times n$ determinant can be computed in $O(\log^2 n)$ time using polynomially many processors

Lemma

Deciding whether a graph G has a perfect matching is in RNC

Finding a Perfect Matching Sequentially

- Notice that if edge e belongs to a perfect matching, then for the graph $G' = G \setminus e$ we have that $\det(A') \neq 0$
- Sequential Matching
  1. Pick an arbitrary edge (i, j) of G
  2. Check whether $G' = G \setminus \{i, j\}$ has a perfect matching
  3. IF YES, add edge (i, j) to the matching M and $G \leftarrow G'$
  4. ELSE $G \leftarrow G \setminus \{(i, j)\}$.
  5. While M is not a perfect matching, repeat 1

Finding a Perfect Matching: Ideas

- Not parallelizable: G may have many perfect matchings, the processors must be coordinated to search for the same matching!
- IDEA: isolate a perfect matching and then employ the algorithm
- HOW? assign random weights and look for the *minimum weight* matching

Isolating Lemma

Lemma (Isolating Lemma)

Let $S = \{e_1, \ldots, e_m\}$ *and* $S_1, \ldots, S_k \subseteq S$. *Let each element* $e_i \in S$ *have a weight* w_i *picked u.a.r. from* $\{0, 1, \ldots, 2m-1\}$. *Define the weight of* S_j *as* $w(S_j) = \sum_{e_i \in S_j} w_i$. *Then*

$$\mathbb{P}\mathrm{r}[\exists \text{ a unique set } S_i \text{ of minimum weight}] \geqslant 1/2$$

A *counterintuitive* lemma: We may have as many as 2^m sets, but we have only $2m^2$ different weights!

Isolating Lemma: Proof (1)

- We say that an element $e \in S$ is *ambiguous* if $\min_{S_j | e \in S_j} w(S_j) = \min_{S_j | e \notin S_j} w(S_j)$
- If no bad element exists, then there exists a unique minimum weight set
- We have to bound the probability that a bad element exists
- Principle of Deferred Decisions: suppose that we have chosen random weights for all elements except e_i
- Then, $W^- = \min_{S_j | e_i \notin S_j} w(S_j)$ is already fixed
- Consider $W^+ = \min_{S_j | e_i \in S_j} w(S_j)$ with $w_i = 0$

Isolating Lemma: Proof (2)

- There is at most one value of w_i such that $W^- = W^+ + w_i$
- Thus, $\mathbb{P}\mathrm{r}[e_i \text{ is bad}] \leqslant 1/2m$
- Union Bound

$$\mathbb{P}\mathrm{r}[\exists \text{ a bad element}] \leqslant \sum_{i=1}^{m} \mathbb{P}\mathrm{r}[e_i \text{ is bad}] \leqslant \sum_{i=1}^{m} \frac{1}{2m} = \frac{1}{2}$$

The Parallel Algorithm

- For each edge (u_i, v_j) pick a random weight w_{ij} from $\{0, 1, \ldots, 2|E| - 1\}$
- The sets S_j denote all the perfect matchings in G
- Isolating Lemma: there is exists a unique minimum weight perfect matching with probability $\geqslant 1/2$
- Assign the values $x_{ij} = 2^{w_{ij}}$ to the variables in A to obtain matrix D

Lemma

If G has a unique minimum weight perfect matching M_0 of weight W_0, then $\det(D) \neq 0$ and the largest power of 2 that divides $\det(D)$ is 2^{W_0}

$$\det(D) = \sum_{\pi} \operatorname{sgn}(\pi) \cdot \prod_{i=1}^{n} 2^{w_{i\pi(i)}} = \sum_{M} \pm 2^{w(M)}$$

The Parallel Algorithm

- Pick IN PARALLEL random weights w_{ij} for each edge
- Compute IN PARALLEL $\det(D)$ and W_0
- for each edge (u_i, v_j) do IN PARALLEL
    - Compute $\det(D_{ij})$ (we remove row i and column j from D)
    - Compute $r_{ij} = \det(D_{ij})\frac{2^{w_{ij}}}{2^{W_0}}$
    - If r_{ij} is ODD, add (u_i, v_j) to M
- Check whether M is a valid perfect matching

Correctness

Lemma

The algorithm outputs a perfect matching with probability at least 1/2

- With probability $\geqslant 1/2$, a unique minimum weight perfect matching exists
- $\det(D_{ij})$ corresponds to the perfect matchings in $G \setminus \{i, j\}$

$$\det(D_{ij}) = \sum_{M \in \mathcal{M}(G \setminus \{i,j\})} \pm 2^{w(M)} = 2^{-w_{ij}} \sum_{M \cup (i,j) \in \mathcal{M}(G)} \pm 2^{w(M \cup (i,j))}$$

- If the minimum weight matching is unique, r_{ij} is odd iff $(i, j) \in M_0$

A few Notes

- We can convert the algorithm to a Las Vegas algorithm
- We can also adapt the algorithm to work for general graphs
- It is an open question whether there is a deterministic fast parallel algorithm for perfect matchings

Outline

What is Hot-potato Routing?

- No buffering of packets
- Any packet arriving at a node other than its destination must *immediately* be forwarded to another node (*would you not want to get rid of a hot potato*?)

- Advantages: algorithms perform very well in practice, simple hardware (e.g. optical networks)
- Drawback: hard theoretical analysis

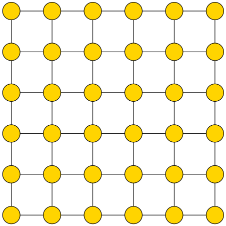
What is Hot-potato Routing?

- No buffering of packets
- Any packet arriving at a node other than its destination must *immediately* be forwarded to another node (*would you not want to get rid of a hot potato*?)

- Advantages: algorithms perform very well in practice, simple hardware (e.g. optical networks)
- Drawback: hard theoretical analysis

The Model

- A $n \times n$ rectangular mesh



- *Synchronous* network: at each *step*, at most one packet is routed to each link
- *Batch Routing:* at time 0, each node sends a packet to a specified destination node
  - **Batch Permutation**
  - **Random Destinations**
  - General Batch problem

A Greedy Approach

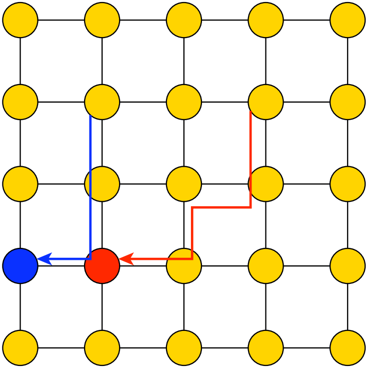
- Greedy: Packets prefer links towards the destination nodes
- When routed
  - *Good links*: bring the packet closer to destination
  - *Bad links*: further away from destination (*deflected* packet)
- We need to specify two things:
  - How do the packets move?
  - How do we resolve conflicts of preference?

The Algorithm (Sketch)

- Packets have 3 states with decreasing priority
  1. RUNNING
  2. EXCITED
  3. NORMAL
- Initially, all packets are *normal* and routed greedily: a node is forwarded to a *good* link, unless a node with higher priority has the same preference (ties break arbitrarily)
- Each time a packet gets deflected, it has a small probability p of getting excited: it tries to take one of the two shortest "one-bend" paths to its destination (*home run*)
- If the home run is interrupted, the nodes comes back to *normal*

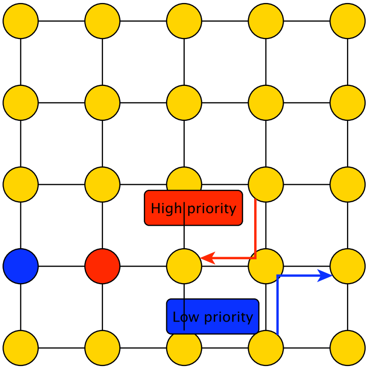
Normal State

Packets are routed greedily towards one of the two links that bring them closer to the destination node



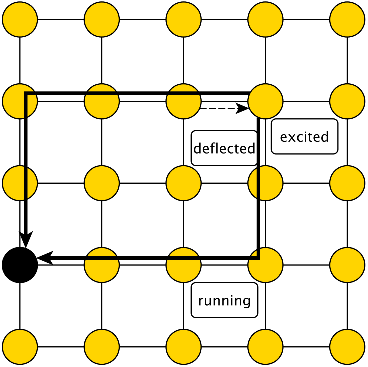
Deflected Packets

- A packet may be *deflected* when a packet with higher priority uses the same link
- With a small probability p, the packet gets *excited*



Home Run

- An *excited* packet follows u.a.r one of the two "one bend" paths towards the destination node
- Then, it changes to *running* state



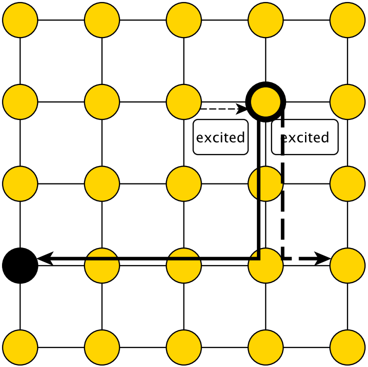
- If interrupted by a higher priority packet, returns to *normal*

Analysis

- What is the probability of a packet completing a home run?
- We consider a powerful adversary: the adversary is allowed to place the other packets at nodes in the mesh, choose their destinations and deflect them at will in order to get them "excited"
- Intuition: The adversary has limited ammunition to make the home run fail

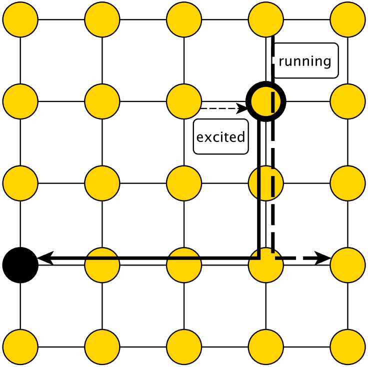
Excited vs Excited

An excited node does not conflict with another excited node with probability at least $(1-p)^3$



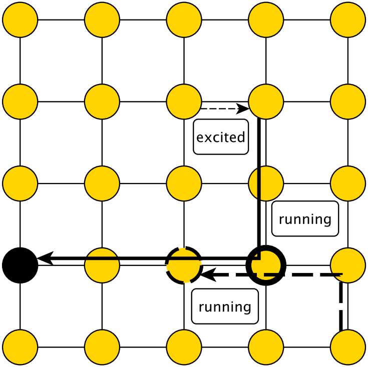
Excited vs Running

- A packet π gets excited at (x, y) at time t
- A node holds no excited packet with probability $p' = (1-p)^4$
- π may be interrupted by a running packet excited at time $t - d$ at node $(x, y + d) \Rightarrow$ at most $n - 1$ such packets, probability of no conflict at least $p' \cdot p'^{n-1} = (1-p)^{4n}$



Running vs Running

- A running packet π conflicts another running packet only during the bend
- π may be interrupted by a running packet having destination at the same row $\Rightarrow n-1$ such packets, each excited with probability p
- probability of no conflict at least $(1-p)^n$



Summing Up

- The probability of completing a home run is

$$2 \cdot \frac{1}{2} \cdot (1-p)^3 \cdot (1-p)^{4n} \cdot (1-p)^n$$

- for $p = 1/n$, the probability is constant c
- each time a packet gets deflected, it reaches the destination with probability $p \cdot c = c/n$
- thus, the expected number of deflections of a packet is $O(n)$
- if a packet is deflected x times, then it will reach its destination in at most $2x + 2n - 2$ steps

Lemma

A packet reaches the destination in expected $O(n)$ steps

More To Do

- If we allow the probability p to vary with time, we can show that

Lemma

With high probability, all packets reach their destination nodes in at most $O(n \ln n)$ *steps*

- For the general batch problem, if m is the maximum row/column congestion of destination nodes, then

Lemma

With high probability, all packets reach their destination nodes in at most $O(m \ln n)$ *steps*

The End

Thank You !