Hybrid Algorithms for Graph Problems

Joint work with Ryan Williams and Maverick Woo

Hybrid Algorithms for Graph Problems

Defying Hardness using Graph Minors and Separators

Joint work with Ryan Williams and Maverick Woo

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Running Time, or

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Running Time, or

Space, or

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Running Time, or

Space, or

Simultaneous Time and Space, or

Introduction

Conventional algorithms guarantee *good*

performance under a prescribed *measure:*

Running Time, or

Space, or

Simultaneous Time and Space, or

Approximation Ratio and Time, etc.

Hybrid Algorithms

Hybrid Algorithms

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

Hybrid Algorithms

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

E.g.

Hybrid Algorithms

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

E.g.

h_1 approximates the optimal solution within a factor of α and runs in polynomial time.

Hybrid Algorithms

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

E.g.

h_1 approximates the optimal solution within a factor of α and runs in polynomial time.

h_2 solves the problem exactly but runs in subexponential time ($2^{o(n)}$).

Hybrid Algorithms cont.

Hybrid Algorithms cont.

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

Hybrid Algorithms cont.

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics* , optimizing different complexity measures.

A *selector* S which on each instance selects in polynomial time the *best* heuristic.

Hybrid Algorithms cont.

Hybrid Algorithms cont.

Defying Hardness: **Some NP-Hard problems are known or conjectured to be *hard* on several complexity measures m_i separately.**

Hybrid Algorithms cont.

Defying Hardness: **Some NP-Hard problems are known or conjectured to be *hard* on several complexity measures m_i separately.**

E.g. Clique cannot be approximated within a factor of n^{ϵ}, and cannot be solved in polynomial time, unless P$=$NP.

Hybrid Algorithms cont.

Defying Hardness: **Some NP-Hard problems are known or conjectured to be *hard* on several complexity measures m_i separately.**

E.g. Clique cannot be approximated within a factor of n^{ϵ}, and cannot be solved in polynomial time, unless P$=$NP.

There exist hybrid algorithms for NP-Hard problems which for each h_i (on the instances on which S chooses h_i to be run) do *strictly* better than the corresponding known hardness guarantees m_i.

Max Cut

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Max Cut

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5})$ by Scott and Sorkin, 2003, or in $O(2^{\omega n/3})$ by Ryan Williams, 2004.

Max Cut

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5})$ by Scott and Sorkin, 2003, or in $O(2^{\omega n/3})$ by Ryan Williams, 2004.

Approximable within 0.87856 using SDP by Goemans and Williamson, 1995 and within 0.5 by Sahni and Gonzales, 1976 without using SDP.

Max Cut

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5})$ by Scott and Sorkin, 2003, or in $O(2^{\omega n/3})$ by Ryan Williams, 2004.

Approximable within 0.87856 using SDP by Goemans and Williamson, 1995 and within 0.5 by Sahni and Gonzales, 1976 without using SDP.

No better than $1/2$-approximation is known without using SDP.

A Simple Hybrid Algorithm for Max Cut

A Simple Hybrid Algorithm for Max Cut

Find a maximum matching, M.

A Simple Hybrid Algorithm for Max Cut

Find a maximum matching, M.

If $|M| < \varepsilon \frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M to find the maximum. Add the vertices from the independent set $V - M$ so that the cut is maximized.

A Simple Hybrid Algorithm for Max Cut

Find a maximum matching, M.

If $|M| < \varepsilon\frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M to find the maximum. Add the vertices from the independent set $V - M$ so that the cut is maximized.

If $|M| \geq \varepsilon\frac{m}{2}$,

for each edge in M, with probability $1/2$ choose which of its endpoints to put in A. Put the other endpoint in B;

A Simple Hybrid Algorithm for Max Cut

Find a maximum matching, M.

If $|M| < \varepsilon\frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M to find the maximum. Add the vertices from the independent set $V - M$ so that the cut is maximized.

If $|M| \geq \varepsilon\frac{m}{2}$,

for each edge in M, with probability $1/2$ choose which of its endpoints to put in A. Put the other endpoint in B;

for each vertex v not covered by M, with probability $1/2$ choose whether to place it in A or B.

Max Cut cont.

Max Cut cont.

If $|M| < \varepsilon \frac{m}{2}$,

M has at most εm vertices, and the rest of the vertices form an independent set I. Placing the vertices of I so that the cut is maximized, given an arrangement of M is easy.

We get an exact solution in $\tilde{O}(2^{\varepsilon m})$ time.

Max Cut cont.

If $|M| < \varepsilon \frac{m}{2}$,

M has at most εm vertices, and the rest of the vertices form an independent set I. Placing the vertices of I so that the cut is maximized, given an arrangement of M is easy.

We get an exact solution in $\tilde{O}(2^{\varepsilon m})$ time.

If $|M| \geq \varepsilon \frac{m}{2}$,

The probability that an edge not in M crosses the cut is $1/2$. Hence we get a cut of expected size at least $(\varepsilon \frac{m}{2}) + \frac{1}{2}(m - \varepsilon \frac{m}{2}) = (\frac{1}{2} + \frac{\varepsilon}{4})m$.

We get a $(\frac{1}{2} + \frac{\varepsilon}{4})$-approximation with no semidefinite programming.

The Longest Path Problem

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: Longest Path is hard to approximate within $2^{O(\frac{\log n}{\log \log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$\left(2^{O(n^{\delta})}\right)$.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: Longest Path is hard to approximate within $2^{O(\frac{\log n}{\log \log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^{\delta})})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$; can be extended to $2^{O(L)}$, where L is the length of the longest path.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: Longest Path is hard to approximate within $2^{O(\frac{\log n}{\log \log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$; can be extended to $2^{O(L)}$, where L is the length of the longest path.

We give a *hybrid* algorithm which for any $\ell(n)$

- either finds a path of length ℓ, or
- solves the Longest Path exactly in time$2^{O(\ell \log L \log \frac{n}{\ell})}$.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: Longest Path is hard to approximate within $2^{O(\frac{\log n}{\log\log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$; can be extended to $2^{O(L)}$, where L is the length of the longest path.

We give a *hybrid* algorithm which for any $\ell(n)$

- either finds a path of length ℓ, or
- solves the Longest Path exactly in time $2^{O(\ell \log L \log \frac{n}{\ell})}$.

Notice that for $\ell = n/polylog(n)$ we get subexponential exact running time and a polylog approximation.

A Path/Separator Lemma

A Path/Separator Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Separator which either finds a path of length at least ℓ or a $1/2 - 1/2$ separator of size at most ℓ.

A Path/Separator Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Separator which either finds a path of length at least ℓ or a $1/2 - 1/2$ separator of size at most ℓ.

1. Start from a node v and add vertices forming a path P until a node f with no neighbors is reached.

A Path/Separator Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Separator which either finds a path of length at least ℓ or a $1/2 - 1/2$ separator of size at most ℓ.

1. Start from a node v and add vertices forming a path P until a node f with no neighbors is reached.

2. If P has length at least ℓ, stop and output P.

A Path/Separator Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Separator which either finds a path of length at least ℓ or a $1/2 - 1/2$ separator of size at most ℓ.

1. Start from a node v and add vertices forming a path P until a node f with no neighbors is reached.
2. If P has length at least ℓ, stop and output P.
3. Else, remove f from P and add it to A.

   If $|A| = n/2$, stop and output P as a separator.

   Otherwise, attempt to continue P with vertices from $V - P - A$ until f with no neighbors is reached. Go to 2.

Tree Width

A *tree decomposition* of a graph G is a tree T and a *bag* collection $W = \{W_1, \ldots, W_{|T|}\}$ such that

Tree Width

A *tree decomposition* of a graph G is a tree T and a *bag* collection $W = \{W_1, \ldots, W_{|T|}\}$ such that

1. $\bigcup_{i=1}^{|T|} W_i = V(G)$,

Tree Width

A *tree decomposition* of a graph G is a tree T and a *bag* collection $W = \{W_1, \ldots, W_{|T|}\}$ such that

1. $\bigcup_{i=1}^{|T|} W_i = V(G)$,
2. if $(u, v) \in E(G)$, then exists W_j with $u, v \in W_j$,

Tree Width

A *tree decomposition* of a graph G is a tree T and a *bag* collection $W = \{W_1, \ldots, W_{|T|}\}$ such that

1. $\bigcup_{i=1}^{|T|} W_i = V(G)$,
2. if $(u, v) \in E(G)$, then exists W_j with $u, v \in W_j$,
3. if j lies on a path in T from i to k, then $W_i \cap W_k \subseteq W_j$.

Tree Width

A *tree decomposition* of a graph G is a tree T and a *bag* collection $W = \{W_1, \ldots, W_{|T|}\}$ such that

1. $\bigcup_{i=1}^{|T|} W_i = V(G)$,
2. if $(u, v) \in E(G)$, then exists W_j with $u, v \in W_j$,
3. if j lies on a path in T from i to k, then $W_i \cap W_k \subseteq W_j$.

The width of a tree decomposition is the maximum size of a bag W_i, minus 1. The *tree width* of a graph G is the minimum width of a tree decomposition of G.

Towards a Hybrid Algorithm

Towards a Hybrid Algorithm

A result by Matousek and Thomas implies: if G has *treewidth* at most K, then there is a $O(L^{K+1}n)$ algorithm to find a path of length L in G, or to determine that no such exists.

Towards a Hybrid Algorithm

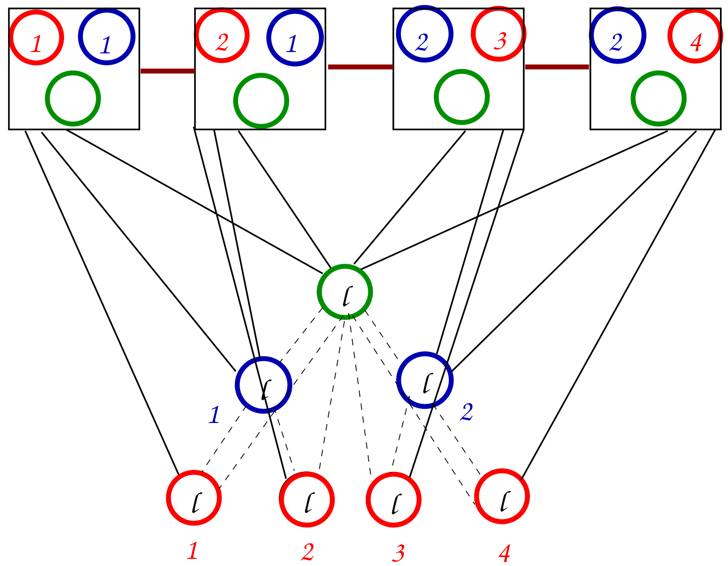
A result by Matousek and Thomas implies: if G has *treewidth* at most K, then there is a $O(L^{K+1}n)$ algorithm to find a path of length L in G, or to determine that no such exists.

We show: If G has a separator decomposition with separator size ℓ, then G has a tree decomposition of width at most $O(\ell \log \frac{n}{\ell})$.

Towards a Hybrid Algorithm

A result by Matousek and Thomas implies: if G has *treewidth* at most K, then there is a $O(L^{K+1}n)$ algorithm to find a path of length L in G, or to determine that no such exists.

We show: If G has a separator decomposition with separator size ℓ, then G has a tree decomposition of width at most $O(\ell \log \frac{n}{\ell})$.



Hybrid Algorithm for Longest Path

Hybrid Algorithm for Longest Path

1. Run Path-Separator algorithm on G and ℓ.

Hybrid Algorithm for Longest Path

1. Run Path-Separator algorithm on G and ℓ.

2. If a path of length ℓ is found, return it.

Hybrid Algorithm for Longest Path

1. Run Path-Separator algorithm on G and ℓ.
2. If a path of length ℓ is found, return it.
3. Otherwise the algorithm returns a separator and two *disjoint* parts G_L and G_R of size at most $n/2$.

Hybrid Algorithm for Longest Path

1. Run Path-Separator algorithm on G and ℓ.
2. If a path of length ℓ is found, return it.
3. Otherwise the algorithm returns a separator and two *disjoint* parts G_L and G_R of size at most $n/2$.
4. Recurse on G_L and G_R to obtain either a path of length ℓ, or a separator decomposition.

Hybrid Algorithm for Longest Path

1. Run Path-Separator algorithm on G and ℓ.

2. If a path of length ℓ is found, return it.

3. Otherwise the algorithm returns a separator and two *disjoint* parts G_L and G_R of size at most $n/2$.

4. Recurse on G_L and G_R to obtain either a path of length ℓ, or a separator decomposition.

5. Run the Matousek and Thomas algorithm on the tree decomposition obtained from the separator tree, on successive powers of 2 for the path length, to obtain the longest path in $\tilde{O}(2^{\ell \log L \log \frac{n}{\ell}})$.

Graph Minors and Separators

Graph Minors and Separators

A subgraph M is an *H–minor* of G if

Graph Minors and Separators

A subgraph M is an H*–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that

Graph Minors and Separators

A subgraph M is an H*–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that

2. if $(v_i, v_j) \in E(H)$, then there is an edge between A_i and A_j in M.

Graph Minors and Separators

A subgraph M is an *H–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that

2. if $(v_i, v_j) \in E(H)$, then there is an edge between A_i and A_j in M.

S is a α – separator of G if $V(G)$ can be partitioned into A, B, S so that

Graph Minors and Separators

A subgraph M is an *H–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that
2. if $(v_i, v_j) \in E(H)$, then there is an edge between A_i and A_j in M.

S is a α – separator of G if $V(G)$ can be partitioned into A, B, S so that

1. there are no edges between A and B, and

Graph Minors and Separators

A subgraph M is an *H–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that

2. if $(v_i, v_j) \in E(H)$, then there is an edge between A_i and A_j in M.

S is a α – separator of G if $V(G)$ can be partitioned into A, B, S so that

1. there are no edges between A and B, and

2. $|A| \le |B| \le \alpha|V(G)|$.

Graph Minors and Separators

A subgraph M is an *H–minor* of G if

1. the vertices of M can be partitioned into $|V(H)|$ parts $A_1, \ldots, A_{|V(H)|}$ called *supernodes*, A_i corresponding to $v_i \in V(H)$, so that
2. if $(v_i, v_j) \in E(H)$, then there is an edge between A_i and A_j in M.

S is a α – separator of G if $V(G)$ can be partitioned into A, B, S so that

1. there are no edges between A and B, and
2. $|A| \leq |B| \leq \alpha|V(G)|$.

Often one says S is a $1/3 - 2/3$ – separator, meaning that in the worst case $|A| = \frac{1}{3}|V(G)|$ and $|B| = \frac{2}{3}|V(G)|$.

A More General Minor – Separator Theorem

The following is a generalization of Plotkin, Rao, Smith, 1994:

Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

A More General Minor – Separator Theorem

The following is a generalization of Plotkin, Rao, Smith, 1994:

Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or

A More General Minor – Separator Theorem

The following is a generalization of Plotkin, Rao, Smith, 1994:

Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or
- finds a $1/3 - 2/3$–separator S of G of size $O(\frac{n}{\ell} + \ell h \log n)$.

A More General Minor – Separator Theorem

The following is a generalization of Plotkin, Rao, Smith, 1994:

Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or
- finds a $1/3 - 2/3$–separator S of G of size $O(\frac{n}{\ell} + \ell h \log n)$.

For large values of ℓ the above can be generalized to finding a minor, or finding a $1/2 - 1/2$–separator.

A More General Minor-Separator Theorem

A More General Minor-Separator Theorem

Fix a parameter $\ell \geq 1$.

A More General Minor-Separator Theorem

Fix a parameter $\ell \geq 1$.

We build a minor M and a set B with $M \cap B = \emptyset$ and B having few neighbors in $W = (V(G) - M - B)$.

A More General Minor-Separator Theorem

Fix a parameter $\ell \geq 1$.

We build a minor M and a set B with $M \cap B = \emptyset$ and B having few neighbors in $W = (V(G) - M - B)$.

In the end either M will be an H-minor of the desired size, or B will be the larger partition of V with $S = N(B)$ becoming the separator.

A More General Minor-Separator Theorem

Fix a parameter $\ell \geq 1$.

We build a minor M and a set B with $M \cap B = \emptyset$ and B having few neighbors in $W = (V(G) - M - B)$.

In the end either M will be an H-minor of the desired size, or B will be the larger partition of V with $S = N(B)$ becoming the separator.

At each stage M is an H'-minor of G where H' is an induced subgraph of H.

A More General Minor-Separator Theorem

Fix a parameter $\ell \geq 1$.

We build a minor M and a set B with $M \cap B = \emptyset$ and B having few neighbors in $W = (V(G) - M - B)$.

In the end either M will be an H-minor of the desired size, or B will be the larger partition of V with $S = N(B)$ becoming the separator.

At each stage M is an H'-minor of G where H' is an induced subgraph of H.

B is a subset of $V(G)$, $B \cap M = \emptyset$, and $|N(B) \cap W| \leq \frac{|B|}{\ell}$.

A More General Minor-Separator Theorem

A More General Minor-Separator Theorem

At each stage we look at a vertex v from $H - H'$ and its neighbors $u_1, \ldots, u_k$ in H'.

A More General Minor-Separator Theorem

At each stage we look at a vertex v from $H - H'$ and its neighbors $u_1, \ldots, u_k$ in H'.

If any of the supernodes in M corresponding to the u_i have no neighbors in W, we move them to B (updating M and thus H').

A More General Minor-Separator Theorem

At each stage we look at a vertex v from $H - H'$ and its neighbors $u_1, \ldots, u_k$ in H'.

If any of the supernodes in M corresponding to the u_i have no neighbors in W, we move them to B (updating M and thus H').

If B became large, we have found a separator.

A More General Minor-Separator Theorem

At each stage we look at a vertex v from $H - H'$ and its neighbors $u_1, \ldots, u_k$ in H'.

If any of the supernodes in M corresponding to the u_i have no neighbors in W, we move them to B (updating M and thus H').

If B became large, we have found a separator.

Otherwise, we pick a node w in W and start doing a *two-stage BFS* in W.

A More General Minor-Separator Theorem

At each stage we look at a vertex v from $H - H'$ and its neighbors $u_1, \ldots, u_k$ in H'.

If any of the supernodes in M corresponding to the u_i have no neighbors in W, we move them to B (updating M and thus H').

If B became large, we have found a separator.

Otherwise, we pick a node w in W and start doing a *two-stage BFS* in W. This BFS finds the new supernode corresponding to v, or more nodes to add to B. If W becomes smaller than $2n/3$, we have found a separator.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

1. Starting from R we do two steps of BFS, setting $T = N(N(R))$.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

1. Starting from R we do two steps of BFS, setting $T = N(N(R))$.
2. If both R did not expand too much ($|T| \leq |R|(1 + 1/\ell)$), and $W - R$ did not shrink too much ($|W - R| \leq (1 + 1/\ell)|W - T|$), stop.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

1. Starting from R we do two steps of BFS, setting $T = N(N(R))$.
2. If both R did not expand too much ($|T| \leq |R|(1 + 1/\ell)$), and $W - R$ did not shrink too much ($|W - R| \leq (1 + 1/\ell)|W - T|$), stop.
3. Otherwise, set $R = T$ and continue from 2.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

1. Starting from R we do two steps of BFS, setting $T = N(N(R))$.
2. If both R did not expand too much ($|T| \leq |R|(1 + 1/\ell)$), and $W - R$ did not shrink too much ($|W - R| \leq (1 + 1/\ell)|W - T|$), stop.
3. Otherwise, set $R = T$ and continue from 2.
4. If in the end R contains a neighbor n_i of each u_i, return a *shortest path tree* from all the n_i to w. This is the new supernode for vertex v and M is a $H' \cup \{v\}$–minor.

Two Stage BFS in W

Recall the parameter $\ell \geq 1$. Let $R = \{w\}$.

1. Starting from R we do two steps of BFS, setting $T = N(N(R))$.
2. If both R did not expand too much ($|T| \leq |R|(1 + 1/\ell)$), and $W - R$ did not shrink too much ($|W - R| \leq (1 + 1/\ell)|W - T|$), stop.
3. Otherwise, set $R = T$ and continue from 2.
4. If in the end R contains a neighbor n_i of each u_i, return a *shortest path tree* from all the n_i to w. This is the new supernode for vertex v and M is a $H' \cup \{v\}$–minor.
5. Otherwise, the *smaller* of $R' = R \cup N(R)$, and $R'' = V - R'$ has *few neighbors in* W ($|N(R')| \leq \frac{|R'|}{\ell}$ and $|N(R'')| \leq \frac{|R''|}{\ell}$).
   We add it to B.

Minor or Separator

Minor or Separator

If we find a minor M, its size is $O(\ell h \log n)$.

Minor or Separator

If we find a minor M, its size is $O(\ell h \log n)$.

Why?: When a supernode corresponding to $v \in V(H)$ is added it has size at most $O(deg_H(v)\ell \log n)$ since the BFS tree has depth at most $2\log_{(1+1/\ell)} n \leq 2\ell \log n$, and since there are at most $deg_H(v)$ neighbors to be covered.

Minor or Separator

If we find a minor M, its size is $O(\ell h \log n)$.

Why?: When a supernode corresponding to $v \in V(H)$ is added it has size at most $O(deg_H(v)\ell \log n)$ since the BFS tree has depth at most $2\log_{(1+1/\ell)} n \leq 2\ell \log n$, and since there are at most $deg_H(v)$ neighbors to be covered.

Once a supernode is added, its size is never changed, unless the supernode is removed.

Minor or Separator

If we find a minor M, its size is $O(\ell h \log n)$.

Why?: When a supernode corresponding to $v \in V(H)$ is added it has size at most $O(deg_H(v)\ell \log n)$ since the BFS tree has depth at most $2\log_{(1+1/\ell)} n \leq 2\ell \log n$, and since there are at most $deg_H(v)$ neighbors to be covered.

Once a supernode is added, its size is never changed, unless the supernode is removed.

In the end the size of the minor is $O(\sum_{v\in H} deg_H(v)\ell \log n) = O(h\ell \log n)$ for $h = |E(H)|$.

Minor or Separator

If we find a minor M, its size is $O(\ell h \log n)$.

Why?: When a supernode corresponding to $v \in V(H)$ is added it has size at most $O(deg_H(v)\ell \log n)$ since the BFS tree has depth at most $2 \log_{(1+1/\ell)} n \leq 2\ell \log n$, and since there are at most $deg_H(v)$ neighbors to be covered.

Once a supernode is added, its size is never changed, unless the supernode is removed.

In the end the size of the minor is $O(\sum_{v \in H} deg_H(v)\ell \log n) = O(h\ell \log n)$ for $h = |E(H)|$.

The separator consists of the (unfinished) minor M and of the neighbors of B in W. Since $|N_W(B)| \leq |B|/\ell \leq \frac{2n}{3\ell}$, the size of the separator is $O(n/\ell + \ell h \log n)$.

A More General Minor-Separator Theorem

To summarize: Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

A More General Minor-Separator Theorem

To summarize: Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or

A More General Minor-Separator Theorem

To summarize: Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or
- finds a $1/3 - 2/3$–separator S of G of size $O(\frac{n}{\ell} + \ell h \log n)$.

A More General Minor-Separator Theorem

To summarize: Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or
- finds a $1/3 - 2/3$–separator S of G of size $O(\frac{n}{\ell} + \ell h \log n)$.

For large values of ℓ the above can be generalized to finding a minor, or finding a $1/2 - 1/2$-separator.

A More General Minor-Separator Theorem

To summarize: Given graphs G, H and some $\ell \geq 1$, there is a polynomial time algorithm which

- either finds an H-minor of G of size $O(\ell h \log n)$, where $h = |E(H)|$, or
- finds a $1/3 - 2/3$–separator S of G of size $O(\frac{n}{\ell} + \ell h \log n)$.

For large values of ℓ the above can be generalized to finding a minor, or finding a $1/2 - 1/2$-separator.

As in the case of Path-Separator we can obtain a separator tree, or an H-minor.

Minimum Bandwidth

Problem: Given a graph G, give a permutation π on the vertices of G so that the maximum edge *stretch* $\max_{(i,j)\in E(G)} |\pi(i) - \pi(j)|$ is minimized.

Best approximation: $O(\log^3 n\sqrt{\log\log n})$ by Dunagan and Vempala, 2001, $O(\sqrt{\frac{n}{B}}\log n)$ by Avrim Blum et al. where B is the optimum bandwidth

Best Exact Algorithm: $\tilde{O}(10^n)$ by Feige and Killian, 2000

Expanders

Expanders

For a subset $S \subseteq V$, let $N(S)$ be S's neighbors.

Expanders

For a subset $S \subseteq V$, let $N(S)$ be S's neighbors.

$G = (V, E)$ is an ε-expander iff for every $S \subseteq V$ with $|S| \leq |V|/2$ we have $|S \cup N(S)| \geq (1 + \varepsilon)|S|$.

That is, S's set of neighbors consists of a **constant fraction** of new nodes.

Expanders

For a subset $S \subseteq V$, let $N(S)$ be S's neighbors.

$G = (V, E)$ is an ε-expander iff for every $S \subseteq V$ with $|S| \leq |V|/2$ we have $|S \cup N(S)| \geq (1 + \varepsilon)|S|$.

That is, S's set of neighbors consists of a **constant fraction** of new nodes.

A graph is d-regular if all its vertices have degree d.

Expanders

For a subset $S \subseteq V$, let $N(S)$ be S's neighbors.

$G = (V, E)$ is an ε-expander iff for every $S \subseteq V$ with $|S| \leq |V|/2$ we have $|S \cup N(S)| \geq (1 + \varepsilon)|S|$.

That is, S's set of neighbors consists of a **constant fraction** of new nodes.

A graph is d-regular if all its vertices have degree d.

Gabber and Galil show how to construct 5–regular $(\frac{2-\sqrt{3}}{4})$–expanders efficiently.

Graphs with Expander Minors have Large Bandwidth

Graphs with Expander Minors have Large Bandwidth

Lemma. Let H be an ε-expander on h nodes for some constant $\varepsilon > 0$. Let G contain an H-minor M. Then the **minimum bandwidth of** G is at least $\Omega(h)$.

Graphs with Expander Minors have Large Bandwidth

Lemma. Let H be an ε-expander on h nodes for some constant $\varepsilon > 0$. Let G contain an H-minor M. Then the **minimum bandwidth of G** is at least $\Omega(h)$.

Pf. Let $|V(M)| = k$.

Graphs with Expander Minors have Large Bandwidth

Lemma. Let H be an ε-expander on h nodes for some constant $\varepsilon > 0$. Let G contain an H-minor M. Then the **minimum bandwidth of G** is at least $\Omega(h)$.

Pf. Let $|V(M)| = k$.

Let π be a linear arrangement of the nodes of M.

Graphs with Expander Minors have Large Bandwidth

Lemma. Let H be an ε-expander on h nodes for some constant $\varepsilon > 0$. Let G contain an H-minor M. Then the **minimum bandwidth of G** is at least $\Omega(h)$.

Pf. Let $|V(M)| = k$.

Let π be a linear arrangement of the nodes of M.

Let h_{LHS} and h_{RHS} be the *number* of supernodes completely contained among the first $k/2$ nodes (respectively, last $k/2$ nodes) in π.

Graphs with Expander Minors have Large Bandwidth

Lemma. Let H be an ε-expander on h nodes for some constant $\varepsilon > 0$. Let G contain an H-minor M. Then the **minimum bandwidth of G** is at least $\Omega(h)$.

Pf. Let $|V(M)| = k$.

Let π be a linear arrangement of the nodes of M.

Let h_{LHS} and h_{RHS} be the *number* of supernodes completely contained among the first $k/2$ nodes (respectively, last $k/2$ nodes) in π.

Let $h_S = h - h_{LHS} - h_{RHS}$.

Lemma Proof cont.

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon \cdot h$ nodes in the first half that connect to distinct nodes in the second half.

Any arrangement with this property has bandwidth at least $\varepsilon \cdot h$.

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon \cdot h$ nodes in the first half that connect to distinct nodes in the second half.

Any arrangement with this property has bandwidth at least $\varepsilon \cdot h$.

If $h_{LHS} < h/3$ or $h_{RHS} < h/3$ then $h_S \geq 2h/3$, so the bandwidth is $\Omega(h)$ in this case.

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon \cdot h$ nodes in the first half that connect to distinct nodes in the second half.

Any arrangement with this property has bandwidth at least $\varepsilon \cdot h$.

If $h_{LHS} < h/3$ or $h_{RHS} < h/3$ then $h_S \geq 2h/3$, so the bandwidth is $\Omega(h)$ in this case.

If $h_{LHS} \geq h/3$, then the supernodes contained in the first half have at least $\varepsilon h/3$ supernodes as neighbors, by the *expansion condition*. Thus either

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon \cdot h$ nodes in the first half that connect to distinct nodes in the second half.

Any arrangement with this property has bandwidth at least $\varepsilon \cdot h$.

If $h_{LHS} < h/3$ or $h_{RHS} < h/3$ then $h_S \geq 2h/3$, so the bandwidth is $\Omega(h)$ in this case.

If $h_{LHS} \geq h/3$, then the supernodes contained in the first half have at least $\varepsilon h/3$ supernodes as neighbors, by the *expansion condition*. Thus either

- $h_S \geq \varepsilon h/6$, which by the above implies the bandwidth is at least $\varepsilon \cdot h/6$, or

Lemma Proof cont.

If $h_S \geq \varepsilon \cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon \cdot h$:

Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon \cdot h$ nodes in the first half that connect to distinct nodes in the second half.

Any arrangement with this property has bandwidth at least $\varepsilon \cdot h$.

If $h_{LHS} < h/3$ or $h_{RHS} < h/3$ then $h_S \geq 2h/3$, so the bandwidth is $\Omega(h)$ in this case.

If $h_{LHS} \geq h/3$, then the supernodes contained in the first half have at least $\varepsilon h/3$ supernodes as neighbors, by the *expansion condition*. Thus either

- $h_S \geq \varepsilon h/6$, which by the above implies the bandwidth is at least $\varepsilon \cdot h/6$, or
- there are at least $\varepsilon h/6$ first half neighbors in the second half, in which case there are $\varepsilon h/6$ edges crossing from nodes in the first half to *distinct* nodes in the second half, so again the bandwidth is at least $\Omega(h)$.

Hybrid Algorithm Idea

Hybrid Algorithm Idea

- Either find a large constant degree expander as a minor of G.

  This guarantees that the bandwidth of G is large, and hence the $O(\sqrt{\frac{n}{B}} \log n)$–approximation algorithm by Avrim et al. gives a good approximation.

Hybrid Algorithm Idea

- Either find a large constant degree expander as a minor of G.

  This guarantees that the bandwidth of G is large, and hence the $O(\sqrt{\frac{n}{B}} \log n)$–approximation algorithm by Avrim et al. gives a good approximation.

- Otherwise use the separator tree to get a good exact algorithm for bandwidth.

How to use the separator tree to solve Minimum Bandwidth

How to use the separator tree to solve Minimum Bandwidth

At each separator node we specify:

How to use the separator tree to solve Minimum Bandwidth

At each separator node we specify:

- a $\log n$ bit index for the position of each separator node in the current allowed set of indices,

How to use the separator tree to solve Minimum Bandwidth

At each separator node we specify:

- a $\log n$ bit index for the position of each separator node in the current allowed set of indices,
- a length n bit string specifying whether left or right subtree nodes go at the corresponding position. We recurse on the left and right subtree separately, using the positions specified by the corresponding bits.

How to use the separator tree to solve Minimum Bandwidth, cont.

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,
- a node from the left subtree in position 6, a node from the right subtree in position 9.

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,
- a node from the left subtree in position 6, a node from the right subtree in position 9.
- The recursive call is for position 6 on the left and position 9 on the right.

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,
- a node from the left subtree in position 6, a node from the right subtree in position 9.
- The recursive call is for position 6 on the left and position 9 on the right.

At the end, the best linear arrangement is returned.

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,
- a node from the left subtree in position 6, a node from the right subtree in position 9.
- The recursive call is for position 6 on the left and position 9 on the right.

At the end, the best linear arrangement is returned.

The recurrence for the running time is (assuming a 1/2-1/2-separator):

$$T(n) \leq 2^{n+\ell \log n} \cdot 2T(n/2) + poly(n)$$

How to use the separator tree to solve Minimum Bandwidth, cont.

For example, for $\ell = 3$, $n = 5$, we may specify $(0, 2, 4)$ and (00111). If the allowed positions are $3, 6, 7, 9, 10$, then

- the first, second and third separator nodes are in positions $3, 7,$ and 10 respectively,
- a node from the left subtree in position 6, a node from the right subtree in position 9.
- The recursive call is for position 6 on the left and position 9 on the right.

At the end, the best linear arrangement is returned.

The recurrence for the running time is (assuming a 1/2-1/2-separator):

$$T(n) \leq 2^{n+\ell \log n} \cdot 2T(n/2) + poly(n)$$

$T(n) = \tilde{O}(4^n \cdot n^{\ell \log(n/\ell)})$, and if ℓ is chosen to be small, say $o(\frac{n}{(\log n \log \log n)})$, $T(n) = 4^{n+o(n)}$.

Conclusion

Conclusion

We introduced *hybrid algorithms*.

Conclusion

We introduced *hybrid algorithms*.

We gave a hybrid algorithm for Longest Path which either finds a path of length ℓ, or solves the problem exactly in time $2^{\ell \log L \log \frac{n}{\ell}}$.

Conclusion

We introduced *hybrid algorithms*.

We gave a hybrid algorithm for Longest Path which either finds a path of length ℓ, or solves the problem exactly in time $2^{\ell \log L \log \frac{n}{\ell}}$.

For $\ell = o(\frac{n}{\log n \log \log n})$ we obtain either a $\log n \log \log n$ approximation, or a subexponential $2^{o(n)}$ exact solution. This beats the known conventional algorithms on both accounts. It also beats the inapproximability ($2^{O(\frac{\log n}{\log \log n})}$) by a *huge* margin.

Conclusion

We introduced *hybrid algorithms*.

We gave a hybrid algorithm for Longest Path which either finds a path of length ℓ, or solves the problem exactly in time $2^{\ell \log L \log \frac{n}{\ell}}$.

For $\ell = o(\frac{n}{\log n \log \log n})$ we obtain either a $\log n \log \log n$ approximation, or a subexponential $2^{o(n)}$ exact solution. This beats the known conventional algorithms on both accounts. It also beats the inapproximability ($2^{O(\frac{\log n}{\log \log n})}$) by a *huge* margin.

We gave a hybrid algorithm for Minimum Bandwidth which either approximates within $\alpha(n) \log^{2.5} n \log \log n$ (for unbounded $\alpha(n)$) or solves exactly in $4^{n+o(n)}$ time. This also beats the best known conventional algorithms on both accounts.

Thank You!