Strongly Polynomial Algorithms for Some Parametric Global Minimum Cut Problems

Hassène Aissi / S. Thomas McCormick / Maurice Queyranne

Paris-Dauphine / Sauder School of Business, UBC $\times 2$

BIRS TSP Sept 2018

S. Thomas McCormick
Sauder School of Business
University of British Columbia

Outline

1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems

Outline

1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems
2. Faster Algorithms for P_{NB}
   - Deterministic
   - Randomized

Outline

1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems
2. Faster Algorithms for P_{NB}
   - Deterministic
   - Randomized
3. Faster Algorithms for $P_{\max}$
   - Deterministic
   - Randomized

Outline

Outline

1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems
2. Faster Algorithms for P_{NB}
   - Deterministic
   - Randomized
3. Faster Algorithms for $P_{\max}$
   - Deterministic
   - Randomized
4. Conclusion

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.
- A (global) cut is induced by node subset $\emptyset \subset C \subset V$, and includes edges $\delta(C) = \{e \in E \mid \text{exactly one end of } e \text{ is in } C\}$.

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.
- A (global) cut is induced by node subset $\emptyset \subset C \subset V$, and includes edges $\delta(C) = \{e \in E \mid \text{exactly one end of } e \text{ is in } C\}$.
- Then a global min cut C^* satisfies $c(\delta(C^*)) = \min_{\text{cuts } C} c(\delta(C))$.

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.
- A (global) cut is induced by node subset $\emptyset \subset C \subset V$, and includes edges $\delta(C) = \{e \in E \mid \text{exactly one end of } e \text{ is in } C\}$.
- Then a global min cut C^* satisfies $c(\delta(C^*)) = \min_{\text{cuts } C} c(\delta(C))$.
- Can compute a global min cut in $O(mn + n^2 \log n)$ deterministic time (Stoer-Wagner = SW, Nagamochi-Ibaraki = NI), or $\tilde{O}(n^2)$ randomized time (Karger-Stein = KS), or $\tilde{O}(m)$ randomized time (Karger = K).

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.
- A (global) cut is induced by node subset $\emptyset \subset C \subset V$, and includes edges $\delta(C) = \{e \in E \mid \text{exactly one end of } e \text{ is in } C\}$.
- Then a global min cut C^* satisfies $c(\delta(C^*)) = \min_{\text{cuts } C} c(\delta(C))$.
- Can compute a global min cut in $O(mn + n^2 \log n)$ deterministic time (Stoer-Wagner = SW, Nagamochi-Ibaraki = NI), or $\tilde{O}(n^2)$ randomized time (Karger-Stein = KS), or $\tilde{O}(m)$ randomized time (Karger = K).
- There are only $O(n^{\lfloor 2\alpha \rfloor})$ α-approximate min cuts; when $\alpha < \frac{4}{3}$ they can all be computed in $O(n^4)$ deterministic time (NI), or $\tilde{O}(n^{\lfloor 2\alpha \rfloor}) = \tilde{O}(n^2)$ randomized time (KS).

Global Min Cut

- We are given an undirected graph $G = (V, E)$ with non-negative distances (costs) $c_e \in \mathbb{R}^E$.
  - Set $m = |E|$, $n = |V|$ as usual.
- A (global) cut is induced by node subset $\emptyset \subset C \subset V$, and includes edges $\delta(C) = \{e \in E \mid \text{exactly one end of } e \text{ is in } C\}$.
- Then a global min cut C^* satisfies $c(\delta(C^*)) = \min_{\text{cuts } C} c(\delta(C))$.
- Can compute a global min cut in $O(mn + n^2 \log n)$ deterministic time (Stoer-Wagner = SW, Nagamochi-Ibaraki = NI), or $\tilde{O}(n^2)$ randomized time (Karger-Stein = KS), or $\tilde{O}(m)$ randomized time (Karger = K).
- There are only $O(n^{\lfloor 2\alpha \rfloor})$ α-approximate min cuts; when $\alpha < \frac{4}{3}$ they can all be computed in $O(n^4)$ deterministic time (NI), or $\tilde{O}(n^{\lfloor 2\alpha \rfloor}) = \tilde{O}(n^2)$ randomized time (KS).
- A vitally important subproblem in separating TSP facets.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions $c^0, \ldots, c^d$, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^d \mu_i c^i(e)$.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions $c^0, \ldots, c^d$, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^{d} \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions $c^0, \ldots, c^d$, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^d \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.
  - To avoid NP Hardness, we instead assume that μ is restricted to $M = \{\mu \in \mathbb{R}^d \mid c_\mu(e) \geq 0 \ \forall\ e \in E\}$.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions c^0, ..., c^d, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^d \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.
  - To avoid NP Hardness, we instead assume that μ is restricted to $M = \{\mu \in \mathbb{R}^d \mid c_\mu(e) \geq 0 \ \forall \ e \in E\}$.
- Why is parametric global min cut interesting?

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions c^0, ..., c^d, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^{d} \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.
  - To avoid NP Hardness, we instead assume that μ is restricted to $M = \{\mu \in \mathbb{R}^d \mid c_\mu(e) \geq 0 \ \forall\ e \in E\}$.
- Why is parametric global min cut interesting?
  - Models “attack-defend” graph problems where a Defender spends a fixed budget on d resources to reinforce edges against an Attacker.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions c^0, ..., c^d, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^{d} \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.
  - To avoid NP Hardness, we instead assume that μ is restricted to $M = \{\mu \in \mathbb{R}^d \mid c_\mu(e) \geq 0 \ \forall\ e \in E\}$.
- Why is parametric global min cut interesting?
  - Models "attack-defend" graph problems where a Defender spends a fixed budget on d resources to reinforce edges against an Attacker.
  - Models situations where costs can change due to external variables.

Parametric Global Min Cut

- Now suppose that edge costs are linear functions of d parameters $\mu \in \mathbb{R}^d$.
- Thus we have $d+1$ edge cost functions $c^0, \ldots, c^d$, and the cost of $e \in E$ w.r.t. μ is $c_\mu(e) = c^0(e) + \sum_{i=1}^d \mu_i c^i(e)$.
  - We do *not* assume that all $c^i(e)$ are non-negative.
  - To avoid NP Hardness, we instead assume that μ is restricted to $M = \{\mu \in \mathbb{R}^d \mid c_\mu(e) \geq 0 \ \forall\ e \in E\}$.
- Why is parametric global min cut interesting?
  - Models "attack-defend" graph problems where a Defender spends a fixed budget on d resources to reinforce edges against an Attacker.
  - Models situations where costs can change due to external variables.
  - It will turn out to further highlight how the small number of α-approximate solutions leads to more efficient algorithms.

The Global Min Cut Value Function

- Define $Z(\mu)$ to be the cost of a global min cut at μ.

The Global Min Cut Value Function

- Define $Z(\mu)$ to be the cost of a global min cut at μ.
  - Since $Z(\mu)$ is the min of many affine functions (one for each cut), it is a piecewise-linear concave function.

The Global Min Cut Value Function

- Define $Z(\mu)$ to be the cost of a global min cut at μ.
  - Since $Z(\mu)$ is the min of many affine functions (one for each cut), it is a piecewise-linear concave function.
  - AMMQ '15 showed that the number of facets of $Z(\mu)$ is $O(m^d n^2 \log^{d-1} n)$ and they can be computed in $O(m^{d\lfloor \frac{d-1}{2} \rfloor} n^{2\lfloor \frac{d-1}{2} \rfloor} \log^{(d-1)\lfloor \frac{d-1}{2} \rfloor + O(1)} n)$ deterministic time, and $O(mn^4 \log n + n^5 \log^2 n)$ when $d = 1$.

The Global Min Cut Value Function

- Define $Z(\mu)$ to be the cost of a global min cut at μ.
  - Since $Z(\mu)$ is the min of many affine functions (one for each cut), it is a piecewise-linear concave function.
  - AMMQ '15 showed that the number of facets of $Z(\mu)$ is $O(m^d n^2 \log^{d-1} n)$ and they can be computed in $O(m^{d\lfloor\frac{d-1}{2}\rfloor} n^{2\lfloor\frac{d-1}{2}\rfloor} \log^{(d-1)\lfloor\frac{d-1}{2}\rfloor+O(1)} n)$ deterministic time, and $O(mn^4 \log n + n^5 \log^2 n)$ when $d = 1$.
  - When all $c^i(e) \geq 0$, Karger '16 improved this to show that the number of facets of $Z(\mu)$ is $O(n^{d+2})$, and they can be computed in $O(n^{2d+2} \log n)$ randomized time.

The Global Min Cut Value Function

- Define $Z(\mu)$ to be the cost of a global min cut at μ.
  - Since $Z(\mu)$ is the min of many affine functions (one for each cut), it is a piecewise-linear concave function.
  - AMMQ '15 showed that the number of facets of $Z(\mu)$ is $O(m^d n^2 \log^{d-1} n)$ and they can be computed in $O(m^{d\lfloor\frac{d-1}{2}\rfloor} n^{2\lfloor\frac{d-1}{2}\rfloor} \log^{(d-1)\lfloor\frac{d-1}{2}\rfloor+O(1)} n)$ deterministic time, and $O(mn^4 \log n + n^5 \log^2 n)$ when $d = 1$.
  - When all $c^i(e) \geq 0$, Karger '16 improved this to show that the number of facets of $Z(\mu)$ is $O(n^{d+2})$, and they can be computed in $O(n^{2d+2} \log n)$ randomized time.
- Computing all of $Z(\mu)$ is good, but is maybe too much?

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_\mu Z(\mu)$.

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_\mu Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_\mu Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).
- In other applications (e.g. sensitivity analysis) we want to solve P_{NB}: Given $\mu^0 \in \mathbb{R}^d$ and direction $\nu \in \mathbb{R}^d$, find the next *breakpoint* of $Z(\mu)$ along the ray starting at μ^0 in direction ν.

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_\mu Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).
- In other applications (e.g. sensitivity analysis) we want to solve P_{NB}: Given $\mu^0 \in \mathbb{R}^d$ and direction $\nu \in \mathbb{R}^d$, find the next *breakpoint* of $Z(\mu)$ along the ray starting at μ^0 in direction ν.
  - P_{NB} is a sort of *ray-shooting* problem.

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_{\mu} Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).
- In other applications (e.g. sensitivity analysis) we want to solve P_{NB}: Given $\mu^0 \in \mathbb{R}^d$ and direction $\nu \in \mathbb{R}^d$, find the next *breakpoint* of $Z(\mu)$ along the ray starting at μ^0 in direction ν.
  - P_{NB} is a sort of *ray-shooting* problem.
  - P_{NB} is effectively a 1-parameter problem, to find the next breakpoint w.r.t. costs $\bar{c}^0 + \lambda \bar{c}^1(e)$ with single parameter λ.

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_\mu Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).
- In other applications (e.g. sensitivity analysis) we want to solve P_{NB}: Given $\mu^0 \in \mathbb{R}^d$ and direction $\nu \in \mathbb{R}^d$, find the next *breakpoint* of $Z(\mu)$ along the ray starting at μ^0 in direction ν.
  - P_{NB} is a sort of *ray-shooting* problem.
  - P_{NB} is effectively a 1-parameter problem, to find the next breakpoint w.r.t. costs $\bar{c}^0 + \lambda\bar{c}^1(e)$ with single parameter λ.
- We could solve $P_{\max}$ and P_{NB} by computing $Z(\mu)$, but we want to find something faster.

Defining the Parametric Problems

- Computing all of $Z(\mu)$ is good, but is maybe too much?
- E.g., for attack-defend the Attacker only wants to solve $\max_{\mu} Z(\mu)$.
- So define $P_{\max}$ to be the problem of computing the max over μ of $Z(\mu)$ (and an associated global min cut).
- In other applications (e.g. sensitivity analysis) we want to solve P_{NB}: Given $\mu^0 \in \mathbb{R}^d$ and direction $\nu \in \mathbb{R}^d$, find the next *breakpoint* of $Z(\mu)$ along the ray starting at μ^0 in direction ν.
  - P_{NB} is a sort of *ray-shooting* problem.
  - P_{NB} is effectively a 1-parameter problem, to find the next breakpoint w.r.t. costs $\bar{c}^0 + \lambda\bar{c}^1(e)$ with single parameter λ.
- We could solve $P_{\max}$ and P_{NB} by computing $Z(\mu)$, but we want to find something faster.
- We also want to see if one is harder than the other.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.
  - Here "linear" means that every comparison is between two affine functions of μ and the data.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.
  - Here "linear" means that every comparison is between two affine functions of μ and the data.
- We show that SW is linear, so Megiddo+SW gives an $O(n^{2d+3}\log^d n)$ deterministic algorithm for $P_{\max}$, and $O(n^5 \log d)$ for P_{NB}.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.
  - Here "linear" means that every comparison is between two affine functions of μ and the data.
- We show that SW is linear, so Megiddo+SW gives an $O(n^{2d+3} \log^d n)$ deterministic algorithm for $P_{\max}$, and $O(n^5 \log d)$ for P_{NB}.
- Tokuyama saw that KS is linear, so Megiddo+KS gives an $O(n^2 \log^{4d+1} n)$ randomized algorithm for $P_{\max}$, and $O(n^2 \log^5 n)$ for P_{NB}.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.
  - Here "linear" means that every comparison is between two affine functions of μ and the data.
- We show that SW is linear, so Megiddo+SW gives an $O(n^{2d+3}\log^d n)$ deterministic algorithm for $P_{\max}$, and $O(n^5 \log d)$ for P_{NB}.
- Tokuyama saw that KS is linear, so Megiddo+KS gives an $O(n^2 \log^{4d+1} n)$ randomized algorithm for $P_{\max}$, and $O(n^2 \log^5 n)$ for P_{NB}.
- These are a lot faster than the $O(m^{d\lfloor\frac{d-1}{2}\rfloor} n^{2\lfloor\frac{d-1}{2}\rfloor} \log^{(d-1)\lfloor\frac{d-1}{2}\rfloor+O(1)} n)$ deterministic and $O(n^{2d+2}\log n)$ randomized algorithms for computing all of $Z(\mu)$.

Megiddo's Parametric Framework

- Megiddo, later with Cohen, gave a black-box way to adapt linear algorithms for non-parametric problems to solve parametric problems.
  - Here "linear" means that every comparison is between two affine functions of μ and the data.
- We show that SW is linear, so Megiddo+SW gives an $O(n^{2d+3}\log^d n)$ deterministic algorithm for $P_{\max}$, and $O(n^5 \log d)$ for P_{NB}.
- Tokuyama saw that KS is linear, so Megiddo+KS gives an $O(n^2 \log^{4d+1} n)$ randomized algorithm for $P_{\max}$, and $O(n^2 \log^5 n)$ for P_{NB}.
- These are a lot faster than the $O(m^{d\lfloor \frac{d-1}{2} \rfloor} n^{2\lfloor \frac{d-1}{2} \rfloor} \log^{(d-1)\lfloor \frac{d-1}{2} \rfloor + O(1)} n)$ deterministic and $O(n^{2d+2}\log n)$ randomized algorithms for computing all of $Z(\mu)$.
- However, we'd still like to do better than generic Megiddo.

Summary of Where We Are

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K

Summary of running times so far.

Summary of Where We Are

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K

Big gap between non-parametric and computing all of $Z(\mu)$ running times, even for $d = 1$

Summary of Where We Are

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K

Much smaller gap between non-parametric and Megiddo running times (compare to $Z(\mu)$ times in blue); for $d = 1$, KS gap is just logs. Note that using Megiddo to solve P_{NB} is just general Megiddo with d set to 1.

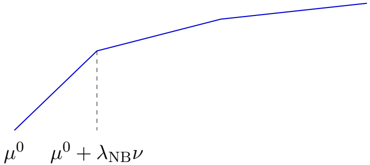
Summary of Where We Are

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	???	???
$P_{\max}$ ($\sim$ gen'l d)	???	???

Hoped-for results in this paper in red. Compare to non-param lower bounds in green, various upper bounds in blue.

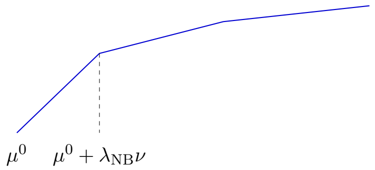
Reducing P_{NB} to $P_{\max}$ with $d = 1$

- P_{NB} wants us to compute λ_{NB}:

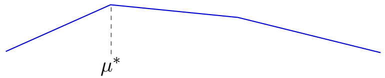


Reducing P_{NB} to $P_{\max}$ with $d = 1$

- P_{NB} wants us to compute λ_{NB}:

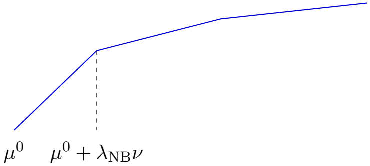


- If we rotate until the local slope at μ^0 is just short of horizontal, then finding λ_{NB} becomes equivalent to computing μ^* in this 1-dimensional problem:

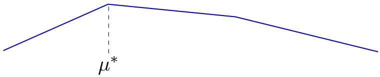


Reducing P_{NB} to $P_{\max}$ with $d = 1$

- P_{NB} wants us to compute λ_{NB}:



- If we rotate until the local slope at μ^0 is just short of horizontal, then finding λ_{NB} becomes equivalent to computing μ^* in this 1-dimensional problem:



- Thus P_{NB} cannot be any harder than $P_{\max}$ for $d = 1$, though it could be easier.

Outline

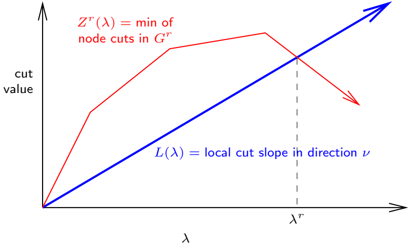
1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems
2. Faster Algorithms for P_{NB}
   - Deterministic
   - Randomized
3. Faster Algorithms for $P_{\max}$
   - Deterministic
   - Randomized
4. Conclusion

Using Stoer-Wagner to Solve P_{NB}

- SW finds a node ordering $v_1, \ldots, v_n$ such that (v_{n-1}, v_n) is a pendent pair, i.e., either $\delta(v_n)$ is a global min cut, or we can contract edge $\{v_{n-1}, v_n\}$ without losing any optimal cuts.

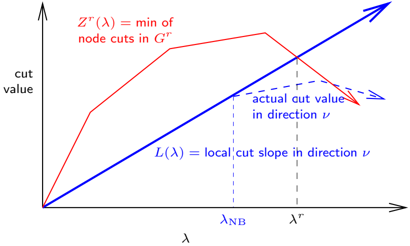
Using Stoer-Wagner to Solve P_{NB}

- SW finds a node ordering $v_1, \ldots, v_n$ such that (v_{n-1}, v_n) is a pendent pair, i.e., either $\delta(v_n)$ is a global min cut, or we can contract edge $\{v_{n-1}, v_n\}$ without losing any optimal cuts.
- Let G^r be contracted graph at iteration r. Define $Z^r(\lambda)$ to be min of $\bar{c}(\delta(v))$ for $v \in V^r$ and compute λ^r like:



Using Stoer-Wagner to Solve P_{NB}

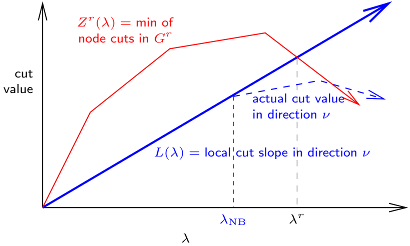
- SW finds a node ordering $v_1, \ldots, v_n$ such that (v_{n-1}, v_n) is a pendent pair, i.e., either $\delta(v_n)$ is a global min cut, or we can contract edge $\{v_{n-1}, v_n\}$ without losing any optimal cuts.
- Let G^r be contracted graph at iteration r. Define $Z^r(\lambda)$ to be min of $\bar{c}(\delta(v))$ for $v \in V^r$ and compute λ^r like:



- Update an UB $\bar{\lambda}$ on λ_{NB} by λ_r, and do SW to find and contract a pendent pair w.r.t. $\bar{\lambda}$; since $Z(\lambda)$ is concave, λ^r upper bounds λ_{NB}.

Using Stoer-Wagner to Solve P_{NB}

- SW finds a node ordering $v_1, \ldots, v_n$ such that (v_{n-1}, v_n) is a pendent pair, i.e., either $\delta(v_n)$ is a global min cut, or we can contract edge $\{v_{n-1}, v_n\}$ without losing any optimal cuts.
- Let G^r be contracted graph at iteration r. Define $Z^r(\lambda)$ to be min of $\bar{c}(\delta(v))$ for $v \in V^r$ and compute λ^r like:



- Update an UB $\bar{\lambda}$ on λ_{NB} by λ_r, and do SW to find and contract a pendent pair w.r.t. $\bar{\lambda}$; since $Z(\lambda)$ is concave, λ^r upper bounds λ_{NB}.
- This is correct, and runs in same $O(mn + n^2 \log n)$ time as SW.

Summary of Running Times

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	SW $O(mn + n^2 \log n)$	???
$P_{\max}$ ($\sim$ gen'l d)	???	???

Here we saved a lot w.r.t. Megiddo, and matched the non-parametric lower bound.

Using Karger-Stein to Solve P_{NB}

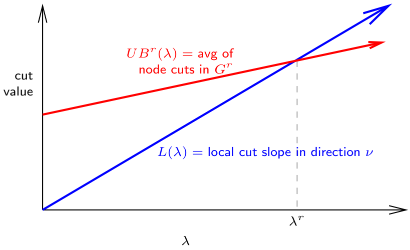
- KS selects an edge to contract randomly, proportional to its cost.

Using Karger-Stein to Solve P_{NB}

- KS selects an edge to contract randomly, proportional to its cost.
- After contracting to 2 nodes, KS show that the remaining induced cut is a min cut with probability at least $1/\binom{n}{2}$, and this can be put into a framework that will identify a min cut with high probability.

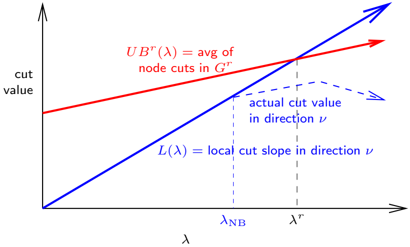
Using Karger-Stein to Solve P_{NB}

- KS selects an edge to contract randomly, proportional to its cost.
- After contracting to 2 nodes, KS show that the remaining induced cut is a min cut with probability at least $1/\binom{n}{2}$, and this can be put into a framework that will identify a min cut with high probability.
- Compute λ^r like this:



Using Karger-Stein to Solve P_{NB}

- KS selects an edge to contract randomly, proportional to its cost.
- After contracting to 2 nodes, KS show that the remaining induced cut is a min cut with probability at least $1/\binom{n}{2}$, and this can be put into a framework that will identify a min cut with high probability.
- Compute λ^r like this:



- Choose e to contract with probability proportional to $c_{\lambda^r}(e)$; since $Z(\lambda)$ is concave, λ^r upper bounds λ_{NB}.

Using Karger-Stein to Solve P_{NB}

- Compute λ_{NB} as the intersection of the final cut line and $L(\lambda)$, and repeat in the KS framework.

Using Karger-Stein to Solve P_{NB}

- Compute λ_{NB} as the intersection of the final cut line and $L(\lambda)$, and repeat in the KS framework.
- As in KS, when λ_{NB} exists, the probability that a global min cut survives all the contractions is at least $1/\binom{n}{2}$; if there is no breakpoint in direction ν, then the algorithm recognizes this with probability one.

Using Karger-Stein to Solve $P_{\rm NB}$

- Compute $\lambda_{\rm NB}$ as the intersection of the final cut line and $L(\lambda)$, and repeat in the KS framework.
- As in KS, when $\lambda_{\rm NB}$ exists, the probability that a global min cut survives all the contractions is at least $1/\binom{n}{2}$; if there is no breakpoint in direction ν, then the algorithm recognizes this with probability one.
- Thus using the KS framework is correct, and runs in same $\tilde{O}(n^2)$ time as KS.

Using Karger-Stein to Solve P_{NB}

- Compute λ_{NB} as the intersection of the final cut line and $L(\lambda)$, and repeat in the KS framework.
- As in KS, when λ_{NB} exists, the probability that a global min cut survives all the contractions is at least $1/\binom{n}{2}$; if there is no breakpoint in direction ν, then the algorithm recognizes this with probability one.
- Thus using the KS framework is correct, and runs in same $\tilde{O}(n^2)$ time as KS.
- There is a minor technical point about how to implement the random edge contractions: Here the parametric costs interfere with the KS matrix update technique, but we can replace the static matrices with separate matrices for $\vec{c}^0$ and $\vec{c}^1$ to achieve the same effect.

Summary of Running Times

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	SW $O(mn + n^2 \log n)$	KS $O(n^2 \log^3 n)$
$P_{\max}$ ($\sim$ gen'l d)	???	???

Here we saved only log factors w.r.t. Megiddo, but that's all the gap we had to work with; our ideas don't seem to extend to Karger's improvement.

Outline

1. Global Min Cut
   - Non-Parametric
   - Parametric
   - The Parametric Problems
2. Faster Algorithms for P_{NB}
   - Deterministic
   - Randomized
3. Faster Algorithms for $P_{\max}$
   - Deterministic
   - Randomized
4. Conclusion

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.
- In PLA we are given:

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
    1. Approximate duality between global MC and max spanning tree.
    2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.
- In PLA we are given:
    - a set $\mathcal{H}$ of hyperplanes (think the μ s.t. $c_\mu(e) = c_\mu(e')$);

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.
- In PLA we are given:
  - a set $\mathcal{H}$ of hyperplanes (think the μ s.t. $c_\mu(e) = c_\mu(e')$);
  - a polytope P (think the region M where all $c_\mu(e) \geq 0$); and

Solving $P_{\max}$: Overview and Techniques

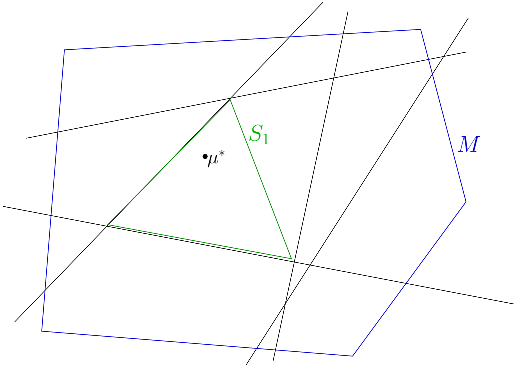
- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.
- In PLA we are given:
  - a set $\mathcal{H}$ of hyperplanes (think the μ s.t. $c_\mu(e) = c_\mu(e')$);
  - a polytope P (think the region M where all $c_\mu(e) \geq 0$); and
  - an unknown target (think μ^*).

Solving $P_{\max}$: Overview and Techniques

- Following Mulmuley and AMMQ we want to use two ideas to compute $\mu^* = \max_\mu Z(\mu)$:
  1. Approximate duality between global MC and max spanning tree.
  2. Ability to compute all $O(n^2)$ α-approximate solutions for $\alpha < \frac{4}{3}$.
- But max spanning tree makes sense only when all costs are linearly ordered, and parametric costs typically are not.
- And we need to narrow down our search for μ^* to a region small enough that the α-approximate min cuts include all cuts defining μ^*.
- We use a technique from computational geometry called point location in arrangements (PLA) to achieve both of these.
- In PLA we are given:
  - a set $\mathcal{H}$ of hyperplanes (think the μ s.t. $c_\mu(e) = c_\mu(e')$);
  - a polytope P (think the region M where all $c_\mu(e) \geq 0$); and
  - an unknown target (think μ^*).
- Then the task is to find a simplex in a cell of $\mathcal{H} \cap P$ containing μ^*.

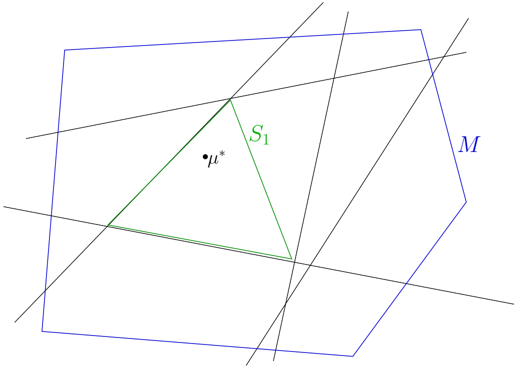
Weak Duality between GMC and Max Spanning Tree

- Define $\mathcal{H}_1$ as the set of $O(m^2)$ hyperplanes where $c_\mu(e) = c_\mu(e')$ and run PLA for $(\mathcal{H}_1, M, \mu^*)$ to get simplex S_1.



Weak Duality between GMC and Max Spanning Tree

- Define $\mathcal{H}_1$ as the set of $O(m^2)$ hyperplanes where $c_\mu(e) = c_\mu(e')$ and run PLA for $(\mathcal{H}_1, M, \mu^*)$ to get simplex S_1.



- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.
- Let $\bar{e}$ be a min-cost edge in T.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.
- Let $\bar{e}$ be a min-cost edge in T.
  - Since every cut hits T we get $Z(\mu^*) \geq c_\mu(\bar{e})$ for all $\mu \in S_1$.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.
- Let $\bar{e}$ be a min-cost edge in T.
  - Since every cut hits T we get $Z(\mu^*) \geq c_\mu(\bar{e})$ for all $\mu \in S_1$.
  - Let $\bar{C}$ be the fundamental cut in $T - \bar{e}$; since T is a MST we have $Z(\mu^*) \leq c_{\mu^*}(\bar{C}) \leq m c_{\mu^*}(\bar{e})$.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.
- Let $\bar{e}$ be a min-cost edge in T.
  - Since every cut hits T we get $Z(\mu^*) \geq c_\mu(\bar{e})$ for all $\mu \in S_1$.
  - Let $\bar{C}$ be the fundamental cut in $T - \bar{e}$; since T is a MST we have $Z(\mu^*) \leq c_{\mu^*}(\bar{C}) \leq mc_{\mu^*}(\bar{e})$.
  - Thus $c_{\mu^*}(\bar{e}) \leq Z(\mu^*) \leq mc_{\mu^*}(\bar{e})$, and so $c_{\mu^*}(\bar{e})$ is a fairly tight estimate of $Z(\mu^*)$.

Weak Duality between GMC and Max Spanning Tree

- By the definition of $\mathcal{H}_1$ and PLA, we know that $\mu^* \in S_1$ and all $c_\mu(e)$ are linearly ordered for $\mu \in S_1$.
- Thus we can compute a max spanning tree T in S_1.
- Let $\bar{e}$ be a min-cost edge in T.
  - Since every cut hits T we get $Z(\mu^*) \geq c_\mu(\bar{e})$ for all $\mu \in S_1$.
  - Let $\bar{C}$ be the fundamental cut in $T - \bar{e}$; since T is a MST we have $Z(\mu^*) \leq c_{\mu^*}(\bar{C}) \leq mc_{\mu^*}(\bar{e})$.
  - Thus $c_{\mu^*}(\bar{e}) \leq Z(\mu^*) \leq mc_{\mu^*}(\bar{e})$, and so $c_{\mu^*}(\bar{e})$ is a fairly tight estimate of $Z(\mu^*)$.
- Now we need to use PLA a second time to further narrow in on μ^* so we can get the cuts inducing it via α-approximate cuts.

Narrowing in on α-Approximate Cuts

- Choose

Narrowing in on α-Approximate Cuts

- Choose
  - $\bar{\alpha}$ s.t. $1 < \bar{\alpha} < \sqrt{\frac{4}{3}}$ (note: $0 < \frac{\bar{\alpha}^2 - 1}{m} < 1$);

Narrowing in on α-Approximate Cuts

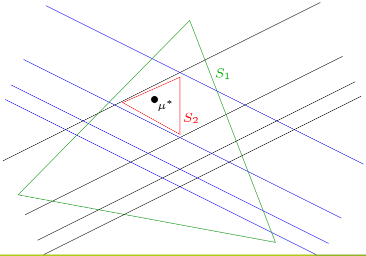
- Choose
  - $\bar{\alpha}$ s.t. $1 < \bar{\alpha} < \sqrt{\frac{4}{3}}$ (note: $0 < \frac{\bar{\alpha}^2-1}{m} < 1$);
  - $p = 1 + \lceil \log \frac{m^2}{\bar{\alpha}^2-1} / \log \bar{\alpha}^2 \rceil$ so that $\frac{\bar{\alpha}^2-1}{m}\bar{\alpha}^{2(p-1)} > m$ (note: $p = O(\log n)$);

Narrowing in on α-Approximate Cuts

- Choose
  - $\bar{\alpha}$ s.t. $1 < \bar{\alpha} < \sqrt{\frac{4}{3}}$ (note: $0 < \frac{\bar{\alpha}^2-1}{m} < 1$);
  - $p = 1 + \lceil \log \frac{m^2}{\bar{\alpha}^2-1} / \log \bar{\alpha}^2 \rceil$ so that $\frac{\bar{\alpha}^2-1}{m}\bar{\alpha}^{2(p-1)} > m$ (note: $p = O(\log n)$);
  - $g_i(\bar{e}, \mu) = \frac{\bar{\alpha}^2-1}{m}\bar{\alpha}^{2(i-1)} c_\mu(\bar{e})$ for $i = 1, \ldots, p$, $g_0(\bar{e}, \mu) = 0$ (note: $g_1(\bar{e}, \mu) < c_\mu(\bar{e})$ and $g_p(\bar{e}, \mu) > mc_\mu(\bar{e})$).

Narrowing in on α-Approximate Cuts

- Choose
  - $\bar{\alpha}$ s.t. $1 < \bar{\alpha} < \sqrt{\frac{4}{3}}$ (note: $0 < \frac{\bar{\alpha}^2-1}{m} < 1$);
  - $p = 1 + \lceil \log \frac{m^2}{\bar{\alpha}^2-1} / \log \bar{\alpha}^2 \rceil$ so that $\frac{\bar{\alpha}^2-1}{m}\bar{\alpha}^{2(p-1)} > m$ (note: $p = O(\log n)$);
  - $g_i(\bar{e}, \mu) = \frac{\bar{\alpha}^2-1}{m}\bar{\alpha}^{2(i-1)} c_\mu(\bar{e})$ for $i = 1, \dots, p$, $g_0(\bar{e}, \mu) = 0$ (note: $g_1(\bar{e}, \mu) < c_\mu(\bar{e})$ and $g_p(\bar{e}, \mu) > mc_\mu(\bar{e})$).
- Define $\mathcal{H}_2$ as the $O(m \log n)$ hyperplanes where $c_\mu(e) = g_i(\bar{e}, \mu)$, $\forall$ $e \in E$, $i = 1, \dots, p$, and set $S_2 = \mathrm{PLA}(\mathcal{H}_2, S_1, \mu^*)$:



Computing Min Cuts and μ^*

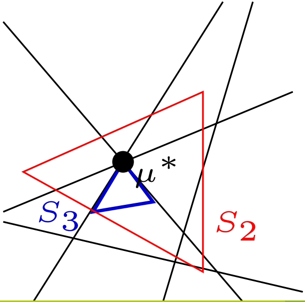
- Due to how we defined the $g_i(\bar{e}, \mu)$, we know that any cut defining μ^* must be an $\bar{\alpha}$-approximate cut for any $\mu \in S_2$.

Computing Min Cuts and μ^*

- Due to how we defined the $g_i(\bar{e}, \mu)$, we know that any cut defining μ^* must be an $\bar{\alpha}$-approximate cut for any $\mu \in S_2$.
- Thus we could compute the $O(n^2)$ $\bar{\alpha}$-approximate cuts in $\mathcal{C}$ and compute their lower envelope to get μ^*, but this would take $\Omega(n^{2d})$ time, too slow.

Computing Min Cuts and μ^*

- Due to how we defined the $g_i(\bar{e}, \mu)$, we know that any cut defining μ^* must be an $\bar{\alpha}$-approximate cut for any $\mu \in S_2$.
- Thus we could compute the $O(n^2)$ $\bar{\alpha}$-approximate cuts in $\mathcal{C}$ and compute their lower envelope to get μ^*, but this would take $\Omega(n^{2d})$ time, too slow.
- Instead, define $\mathcal{H}_3$ as the $O(n^4)$ hyperplanes where $c_\mu(C) = c_\mu(C')$ for C, $C' \in \mathcal{C}$ and set $S_3 = \mathrm{PLA}(\mathcal{H}_3, S_2, \mu^*)$.



Computing Min Cuts and μ^*

- Since μ^* is the intersection of d cuts in $\mathcal{C}$, it must be a vertex of S_3, and so this last call of PLA finds μ^* more efficiently.

Computing Min Cuts and μ^*

- Since μ^* is the intersection of d cuts in $\mathcal{C}$, it must be a vertex of S_3, and so this last call of PLA finds μ^* more efficiently.
- PLA is a recursive procedure; when we solve the recursion, we get the claimed $O(n^4 \log^{d-1} n)$ running time.

Computing Min Cuts and μ^*

- Since μ^* is the intersection of d cuts in $\mathcal{C}$, it must be a vertex of S_3, and so this last call of PLA finds μ^* more efficiently.
- PLA is a recursive procedure; when we solve the recursion, we get the claimed $O(n^4 \log^{d-1} n)$ running time.
- I skipped a technicality that arises when $c_\mu(\bar{e}) = 0$ for some $\mu \in S_1$.

Summary of Running Times

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	SW $O(mn + n^2 \log n)$	KS $O(n^2 \log^3 n)$
$P_{\max}$ ($\sim$ gen'l d)	$O(n^4 \log^{d-1} n)$	???

We saved a lot compared to Megiddo, but even for $d = 1$ still much slower than our deterministic P_{NB} algorithm, suggesting that $P_{\max}$ for $d = 1$ is strictly harder than P_{NB}.

Summary of Running Times

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	SW $O(mn + n^2 \log n)$	KS $O(n^2 \log^3 n)$
$P_{\max}$ ($\sim$ gen'l d)	$O(n^4 \log^{d-1} n)$	???

Notice that running time for our $P_{\max}$ algorithm is just log factors more than for computing all $\bar{\alpha}$-approximate min cuts.

Solving $P_{\max}$ Randomly

- So far we don't know how to do this ...

Final Summary of Running Times

Problem	Deterministic	Randomized
Non-param GMC	SW $O(mn + n^2 \log n)$	K $\tilde{O}(m)$ (KS $\tilde{O}(n^2)$)
All $\alpha < \frac{4}{3}$-approx	NI $O(n^4)$	KS $\tilde{O}(n^2)$
Megiddo $d = 1$	SW $O(n^5 \log n)$	KS $O(n^2 \log^5 n)$
Megiddo gen'l d	SW $O(n^{2d+3} \log^d n)$	KS $O(n^2 \log^{4d+1} n)$
$Z(\mu)$ $d = 1$	$O(mn^4 \log n + n^5 \log^2 n)$	$O(n^4 \log n)$ K
$Z(\mu)$ gen'l d	(big) AMMQ	$O(n^{2d+2} \log n)$ K
P_{NB} ($\sim d = 1$)	SW $O(mn + n^2 \log n)$	KS $O(n^2 \log^3 n)$
$P_{\max}$ ($\sim$ gen'l d)	$O(n^4 \log^{d-1} n)$	???

New results in this paper in red. Compare to non-param lower bounds in green, various upper bounds in blue.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.
  - The P_{NB} algorithms are essentially as fast as the non-parametric algorithms.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.
  - The P_{NB} algorithms are essentially as fast as the non-parametric algorithms.
  - The deterministic $P_{\max}$ algorithm further elaborates computational geometry techniques and is much faster than Megiddo+SW.

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.
  - The P_{NB} algorithms are essentially as fast as the non-parametric algorithms.
  - The deterministic $P_{\max}$ algorithm further elaborates computational geometry techniques and is much faster than Megiddo+SW.
- Open questions:

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.
  - The P_{NB} algorithms are essentially as fast as the non-parametric algorithms.
  - The deterministic $P_{\max}$ algorithm further elaborates computational geometry techniques and is much faster than Megiddo+SW.
- Open questions:
  - Can we use Karger's ideas to further speed up P_{NB} to $\tilde{O}(m)$?

Conclusion

- Solving P_{NB} and $P_{\max}$ by computing $Z(\mu)$ is slow.
- We could use Megiddo+SW to solve them faster deterministically, or Megiddo+KS to solve them faster randomly, which give the impression that P_{NB} and $P_{\max}$ for $d = 1$ have the same complexity.
  - Our algorithms suggest that P_{NB} is easier than $P_{\max}$ for $d = 1$.
- We propose specialized algorithms for solving P_{NB} and $P_{\max}$ that are significantly faster than Megiddo.
  - The P_{NB} algorithms are essentially as fast as the non-parametric algorithms.
  - The deterministic $P_{\max}$ algorithm further elaborates computational geometry techniques and is much faster than Megiddo+SW.
- Open questions:
  - Can we use Karger's ideas to further speed up P_{NB} to $\tilde{O}(m)$?
  - There should be a faster, specialized, randomized algorithm for $P_{\max}$.

Any questions?

Questions?

Comments?