Using greediness for parameterization

É. Bonnet, B. Escoffier, V. Th. Paschos, É. Tourniaire

September 2013
1. Definitions
   - Local cardinality constraint graph problems
   - Max and min \((k, n - k)\)-cut

2. Protective branching
   - An \(O^*((\Delta + 1)^k)\) algorithm for degrading contribution problems
   - An interesting consequence for max \((k, n-k)\)-cut

3. Non degrading contribution
   - An \(O^*((\Delta k)^{2k})\) algorithm
   - Other results
Local cardinality constraint (lcc) graph problem: find a set $V'$ of $k$ vertices to optimize $f(\delta(V'), |N(V')|, |E(V')|)$ where $f$ is a linear function.
Local cardinality constraint (lcc) graph problem: find a set $V'$ of $k$ vertices to optimize $f(\delta(V'), |N(V')|, |E(V')|)$ where $f$ is a linear function.

- $k$-sparsest, $k$-densest, max $(k,n-k)$-cut, min $(k,n-k)$-cut, max $k$-dominating set, min $k$-dominating set...
Local cardinality constraint (lcc) graph problem: find a set $V'$ of $k$ vertices to optimize $f(\delta(V'), |N(V')|, |E(V')|)$ where $f$ is a linear function.

- k-sparsest, k-densest, max (k,n-k)-cut, min (k,n-k)-cut, max k-dominating set, min k-dominating set...
- $W[1]$-hard w.r.t $k$: $O(g(k)n^c)$ algorithms (FPT) are very unlikely.
Local cardinality constraint (lcc) graph problem: find a set $V'$ of $k$ vertices to optimize $f(\delta(V'), |N(V')|, |E(V')|)$ where $f$ is a linear function.

- $k$-sparsest, $k$-densest, max $(k,n-k)$-cut, min $(k,n-k)$-cut, max $k$-dominating set, min $k$-dominating set...

- $W[1]$-hard w.r.t $k$: $O(g(k)n^c)$ algorithms (FPT) are very unlikely.

- What about $O(g(k, \Delta)n^c)$ algorithms?
Yes

Yes

Yes

Yes

Yes

Yes but...


- $g(k, \Delta) \approx 2^{(\Delta+1)k}$
Max \((k, n - k)\)-cut

Input: a graph \(G = (V, E)\) and two integers \(p, k\)
Output: Is there \(V' \subseteq V\) such that
\[ |V'| = k \]
\[ \text{val}(V') = \delta(V') = |E(V', V \setminus V')| \geq p \]
Min \((k, n - k)\)-cut

Input: a graph \(G = (V, E)\) and two integers \(p, k\)
Output: Is there \(V' \subseteq V\) such that
- \(|V'| = k\)
- \(val(V') = \delta(V') = |E(V', V \setminus V')| \leq p\)
Example

$k=4, \ p=5$
Definitions

Protective branching
Non degrading contribution

Local cardinality constraint graph problems
Max and min \((k, n - k)\)-cut

\(V'\)
Theorem

Max \((k, n - k)\)-cut can be solved in \(O^*((\Delta + 1)^k)\).
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$. 
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$.
- Maintain $U$ (unmarked, plain), $T$ (taken, filled) two lists of vertices partitioning $V$. 
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$.
- Maintain $U$ (unmarked, plain), $T$ (taken, filled) two lists of vertices partitioning $V$.
- Greedy choice for the branching.
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$.
- Maintain $U$ (unmarked, plain), $T$ (taken, filled) two lists of vertices partitioning $V$.
- Greedy choice for the branching.
- Hybridation technique to prove the optimality.
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$.
- Maintain $U$ (unmarked, plain), $T$ (taken, filled) two lists of vertices partitioning $V$.
- Greedy choice for the branching.
- Hybridation technique to prove the optimality.
- Contribution: $c_T(v) = |N(v) \cap U|$. 
Ingredients

- A marking and branching algorithm with a tree of size $f(\Delta, k)$.
- Maintain $U$ (unmarked, plain), $T$ (taken, filled) two lists of vertices partitioning $V$.
- Greedy choice for the branching.
- Hybridation technique to prove the optimality.
- Contribution: $c_T(v) = |N(v) \cap U|$.
- Degrading contribution: $T \subseteq T' \Rightarrow c_T(v) \leq c_{T'}(v)$. 
The algorithm

Set $U = V$, $T = \emptyset$.

$mkc(G, U, T, k, p)$:
if $k > 0$ then
The algorithm

Set $U = V$, $T = \emptyset$.

$mkc(G, U, T, k, p)$:

if $k > 0$ then

- Pick a vertex $v \in U$ maximizing
  $\delta_{U,T}(v) = |N(v) \cap U| - |N(v) \cap T|$.

else output $(T, val(T) \geq p)$.
The algorithm

Set $U = V$, $T = \emptyset$.

$mkc(G, U, T, k, p)$:
if $k > 0$ then

- Pick a vertex $v \in U$ maximizing
  $\delta_{U,T}(v) = |N(v) \cap U| - |N(v) \cap T|$.
- $N(v) \cap U = \{v_1, \ldots, v_l\}$ with $l \leq \Delta$. 

else output $(T, \text{val}(T) \geq p)$.
The algorithm

Set $U = V, T = \emptyset$.

$mkc(G, U, T, k, p)$:
if $k > 0$ then

- Pick a vertex $v \in U$ maximizing $\delta_{U,T}(v) = |N(v) \cap U| - |N(v) \cap T|$.
- $N(v) \cap U = \{v_1, \ldots, v_l\}$ with $l \leq \Delta$.
- $mkc(G, U \setminus \{v\}, T \cup \{v\}, k - 1, p)$,  
  $mkc(G, U \setminus \{v_1\}, T \cup \{v_1\}, k - 1, p)$, $\ldots$,  
  $mkc(G, U \setminus \{v_l\}, T \cup \{v_l\}, k - 1, p)$
The algorithm

Set $U = V$, $T = \emptyset$.

$mkc(G, U, T, k, p)$:

if $k > 0$ then

- Pick a vertex $v \in U$ maximizing
  \[ \delta_{U,T}(v) = |N(v) \cap U| - |N(v) \cap T|. \]
- $N(v) \cap U = \{v_1, \ldots, v_l\}$ with $l \leq \Delta$.
- $mkc(G, U \setminus \{v\}, T \cup \{v\}, k - 1, p)$,
  $mkc(G, U \setminus \{v_1\}, T \cup \{v_1\}, k - 1, p)$, \ldots,
  $mkc(G, U \setminus \{v_l\}, T \cup \{v_l\}, k - 1, p)$

else output $(T, val(T) \geq p)$
An \( O^*((\Delta+1)^k) \) algorithm for degrading contribution problems

An interesting consequence for max \((k,n-k)\)-cut

\[ k=2, p=4 \]
An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems

An interesting consequence for max (k,n-k)-cut

$k=2, p=4$
An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems

An interesting consequence for max $(k,n-k)$-cut

$k=2, p=4$

Using greediness for parameterization
$k=2, p=4$
$k=2, p=4$
An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems

An interesting consequence for max $(k,n-k)$-cut

$k=2, p=4$
An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems
An interesting consequence for max $(k,n-k)$-cut

$k=2, p=4$
k=2, p=4

An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems
An interesting consequence for max (k,n-k)-cut
An $O^*(\Delta+1)^k$ algorithm for degrading contribution problems
An interesting consequence for max $(k, n-k)$-cut

$k=2, p=4$
An $O^*((\Delta+1)^k)$ algorithm for degrading contribution problems

An interesting consequence for max $(k,n-k)$-cut

$k=2, p=4$
$k=2, p=4$
Complexity

The branching tree has:
Complexity

The branching tree has:

- Arity $\Delta + 1$ (at most).
Complexity

The branching tree has:

- Arity $\Delta + 1$ (at most).
- Depth $k$. 
Complexity

The branching tree has:

- Arity $\Delta + 1$ (at most).
- Depth $k$.
- $O((\Delta + 1)^k)$ leaves.
Soundness

$V_{opt}$ an optimal solution.
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.

![Diagram](image_url)
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.

$$\text{val}(V_{opt} \setminus \{z\} \cup \{v\}) \geq \text{val}(V_{opt}).$$
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.

\[ \text{val}(V_{opt} \setminus \{z\} \cup \{v\}) \geq \text{val}(V_{opt}). \]
- Iterate this principle at most $k$ times.
Soundness

- $V_{opt}$ an optimal solution.
- Consider in the branching tree the deviating point.

\[
\text{val}(V_{opt} \setminus \{z\} \cup \{v\}) \geq \text{val}(V_{opt}).
\]

- Iterate this principle at most $k$ times.
- Uses degrading contribution.
• Here, branching is not an end in itself but protects greediness.
• Close to Greedy Localization technique.
Corollary

Max \((k, n - k)\)-cut w.r.t \(p\) is FPT

\[
\begin{align*}
    k & \quad p & \quad \frac{n}{2} & \quad n - k & \quad n \\
    \Delta & \quad & \quad & \quad & \\
\end{align*}
\]

- \(p \geq \Delta\)
- \(p \geq k\)

É. Bonnet, B. Escoffier, V. Th. Paschos, É. Tourniaire

Using greediness for parameterization
\[ p \geq \min(rk, n - k) \geq k \]

(a) Vertices \( v \in V_2 \) and \( v' \in V_1 \) (that has at least one neighbor in \( V_1 \)) will be swapped.

(b) With the swapping the cut size increases.
**Theorem**

\[ \text{Min} (k, n - k)\text{-cut can be solved in } O^*((\Delta k)^{2k}). \]
What can we do without degrading contribution?

Problem: We can not build the solution vertex by vertex anymore.
What can we do without degrading contribution?

Problem: We can not build the solution vertex by vertex anymore.

\[ G' \]
What can we do without degrading contribution?

Problem: We can not build the solution vertex by vertex anymore.
What can we do without degrading contribution?

Problem: We can not build the solution vertex by vertex anymore.

Solution: Consider connected induced subgraph of size up to k.
Good news

**Lemma**

*One can enumerate the connected induced subgraphs of size $k$ in $O^*(\Delta^{2k})$.  

Idea: there is an injective function from those connected subgraphs to the binary trees with $k \lceil \log \Delta \rceil$ nodes.*
Bad news

Informally: an optimal solution is not necessarily a greedily chosen combination of connected components.
Outline of the algorithm

- Compute $S_1, \ldots, S_k$ where $S_i$ is a set of $i$ vertices inducing a connected component, and minimizes $\delta(.)$.
Outline of the algorithm

- Compute $S_1, \ldots, S_k$ where $S_i$ is a set of $i$ vertices inducing a connected component, and minimizes $\delta(.)$.
- Branch on each vertex of each $S_i$: the branching tree has size $k^{2k}$.
Outline of the algorithm

- Compute $S_1, \ldots, S_k$ where $S_i$ is a set of $i$ vertices inducing a connected component, and minimizes $\delta(.).$
- Branch on each vertex of each $S_i$: the branching tree has size $k^{2^k}$.
- Overall complexity: $O^*((\Delta k)^{2^k})$. 

Outline of the algorithm

- Compute $S_1, \ldots, S_k$ where $S_i$ is a set of $i$ vertices inducing a connected component, and minimizes $\delta(.)$.
- Branch on each vertex of each $S_i$: the branching tree has size $k^{2k}$.
- Overall complexity: $O^*((\Delta k)^{2k})$.
- Soundness: For each size of maximal connected component in $V_{opt}$, one can hybridate with a connected component of the same size.
**Theorem**

Max \((k,n-k)\)-cut has a fpt approximation schema.

\[ V' = \{v_1, \ldots, v_k\} \text{ the } k \text{ largest-degree vertices } d_1, \ldots, d_k. \text{ Let } B = \sum_{i=1}^{k} d_i. \]

\[ \text{SOL} \geq B - k^2, \text{ OPT} \leq B. \]

So, \( r \geq 1 - \frac{k^2}{B} \geq 1 - \frac{k^2}{\Delta} \).

- either \( \varepsilon \geq \frac{k^2}{\Delta} \), \( \Rightarrow \) \((1 - \varepsilon)\)-approximation.
- either \( \varepsilon \leq \frac{k^2}{\Delta} \), then \( \Delta \leq \frac{k^2}{\varepsilon} \) \( \Rightarrow \) fpt algorithm in \( k \).
Theorem

Min \((k,n-k)\)-cut has a randomized fpt approximation schema.

[Feige, Krauthgamer, Nissim ’03] If \(k < \log n\), there is a randomized polytime \((1 + \varepsilon)\)-approximation.
Conclusion and open questions
Conclusion and open questions

- Branching to protect local choices.
Conclusion and open questions

- Branching to protect local choices.
- Fear the worst to hybridate.
Conclusion and open questions

- Branching to protect local choices.
- Fear the worst to hybridate.
- An $O^*((c_1 \Delta)^{c_2 k})$ algorithm for all local cardinality constraint problems?
Conclusion and open questions

- Branching to protect local choices.
- Fear the worst to hybridate.
- An $O^*((c_1\Delta)^{c_2k})$ algorithm for all local cardinality constraint problems?
- ...at least for min (k,n-k)-cut?