Maximum Degree-Bounded Connected Subgraph:

Hardness and Approximation

Ignasi Sau

Joint work with O. Amini, D. Peleg, S. Perénnes, and S. Saurabh

Mascotte Project - INRIA/CNRS-I3S/UNSA - FRANCE Applied Mathematics IV Department of UPC - SPAIN

Outline of the talk

- Definition of the problem
- Example
- State of the art + our results
- Approximation algorithm
- Preliminaries
- Hardness results
- Conclusions and further research

• MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):

Input:

- an undirected graph G = (V, E),
- an integer $d \ge 2$, and
- a weight function $\omega : E \to \mathbb{R}^+$.

Output:

a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. G' = (V, E')

- ▶ is connected, and
- has **maximum degree** $\leq d$.
- It is one of the classical **NP**-complete problems of *[Garey and Johnson, Computers and Intractability, 1979].*

• If the output subgraph is not required to be connected, the problem is in **P** for any *d* (using matching techniques).

For fixed d = 2 it is the well known LONGEST PATH (OR CYCLE)

• MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):

Input:

- an undirected graph G = (V, E),
- an integer $d \ge 2$, and
- a weight function $\omega : \boldsymbol{E} \to \mathbb{R}^+$.

Output:

a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. G' = (V, E')

- is connected, and
- has **maximum degree** $\leq d$.
- It is one of the classical **NP**-complete problems of *[Garey and Johnson, Computers and Intractability, 1979].*

• If the output subgraph is not required to be connected, the problem is in **P** for any *d* (using matching techniques).

For fixed d = 2 it is the well known LONGEST PATH (OR CYCLE)

Ignasi Sau (Technion)

• MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):

Input:

- an undirected graph G = (V, E),
- an integer $d \ge 2$, and
- a weight function $\omega : \boldsymbol{E} \to \mathbb{R}^+$.

Output:

a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. G' = (V, E')

- is connected, and
- has **maximum degree** $\leq d$.
- It is one of the classical **NP**-complete problems of *[Garey and Johnson, Computers and Intractability, 1979].*

• If the output subgraph is not required to be connected, the problem is in **P** for any *d* (using matching techniques).

For fixed d = 2 it is the well known LONGEST PATH (OR CYCLE)

Ignasi Sau (Technion)

• MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):

Input:

- an undirected graph G = (V, E),
- an integer $d \ge 2$, and
- a weight function $\omega : E \to \mathbb{R}^+$.

Output:

a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. G' = (V, E')

- is connected, and
- has **maximum degree** $\leq d$.
- It is one of the classical **NP**-complete problems of *[Garey and Johnson, Computers and Intractability, 1979].*
- If the output subgraph is not required to be connected, the problem is in P for any d (using matching techniques).

• For fixed d = 2 it is the well known LONGEST PATH (OR CYCLE)

Ignasi Sau (Technion)

• MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):

Input:

- an undirected graph G = (V, E),
- an integer $d \ge 2$, and
- a weight function $\omega : \boldsymbol{E} \to \mathbb{R}^+$.

Output:

a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. G' = (V, E')

- is connected, and
- has **maximum degree** $\leq d$.
- It is one of the classical **NP**-complete problems of *[Garey and Johnson, Computers and Intractability, 1979].*
- If the output subgraph is not required to be connected, the problem is in P for any d (using matching techniques).
- For fixed d = 2 it is the well known LONGEST PATH (OR CYCLE)

Ignasi Sau (Technion)

Example with d = 3, $\omega(e) = 1$ for all $e \in E(G)$



Example with d = 3 (II)



Example with d = 3 (III)



Example with d = 3 (IV)



To the best of our knowledge, there were no results in the literature except for the case d = 2, a.k.a. the **LONGEST PATH** problem:

- Approximation algorithms: $\mathcal{O}\left(\frac{n}{\log n}\right)$ -approximation, using the **color-coding** method. N. Alon, R. Yuster and U. Zwick, STOC'94. $\mathcal{O}\left(n\left(\frac{\log \log n}{\log n}\right)^2\right)$ -approximation. A. Björklund and T. Husfeldt, SIAM J. Computing'03.
- Hardness results: It does not accept *any* constant-factor approximation.
 D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97.

To the best of our knowledge, there were no results in the literature except for the case d = 2, a.k.a. the **LONGEST PATH** problem:

• Approximation algorithms:

 $\mathcal{O}\left(\frac{n}{\log n}\right)$ -approximation, using the **color-coding** method. N. Alon, R. Yuster and U. Zwick, STOC'94. $\mathcal{O}\left(n\left(\frac{\log \log n}{\log n}\right)^2\right)$ -approximation. A. Björklund and T. Husfeldt, SIAM J. Computing'03.

 Hardness results: It does not accept *any* constant-factor approximation.
 D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97.

To the best of our knowledge, there were no results in the literature except for the case d = 2, a.k.a. the **LONGEST PATH** problem:

• Approximation algorithms:

 $\mathcal{O}\left(\frac{n}{\log n}\right)$ -approximation, using the **color-coding** method. N. Alon, R. Yuster and U. Zwick, STOC'94. $\mathcal{O}\left(n\left(\frac{\log \log n}{\log n}\right)^2\right)$ -approximation. A. Björklund and T. Husfeldt, SIAM J. Computing'03.

 Hardness results: It does not accept *any* constant-factor approximation.
 D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97.

To the best of our knowledge, there were no results in the literature except for the case d = 2, a.k.a. the **LONGEST PATH** problem:

Approximation algorithms:

 $\mathcal{O}\left(\frac{n}{\log n}\right)$ -approximation, using the **color-coding** method. N. Alon, R. Yuster and U. Zwick, STOC'94. $\mathcal{O}\left(n\left(\frac{\log \log n}{\log n}\right)^2\right)$ -approximation. A. Björklund and T. Husfeldt, SIAM J. Computing'03.

Hardness results:

It does not accept *any* constant-factor approximation. D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97.

• Approximation algorithms (n = |V(G)|, m = |E(G)|):

• min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.

- min $\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor.
- Hardness results:
 - For each fixed d ≥ 2, MDBCS_d does not accept any constant-factor approximation in general graphs.

• Approximation algorithms (n = |V(G)|, m = |E(G)|):

- min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.
- $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor.
- Hardness results:
 - For each fixed $d \ge 2$, MDBCS_d does not accept *any* constant-factor approximation in general graphs.

• Approximation algorithms (n = |V(G)|, m = |E(G)|):

- min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.
- $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor.

Hardness results:

For each fixed *d* ≥ 2, MDBCS_d does not accept *any* constant-factor approximation in general graphs.

• Approximation algorithms (n = |V(G)|, m = |E(G)|):

- min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.
- $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor.

Hardness results:

► For each fixed d ≥ 2, MDBCS_d does not accept any constant-factor approximation in general graphs.

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection F of k connected components, we glue them in k - 1 phases. In each phase:
 - For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component \overline{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component \overline{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component \hat{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component \hat{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - ▶ Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C')
 - ▶ Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component \overline{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

Description of the algorithm: Two cases according to H = (W, F):

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$

► Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').

- ▶ Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
- ▶ Then we merge C, C', and the path $p(u, u') \rightarrow$ new component C.

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - ▶ Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - ▶ Let $u \in C$ and $u' \in C'$ be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge *C*, *C'*, and the path $p(u, u') \rightarrow$ new component \hat{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let u ∈ C and u' ∈ C' be two vertices realizing this distance. Let p(u, u') be a shortest path between u and u' in G.
 - ▶ Then we merge *C*, *C'*, and the path $p(u, u') \rightarrow$ new component *C*.

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let *u* ∈ *C* and *u'* ∈ *C'* be two vertices realizing this distance. Let *p*(*u*, *u'*) be a shortest path between *u* and *u'* in *G*.
 - ▶ Then we merge *C*, *C*', and the path $p(u, u') \rightarrow$ new component \hat{C} .

Input: undirected graph G = (V, E), a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \ge 2$. Let n = |V|, m = |E|, and $\rho = \min\{n/2, m/d\}$.

F: set of *d* heaviest edges in *G*, with weight $\omega(F)$. *W*: set of endpoints of those edges. Let H = (W, F).

- (1) If H = (W, F) is connected, the algorithm returns H. **Claim**: this yields a ρ -approximation.
- (2) If H = (W, F) consists of a collection \mathcal{F} of k connected components, we *glue* them in k 1 phases. In each phase:
 - ► For every two components $C, C' \in \mathcal{F}$, compute $d(C, C') = \min\{dist(u, u', G) \mid u \in C, u' \in C'\}.$
 - Take a pair $C, C' \in \mathcal{F}$ attaining the smallest d(C, C').
 - Let *u* ∈ *C* and *u'* ∈ *C'* be two vertices realizing this distance. Let *p*(*u*, *u'*) be a shortest path between *u* and *u'* in *G*.
 - ▶ Then we merge *C*, *C'*, and the path $p(u, u') \rightarrow$ new component \hat{C} .

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- Claim: after *i* phases, Δ(H) ≤ d − k + i + 1. The proof is done by induction. When i = k − 1 we get Δ(H) ≤ d.

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- Claim: after *i* phases, Δ(H) ≤ d − k + i + 1. The proof is done by induction. When i = k − 1 we get Δ(H) ≤ d.

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- Claim: after *i* phases, Δ(H) ≤ d − k + i + 1. The proof is done by induction. When i = k − 1 we get Δ(H) ≤ d.

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- Claim: after *i* phases, Δ(H) ≤ d − k + i + 1. The proof is done by induction. When i = k − 1 we get Δ(H) ≤ d.

Preliminaries: hardness of approximation

Class APX (Approximable):

an NP-complete optimization problem is in APX if it can be approximated within a constant factor.

Example: VERTEX COVER

• Class PTAS (Polynomial-Time Approximation Scheme):

an NP-complete optimization problem is in PTAS if it can be approximated within a constant factor $1 + \varepsilon$, for *all* $\varepsilon > 0$ (the best one can hope for an NP-complete problem).

Example: MAXIMUM KNAPSACK

(1) First we prove that MDBCS_d ∉ PTAS: Reduction from TSP(1,2).

(2) Then we prove that $MDBCS_d \notin APX$:

- Let $\alpha > 1$ be the hardness factor of MDBCS_d given by (1).
- We use a technique called error amplification:
 - * We build a sequence of families of graphs \mathcal{G}^k , such that MDBCS_d is hard to approximate in \mathcal{G}^k within a factor α^k , unless P = NP.
 - * This proves that the problem is not in APX.

(for any constant *C*, $\exists k > 0$ such that $\alpha^k > C$).

Let G¹ = G. We explain the construction of G²: first take our graph G and...

- (1) First we prove that MDBCS_d ∉ PTAS: Reduction from TSP(1,2).
- (2) Then we prove that $MDBCS_d \notin APX$:
 - Let $\alpha > 1$ be the hardness factor of MDBCS_d given by (1).
 - We use a technique called error amplification:
 - * We build a sequence of families of graphs \mathcal{G}^k , such that MDBCS_d is hard to approximate in \mathcal{G}^k within a factor α^k , unless P = NP.
 - ★ This proves that the problem is not in APX.

(for any constant *C*, $\exists k > 0$ such that $\alpha^k > C$).

Let G¹ = G. We explain the construction of G²: first take our graph G and...

- (1) First we prove that MDBCS_d ∉ PTAS: Reduction from TSP(1,2).
- (2) Then we prove that $MDBCS_d \notin APX$:
 - Let $\alpha > 1$ be the hardness factor of MDBCS_d given by (1).
 - We use a technique called error amplification:
 - * We build a sequence of families of graphs \mathcal{G}^k , such that MDBCS_d is hard to approximate in \mathcal{G}^k within a factor α^k , unless P = NP.
 - ★ This proves that the problem is not in APX.

(for any constant *C*, $\exists k > 0$ such that $\alpha^k > C$).

• Let $G^1 = G$. We explain the construction of G^2 : first take our graph G and...

- (1) First we prove that MDBCS_d ∉ PTAS: Reduction from TSP(1,2).
- (2) Then we prove that $MDBCS_d \notin APX$:
 - Let $\alpha > 1$ be the hardness factor of MDBCS_d given by (1).
 - We use a technique called error amplification:
 - * We build a sequence of families of graphs \mathcal{G}^k , such that MDBCS_d is hard to approximate in \mathcal{G}^k within a factor α^k , unless P = NP.
 - This proves that the problem is not in APX.

(for any constant C, $\exists k > 0$ such that $\alpha^k > C$).

Let G¹ = G. We explain the construction of G²: first take our graph G and...

- (1) First we prove that MDBCS_d ∉ PTAS: Reduction from TSP(1,2).
- (2) Then we prove that $MDBCS_d \notin APX$:
 - Let $\alpha > 1$ be the hardness factor of MDBCS_d given by (1).
 - We use a technique called error amplification:
 - * We build a sequence of families of graphs \mathcal{G}^k , such that MDBCS_d is hard to approximate in \mathcal{G}^k within a factor α^k , unless P = NP.
 - ★ This proves that the problem is not in APX.

(for any constant C, $\exists k > 0$ such that $\alpha^k > C$).

Let G¹ = G. We explain the construction of G²: first take our graph G and...

For each pair of vertices $\{u, v\} \in V^2$, $u \neq v$, we build the graph $G_{u,v}^2$ in the following way:



Error amplification to prove that $MDBCS_d \notin APX$ (II) We replace each edge $e_i = (x, y) \in E(G)$ with a copy G_i of G, i = 1, ..., m:



The copy of the vertex $u \in V(G)$ in G_i is labeled u_i . For each $e_i = (x, y) \in E(G)$, we add the edges (x, u_i) and (y, v_i) with weight ε , $0 < \varepsilon << 1$.



- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x.
 To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - Case b: S₂ intersects strictly fewer than √x copies of G.
 Let S₁ be S₂ ∩ G_i, with G_i being the copy of G in G² such that |E(S₂ ∩ G_i)| is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

• Combining Claims 1 and 2: if there exists a ρ -approximation in G^2 , then it is possible to find a solution for G with weight at least

 $\frac{OPI_2}{\rho} \geq \frac{OPI_1}{\sqrt{\rho}} \Rightarrow$ we have a $\sqrt{\rho}$ -approximation in G.

- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x. To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - Case b: S₂ intersects strictly fewer than √x copies of G.
 Let S₁ be S₂ ∩ G_i, with G_i being the copy of G in G² such that |E(S₂ ∩ G_i)| is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

 Combining Claims 1 and 2: if there exists a ρ-approximation in G², then it is possible to find a solution for G with weight at least

 $\frac{OPI_2}{\rho} \geq \frac{OPI_1}{\sqrt{\rho}} \Rightarrow$ we have a $\sqrt{\rho}$ -approximation in G.

- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- **Claim 2**: Given any solution S_2 in G^2 with weight *x*, it is possible to find a solution S_1 in *G* with weight at least \sqrt{x} .

To prove the claim, we distinguish two cases:

- Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
- Case b: S₂ intersects strictly fewer than √x copies of G.
 Let S₁ be S₂ ∩ G_i, with G_i being the copy of G in G² such that |E(S₂ ∩ G_i)| is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

 Combining Claims 1 and 2: if there exists a ρ-approximation in G², then it is possible to find a solution for G with weight at least

 $\frac{1}{2} \frac{OPI_1}{\sqrt{\rho}} \ge \frac{OPI_1}{\sqrt{\rho}} \Rightarrow$ we have a $\sqrt{\rho}$ -approximation in G.

- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x.
 To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - Case b: S₂ intersects strictly fewer than √x copies of G.
 Let S₁ be S₂ ∩ G_i, with G_i being the copy of G in G² such that |E(S₂ ∩ G_i)| is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.



- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x.
 To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - **Case b:** S_2 intersects strictly fewer than \sqrt{x} copies of *G*. Let S_1 be $S_2 \cap G_i$, with G_i being the copy of *G* in G^2 such that $|E(S_2 \cap G_i)|$ is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

$$\left/rac{OPT_2}{
ho}\geqrac{OPT_1}{\sqrt{
ho}}\Rightarrow$$
 we have a $\sqrt{
ho}$ -approximation in $G.$

- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x.
 To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - **Case b:** S_2 intersects strictly fewer than \sqrt{x} copies of G. Let S_1 be $S_2 \cap G_i$, with G_i being the copy of G in G^2 such that $|E(S_2 \cap G_i)|$ is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

$$\frac{OPT_2}{\rho} \geq \frac{OPT_1}{\sqrt{\rho}} \Rightarrow$$
 we have a $\sqrt{\rho}$ -approximation in *G*.

- Suppose we have an approx. algo C with ratio ρ. We define G² as the graph G²_{u,v} for which algorithm C gives the best solution.
- Claim 1: $OPT_2 \ge OPT_1^2 + 2\varepsilon \cdot OPT_1 \approx OPT_1^2$.
- Claim 2: Given any solution S₂ in G² with weight x, it is possible to find a solution S₁ in G with weight at least √x.
 To prove the claim, we distinguish two cases:
 - Case a: S₂ intersects at least √x copies of G.
 Let S₁ be the subgraph of G induced by the edges corresponding to these copies of G in G².
 - **Case b:** S_2 intersects strictly fewer than \sqrt{x} copies of G. Let S_1 be $S_2 \cap G_i$, with G_i being the copy of G in G^2 such that $|E(S_2 \cap G_i)|$ is maximized.

In both cases S_1 is connected, has maximum degree at most d, and has at least \sqrt{x} edges.

$$\sqrt{\frac{OPT_2}{\rho}} \ge \frac{OPT_1}{\sqrt{\rho}} \Rightarrow$$
 we have a $\sqrt{\rho}$ -approximation in *G*.

• We have proved that $MDBCS_d$, $d \ge 2$, is not in APX.

• We have provided approximation algorithms for any *d*:

- min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.
- min $\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- We have also proved that when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor in unweighted graphs.

• Further Research:

- Close the huge complexity gap of $MDBCS_d$, $d \ge 2$.
- Find polynomial cases or better approximation algorithms for specific classes of graphs.
- Consider a parameterized version of the problem.

- We have proved that $MDBCS_d$, $d \ge 2$, is not in APX.
- We have provided approximation algorithms for any *d*:
 - min $\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for weighted graphs.
 - min $\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
 - We have also proved that when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor in unweighted graphs.
- Further Research:
 - Close the huge complexity gap of $MDBCS_d$, $d \ge 2$.
 - Find polynomial cases or better approximation algorithms for specific classes of graphs.
 - Consider a parameterized version of the problem.

- We have proved that $MDBCS_d$, $d \ge 2$, is not in APX.
- We have provided approximation algorithms for any *d*:
 - $\min\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for **weighted** graphs.
 - $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
 - We have also proved that when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor in unweighted graphs.
- Further Research:
 - Close the huge complexity gap of $MDBCS_d$, $d \ge 2$.
 - Find polynomial cases or better approximation algorithms for specific classes of graphs.
 - Consider a parameterized version of the problem.

• We have proved that $MDBCS_d$, $d \ge 2$, is not in APX.

• We have provided approximation algorithms for any *d*:

- $\min\{\frac{n}{2}, \frac{m}{d}\}$ -approximation algorithm for **weighted** graphs.
- min $\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- We have also proved that when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor in unweighted graphs.

• Further Research:

- Close the huge complexity gap of $MDBCS_d$, $d \ge 2$.
- Find polynomial cases or better approximation algorithms for specific classes of graphs.
- Consider a parameterized version of the problem.

• We have proved that $MDBCS_d$, $d \ge 2$, is not in APX.

• We have provided approximation algorithms for any *d*:

- min{ $\frac{n}{2}$, $\frac{m}{d}$ }-approximation algorithm for weighted graphs.
- min $\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$ -approximation algorithm for **unweighted** graphs.
- We have also proved that when G accepts a low-degree spanning tree, in terms of d, then MDBCS_d can be approximated within a small constant factor in unweighted graphs.

• Further Research:

- Close the huge complexity gap of $MDBCS_d$, $d \ge 2$.
- Find polynomial cases or better approximation algorithms for specific classes of graphs.
- Consider a parameterized version of the problem.

Thanks!