智能网络与优化实验室

Intelligent Network and Optimization Laboratory, Renmin University

Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs

Author: Chenhao Ma, Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V.S. Lakshmanan, Wenjie Zhang, Xuemin Lin

One of the Best Papers in SIGMOD 2020

Xiaojia Xu

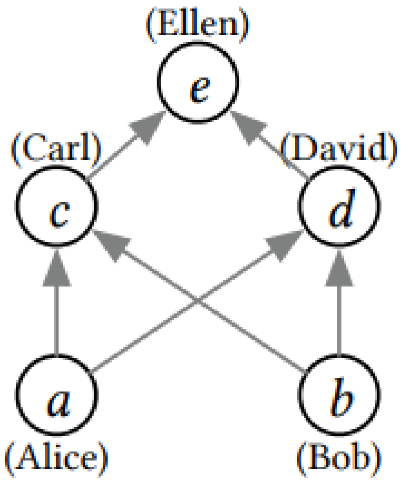
Problem

- **The directed densest subgraph (DDS) problem**

- **Given a directed graph $G = (\mathrm{V}, \mathrm{E})$, the directed densest subgraph (DDS) is the problem of discovering the $(S^*, T^*) - induced$ subgraph, whose density is the highest among all the possible $(S, T) - induced$ subgraphs**

[4] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment* 5, 5 (2012), 454–465.
[23] Aristides Gionis and Charalampos E Tsourakakis. 2015. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2313–2314.
[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.
[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

Background



(a) A directed graph

[31] Ravi Kannan and V Vinay. 1999. *Analyzing the structure of large graphs.* Rheinische Friedrich-Wilhelms-Universität Bonn Bonn.

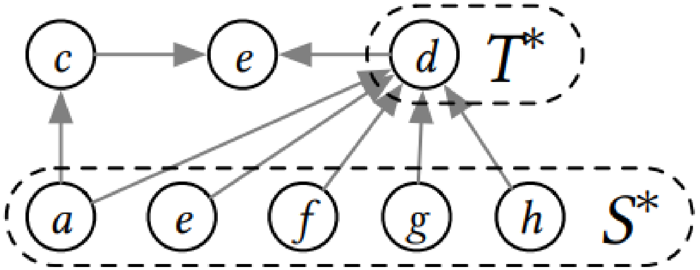


Figure 1: An example of fake follower detection.

[26] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 895–904.

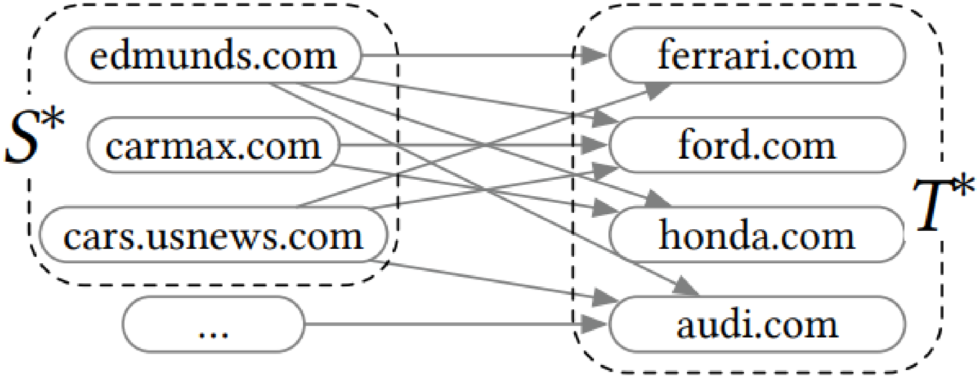


Figure 2: An example of web community.

[33] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

State-of-the-art

Table 1: Summary of exact algorithms.

Algorithm	Time complexity
LP-Exact [10]	$\Omega(n^6)$
Exact [32]	$O(n^3 m \log n)$
DC-Exact **(Ours)**	$O(k \cdot nm \log n)$

Table 2: Summary of approximation algorithms.

Algorithm	Approx. ratio	Time complexity
KV-Approx [31]	$O(\log n)$	$O(s^3 n)$
PM-Approx [4]	$2\delta(1+\epsilon)$	$O(\frac{\log n}{\log \delta} \log_{1+\epsilon} n(n+m))$
KS-Approx [32]	>2	$O(n+m)$
BS-Approx [10]	2	$O(n^2 \cdot (n+m))$
Core-Approx **(Ours)**	2	$O(\sqrt{m}(n+m))$

Note: s is the sample size; ϵ, δ are the error tolerance parameters.

[4] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment* 5, 5 (2012), 454–465.

[31] Ravi Kannan and V Vinay. 1999. *Analyzing the structure of large graphs.* Rheinische Friedrich-Wilhelms-Universität Bonn Bonn.

[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

Fundamentals

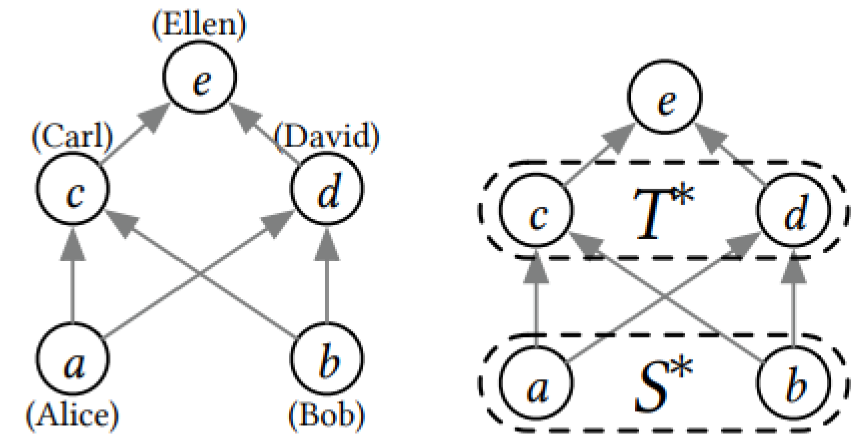
Notation	Meaning
$G=(V,E)$	a directed graph with vertex set V and edge set E
n, m	$n = \|V\|$, $m = \|E\|$
$H=G[S,T]$	the subgraph induced by S and T in G
$E(S,T)$	the edges induced by S and T in G
$d^-_G(v)$, $d^+_G(v)$	the outdegree and indegree of a vertex $v \in G$ resp.
$\rho(S,T)$	the density of the (S, T)-induced subgraph
$D = G[S^*, T^*]$	the densest subgraph D in G
ρ^*	$\rho^* = \max_{S,T\subset V}\{\rho(S,T)\} = \rho(S^*,T^*)$
$\widetilde{D} = G[\widetilde{S^*}, \widetilde{T^*}]$	the approximate densest subgraph in G
$\widetilde{\rho^*} = \rho(\widetilde{S^*}, \widetilde{T^*})$	the density of $\widetilde{D}$
$F = (V_F, E_F)$	a flow network with node set V_F and edge set E_F

Fundamentals

Definition 3.1 (Density of a directed graph). Given a directed graph G=(V, E) and two sets of vertices $S, T \subseteq V$, the density of the (S, T)-induced subgraph $G[S, T]$ is

$$\rho(S,T) = \frac{|E(S,T)|}{\sqrt{|S| \cdot |T|}}. \quad (1)$$

Definition 3.2 (DDS). Given a directed graph G=(V, E), a directed densest subgraph (DDS) D is the (S^*, T^*)-induced subgraph, whose density is the highest among all the possible (S, T)-induced subgraphs.



(a) A directed graph **(b) Directed DS**

$$\boxed{\rho(S^*, T^*) = \frac{4}{\sqrt{2\times 2}}}$$

$$\rho(V, V) = \frac{6}{\sqrt{5\times 5}} = \frac{6}{5}$$

EXISTING ALGORITHMS: The Exact Algorithm

Algorithm 1: Exact [32]

Input : G=(V, E)

Output : The exact DDS D=G[S^*, T^*]

$\rho^* \leftarrow 0$;

foreach $a \in \{\frac{n_1}{n_2} \mid 0 < n_1, n_2 <= n\}$ **do**

  $l \leftarrow 0$, $r \leftarrow \max_{u \in V}\{d^+_G(u), d^-_G(u)\}$;

  while $r - l \geq \frac{\sqrt{n}-\sqrt{n-1}}{n\sqrt{n-1}}$ **do**

    $g \leftarrow \frac{l+r}{2}$;

    F=(V_F, E_F)← BuildFlowNetwok(G, a, g);

    $\langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ Min-ST-Cut(F);

    if $\mathcal{S}$={s} **then** $r \leftarrow g$;

    else

      $l \leftarrow g$;

      if $g > \rho^*$ **then** $D \leftarrow G[\mathcal{S} \cap A, \mathcal{S} \cap B]$, $\rho^* = g$;

return D;

Function BuildFlowNetwok($G = (V, E)$, a, g):

  $A \leftarrow \{\alpha_u | u \in V\}$, $B \leftarrow \{\beta_u | u \in V\}$, $E_F \leftarrow \emptyset$;

  $V_F \leftarrow \{s\} \cup A \cup B \cup \{t\}$;

  for $\alpha_u \in A$ **do** add (s, α_u) to E_F with capacity m;

  for $\beta_u \in B$ **do** add (s, β_u) to E_F with capacity m;

  for $\alpha_u \in A$ **do** add (α_u, t) to E_F with capacity $m + \frac{g}{\sqrt{a}}$;

  for $\beta_u \in B$ **do** add (β_u, t) to E_F with capacity $m + \sqrt{a}g - 2d^+_G(u)$;

  for $(u, v) \in E$ **do** add (β_v, α_u) to E_F with capacity 2;

  return $F = (V_F, E_F)$

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

EXISTING ALGORITHMS: The Exact Algorithm

Algorithm 1: Exact [32]

Input : $G=(V, E)$
Output : The exact DDS $D=G[S^*, T^*]$

$\rho^* \leftarrow 0$;
foreach $a \in \{\frac{n_1}{n_2} \mid 0 < n_1, n_2 <= n\}$ **do**
  $l \leftarrow 0, r \leftarrow \max_{u \in V}\{d_G^+(u), d_G^-(u)\}$;
  while $r - l \geq \frac{\sqrt{n}-\sqrt{n-1}}{n\sqrt{n-1}}$ **do**
    $g \leftarrow \frac{l+r}{2}$;
    $F=(V_F, E_F) \leftarrow$ `BuildFlowNetwok`(G, a, g);
    $\langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ `Min-ST-Cut`(F);
    if $\mathcal{S}=\{s\}$ **then** $r \leftarrow g$;
    else
      $l \leftarrow g$;
      if $g > \rho^*$ **then** $D \leftarrow G[\mathcal{S} \cap A, \mathcal{S} \cap B]$, $\rho^* = g$;
return D;
Function `BuildFlowNetwok`$(G = (V, E), a, g)$:
  $A \leftarrow \{\alpha_u | u \in V\}$, $B \leftarrow \{\beta_u | u \in V\}$, $E_F \leftarrow \emptyset$;
  $V_F \leftarrow \{s\} \cup A \cup B \cup \{t\}$;
  for $\alpha_u \in A$ **do** add (s, α_u) to E_F with capacity m;
  for $\beta_u \in B$ **do** add (s, β_u) to E_F with capacity m;
  for $\alpha_u \in A$ **do** add (α_u, t) to E_F with capacity $m + \frac{g}{\sqrt{a}}$;
  for $\beta_u \in B$ **do** add (β_u, t) to E_F with capacity $m + \sqrt{a}g - 2d_G^+(u)$;
  for $(u, v) \in E$ **do** add (β_v, α_u) to E_F with capacity 2;
  return $F = (V_F, E_F)$

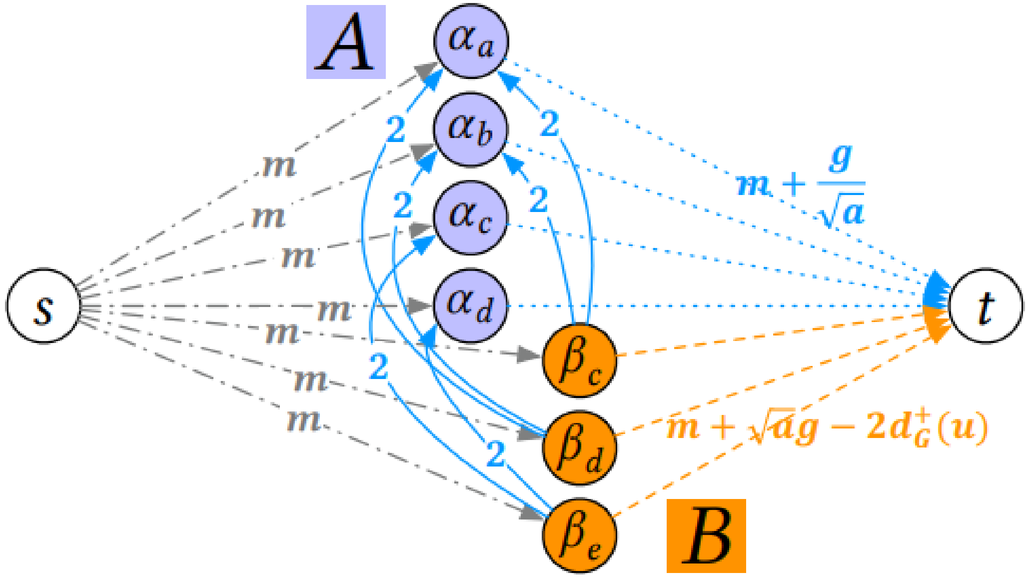


Figure 4: Illustrating the flow network.

EXISTING ALGORITHMS: Approximation Alogrithms

Algorithm 2: KS-Approx [32]

Input : G=(V, E)

Output : An approximate DDS $\widetilde{D}$

$\widetilde{\rho}^* \leftarrow 0, \widetilde{D} \leftarrow \emptyset, S \leftarrow V, T \leftarrow V$;

while $|E|>0$ **do**

  if $\rho(S,T) > \widetilde{\rho}^*$ **then**

    $\widetilde{\rho}^* \leftarrow \rho(S,T), \widetilde{D} \leftarrow (S,T)$;

  $u_+ \leftarrow \arg\min_u d_G^+(u), u_- \leftarrow \arg\min_u d_G^-(u)$;

  if $d_G^+(u_+) \leq d_G^-(u_-)$ **then**

    $E \leftarrow E \setminus \{(v,u_+)|v \in S\}, T \leftarrow T \setminus \{u_+\}$;

  else

    $E \leftarrow E \setminus \{(u_-,v)|v \in T\}, S \leftarrow S \setminus \{u_-\}$;

return $\widetilde{D}$;

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

EXISTING ALGORITHMS: Approximation Alogrithms

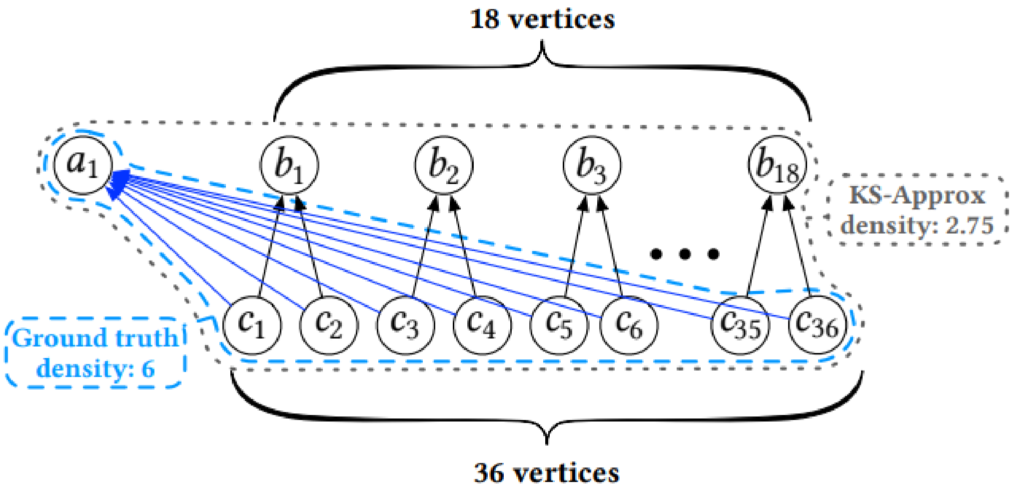


Figure 5: A counter-example for KS-Approx.

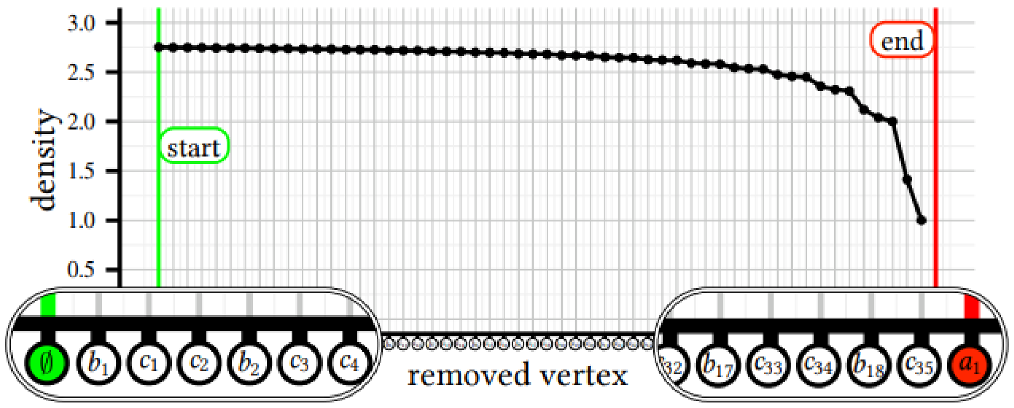


Figure 6: Running steps by KS-Approx.

Why KS-Approx fails? KS-Approx is supported by Theorem 2 in [32]. Theorem 2 requires that there is a iteration that $\forall u \in S, d^-_{G[S,T]}(u) \geq \lambda_o = |E(S^*, T^*)| \cdot \left(1 - \sqrt{1 - \frac{1}{|S^*|}}\right)$ and $\forall v \in T, d^+_{G[S,T]}(v) \geq \lambda_i = |E(S^*, T^*)| \cdot \left(1 - \sqrt{1 - \frac{1}{|T^*|}}\right)$. In Example 4.1, $S^* = \{c_i | 1 \leq i \leq 36\}$ and $T^* = \{a_1\}$. Thus, $\lambda_o = 0.5035$ and $\lambda_i = 36$. By reviewing the iterations of KS-Approx over the counter-example, we can find such condition cannot be guaranteed simultaneously.

- **FKS-Approx [1]**

[1] 2019. Supplementary Note. https://i.cs.hku.hk/~chma2/sup-sigmod2020.pdf. (2019).

EXISTING ALGORITHMS: Approximation Alogrithms

Algorithm 3: BS-Approx [10]

Input : G=(V, E)

Output : An approximate DDS $\widetilde{D}$

$\widetilde{\rho}^* \leftarrow 0, \widetilde{D} \leftarrow \emptyset$;

foreach $a \in \{\frac{n_1}{n_2} | 0 < n_1, n_2 <= n\}$ **do**

  $S \leftarrow V, T \leftarrow V$;

  while $S \neq \emptyset \wedge T \neq \emptyset$ **do**

    if $\rho(S, T) > \widetilde{\rho}^*$ **then** $\widetilde{D} \leftarrow G[S, T]$, $\widetilde{\rho}^* \leftarrow \rho(S, T)$;

    $u \leftarrow \arg\min_{u \in S} d^-_G(u)$;

    $v \leftarrow \arg\min_{v \in T} d^+_G(v)$;

    if $\sqrt{a} \cdot d^-_G(u) \leq \frac{1}{\sqrt{a}} \cdot d^+_G(v)$ **then** $S \leftarrow S \setminus \{u\}$;

    else $T \leftarrow T \setminus \{v\}$;

return $\widetilde{D}$;

[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

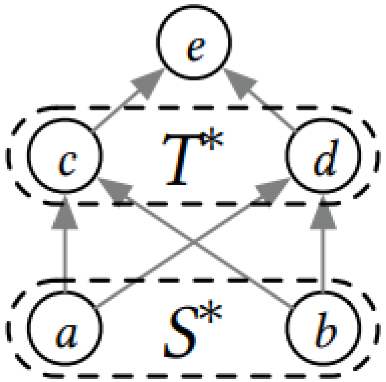
Algorithms

Definition 5.1 (k-core [5, 47]). Given an undirected graph G and an integer k ($k \geq 0$), the k-core, denoted by $\mathcal{H}_k$, is the largest subgraph of G, such that $\forall v \in \mathcal{H}_k$, $deg_{\mathcal{H}_k}(v) \geq k$.

Definition 5.2 ([x, y]-core). Given a directed graph G=(V, E), an (S, T)-induced subgraph H=G[S, T] is called an [x, y]-**core**, if it satisfies:

(1) $\forall u \in S, d^-_H \geq x$ and $\forall v \in T, d^+_H \geq y$;
(2) $\nexists H'$, s.t. $H \subset H'$ and H' satisfies (1).

We call [x, y] the **core number pair** of the [x, y]-core, abbreviated as **cn-pair**.



(b) Directed DS

Example 5.3. The subgraph induced by (S^*, T^*), i.e., $D = G[S^*, T^*]$ in Figure 3b is a $[2, 2]$-core. $H = G[\{a, b, c, d\}, \{c, d, e\}]$ is a $[1, 2]$-core, and D is contained in H. □

Algorithms

Lemma 5.4 (Nested property). *An $[x, y]$-core is contained by an $[x', y']$-core, where $x \geq x' \geq 0$ and $y \geq y' \geq 0$. In other words, if H=$G[S, T]$ is an $[x, y]$-core, there must exist an $[x', y']$-core H'=$G[S', T']$, such that $S \subseteq S'$ and $T \subseteq T'$.*

A Core-based Exact Algorithm

LEMMA 5.5. *Given a directed graph G=(V, E) and its DDS D=$G[S^*, T^*]$ with density ρ^*, we have following conclusions:*

(1) for any subset U_S of S^*, removing U_S from S^* will result in the removal of at least $\frac{\rho^*}{2\sqrt{a}} \times |U_S|$ edges from D,

(2) for any subset U_T of T^*, removing U_T from T^* will result in the removal of at least $\frac{\sqrt{a}\rho^*}{2} \times |U_T|$ edges from D,

where $a = \frac{|S^*|}{|T^*|}$.

A Core-based Exact Algorithm

LEMMA 5.5. *Given a directed graph G=(V, E) and its DDS D=$G[S^*, T^*]$ with density ρ^*, we have following conclusions:*

(1) for any subset U_S of S^*, removing U_S from S^* will result in the removal of at least $\frac{\rho^*}{2\sqrt{a}} \times |U_S|$ edges from D,

(2) for any subset U_T of T^*, removing U_T from T^* will result in the removal of at least $\frac{\sqrt{a}\rho^*}{2} \times |U_T|$ edges from D,

where $a = \frac{|S^*|}{|T^*|}$.

PROOF. We prove the lemma by contradiction. For (1), we assume that D is the DDS and removing U_S from D results in the removal of less than $\frac{\rho^*}{2\sqrt{a}} \times |U_S|$ edges from D. This implies that, after removing U_S from S^*, the density of the residual graph, denoted by D_R=$G[S^* \setminus U_S, T^*]$, will be

$$\begin{aligned}\rho(S^* \setminus U_S, T^*) = \frac{|E(S^* \setminus U_S, T^*)|}{\sqrt{|S^* \setminus U_S||T^*|}} &> \frac{\rho^* \sqrt{|S^*||T^*|} - \frac{\rho^*}{2\sqrt{a}}|U_S|}{\sqrt{(|S^*| - |U_S|)|T^*|}} \\ &= \rho^* \frac{|S^*| - \frac{|U_S|}{2}}{\sqrt{|S^*|^2 - |S^*||U_S|}} \\ &= \rho^* \frac{|S^*| - \frac{|U_S|}{2}}{\sqrt{(|S^*| - \frac{|U_S|}{2})^2 - \frac{|U_S|^2}{4}}} \\ &> \rho^*.\end{aligned}$$

However, this contradicts the assumption that D is the DDS, so the conclusion of (1) holds. Similarly, we can prove that the conclusion of (2) holds as well. Hence, the lemma holds. □

A Core-based Exact Algorithm

THEOREM 5.6. *Given a graph G=(V, E), the DDS D=G[S^*, T^*] is contained in the* $\left[\lceil \frac{\rho^*}{2\sqrt{a}} \rceil, \lceil \frac{\sqrt{a}\rho^*}{2} \rceil\right]$*-core, where* $a = \frac{|S^*|}{|T^*|}$.

PROOF. By Lemma 5.5, removing any single vertex u from S^* will result in the removal of $\lceil \frac{\rho^*}{2\sqrt{a}} \rceil$ edges from D, so we conclude that for each vertx $u \in S^*$, $d_D^-(u) \geq \lceil \frac{\rho^*}{2\sqrt{a}} \rceil$. Similarly, for each vertex $v \in T^*$, we have $d_D^+(v) \geq \lceil \frac{\sqrt{a}\rho^*}{2} \rceil$. Thus, by the definition of [x, y]-core, we conclude that the DDS is in the $\left[\lceil \frac{\rho^*}{2\sqrt{a}} \rceil, \lceil \frac{\sqrt{a}\rho^*}{2} \rceil\right]$-core, where $a = \frac{|S^*|}{|T^*|}$. □

A Core-based Exact Algorithm

Algorithm 4: Core-Exact

Input : $G=(V,E)$
Output : The exact DDS $D=G[S^*,T^*]$

$\widetilde{\rho^*} \leftarrow$ run a 2-approximation algorithm;
$\rho^* \leftarrow \widetilde{\rho^*}$;
foreach $a \in \{\frac{n_1}{n_2} | 0 < n_1, n_2 <= n\}$ **do**
  $l \leftarrow \rho^*, r \leftarrow 2\widetilde{\rho^*}$;
  while $r - l \geq \frac{\sqrt{n}-\sqrt{n-1}}{n\sqrt{n-1}}$ **do**
    $g \leftarrow \frac{l+r}{2}, x \leftarrow \lceil \frac{l}{2\sqrt{a}} \rceil, y \leftarrow \lceil \frac{\sqrt{a}l}{2} \rceil$;
    $G_r \leftarrow$ Get-XY-Core(G, x, y);
    $F = (V_F, E_F) \leftarrow$ BuildFlowNetwok(G_r, a, g);
    $\langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ Min-ST-Cut(F);
    if $\mathcal{S} = \{s\}$ **then** $r \leftarrow g$;
    else
      $l \leftarrow g$;
      if $g > \rho^*$ **then** $D \leftarrow G[\mathcal{S} \cap A, \mathcal{S} \cap B]$, $\rho^* = g$;
return D;

- **time complexity:** $O(n^3 m \log n)$

A Divide-and-conquer Exact Algorithm

$$\max_{S,T \in V} g$$
$$\text{s.t. } \frac{|S|}{\sqrt{a}}\left(g - \frac{|E(S,T)|}{|S|/\sqrt{a}}\right) + |T|\sqrt{a}\left(g - \frac{|E(S,T)|}{|T|\sqrt{a}}\right) \leq 0. \quad (2)$$

g is the maximum value the binary search can obtain when a is fixed. Then, we can derive the following lemma.

LEMMA 5.7. *Given a graph $G=(V,E)$ and a specific a, assume that S' and T' are the optimal choices for Equation (2). Let $b=\frac{|S'|}{|T'|}$ and $c=\frac{a^2}{b}$. Then, for any (S,T)-induced subgraph $G[S,T]$ of G, if $\min\{b,c\} \leq \frac{|S|}{|T|} \leq \max\{b,c\}$, we have $\rho(S,T) \leq \rho(S',T')$.*

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

A Divide-and-conquer Exact Algorithm

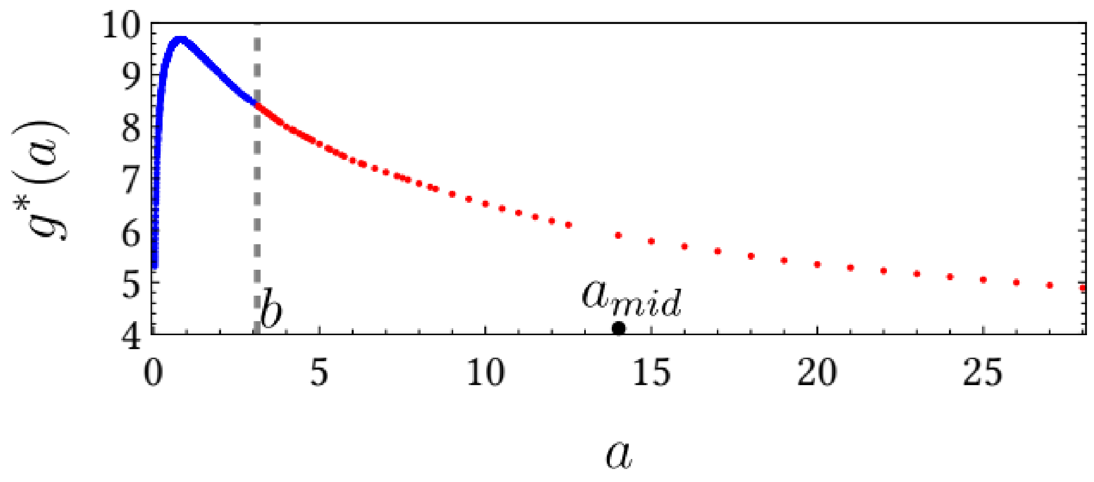


Figure 7: The prunning effectiveness of Lemma 5.7.

Example 5.8. We consider a small dataset MA [46] which consists of 28 vertices and 217 edges; this implies that the values of a are in the range $[\frac{1}{28}, 28]$. We plot the values of $g^*(a)$ for $a \in [\frac{1}{28}, 28]$ in Figure 7. Let $a_{mid}=(\frac{1}{28}+28)/2$. After applying the binary search for a_{mid}, we get a=14.02, b=3.125, and c=62.88, by Lemma 5.7. Therefore, we can skip the binary search for all the 78 values of $a \in (3.125, 62.88)$, which are marked in red in Figure 7. □

EXACT ALGORITHMS

Algorithm 5: DC-Exact

Input : G=(V, E)

Output : The exact DDS D=G[S^*, T^*]

$a_l \leftarrow \frac{1}{n}, a_r \leftarrow n, \rho^* \leftarrow 0, D \leftarrow \emptyset$;

Divide-Conquer(a_l, a_r);

return D;

Function Divide-Conquer(a_l, a_r):

- $a_{mid} \leftarrow \frac{a_l + a_r}{2}$;
- run Lines 4-13 of Algorithm 4 (replace $x \leftarrow \lceil \frac{l}{2\sqrt{a}} \rceil, y \leftarrow \lceil \frac{\sqrt{a}l}{2} \rceil$ with $x \leftarrow \lceil \frac{l}{2\sqrt{a_r}} \rceil$, $y \leftarrow \lceil \frac{\sqrt{a_l}l}{2} \rceil$);
- let $G[S', T']$ be the DDS found by binary search;
- $b \leftarrow \frac{|S'|}{|T'|}$;
- $c \leftarrow \frac{a_{mid}^2}{b}$;
- **if** $b > c$ **then** Swap(b, c);
- **if** $a_l \leq b$ **then** Divide-Conquer(a_l, b);
- **if** $c \leq a_r$ **then** Divide-Conquer(c, a_r);

- **time complexity:** $O(knmlogn)$

A CORE-BASED APPROXIMATION ALGORITHM

LEMMA 6.1 (LOWER BOUND OF DENSITY OF $[x, y]$-CORE). *Given a graph G and an $[x, y]$-core, denoted by H=$G[S, T]$, in G, the density of H is*

$$\rho(S, T) \geq \sqrt{xy}. \tag{6}$$

$$\boxed{\rho(S, T) = \frac{|E(S, T)|}{\sqrt{|S||T|}} = \sqrt{\frac{|E(S, T)|^2}{|S||T|}} \geq \sqrt{\frac{x|S| \cdot y|T|}{|S||T|}} = \sqrt{xy}.}$$

A CORE-BASED APPROXIMATION ALGORITHM

Definition 6.2 (Maximum cn-pair). Given a graph G=(V, E), a cn-pair [x, y] is called the **maximum cn-pair**, if $x \cdot y$ achieves the maximum value among all the possible [x, y]-cores. We denote the maximum cn-pair by [x^*, y^*].

LEMMA 6.3 (UPPER BOUND OF ρ^*). *Given a graph G=(V, E) and its maximum cn-pair [x^*, y^*], the density ρ^* of the DDS is*

$$\rho^* \leq 2\sqrt{x^* y^*}. \tag{7}$$

PROOF. We prove the lemma by contradiction. Assume that $\rho^* > 2\sqrt{x^* y^*}$. Let $a^* = \frac{|S^*|}{|T^*|}$. Then, by Theorem 5.6, we conclude that the DDS is in the [x', y']-core, where $x' > \frac{\sqrt{x^* y^*}}{\sqrt{a^*}}$ and $y' > \sqrt{a^*}\sqrt{x^* y^*}$, so $x'y' > x^* y^*$, which contradicts the fact that [x^*, y^*] is the maximum cn-pair of G. Therefore, ρ^* is at most $2\sqrt{x^* y^*}$. □

THEOREM 5.6. *Given a graph G=(V, E), the DDS D=G[S^*, T^*] is contained in the $\left[\lceil \frac{\rho^*}{2\sqrt{a}} \rceil, \lceil \frac{\sqrt{a}\rho^*}{2} \rceil\right]$-core, where $a = \frac{|S^*|}{|T^*|}$.*

A CORE-BASED APPROXIMATION ALGORITHM

THEOREM 6.4. *Given a graph G=(V, E), the core whose cn-pair is the maximum cn-pair, i.e., $[x^*, y^*]$-core, is a 2-approximation solution to the DDS problem.*

PROOF. Let the $[x^*, y^*]$-core be an (S, T)-induced subgraph. By Lemma 6.1, we have $\rho(S,T) \geq \sqrt{x^* y^*}$. According to Lemma 6.3, we have $\rho^* \leq 2\sqrt{x^* y^*}$, so we conclude

$$\frac{\rho^*}{\rho(S,T)} \leq \frac{2\sqrt{x^* y^*}}{\sqrt{x^* y^*}} = 2. \tag{8}$$

Hence, the theorem holds. □

A CORE-BASED APPROXIMATION ALGORITHM

Definition 6.5 (Maximum equal cn-pair). Given a graph G=(V, E), a cn-pair [x, x] is the **maximum equal cn-pair**, if x achives the maximum value among all the possible [x, x]-cores. We denote the maximum equal cn-pair by [γ, γ].

LEMMA 6.6. *Given a graph G=(V, E) and its maximum equal cn-pair* [γ, γ], *for any cn-pair* [x, y], *we have either $x \leq \gamma$ or $y \leq \gamma$, or both of them.*

PROOF. We prove this lemma by contradiction. Assume there is a cn-pair [x, y] where $x > \gamma$ and $y > \gamma$. Then, let $\gamma' = \min\{x, y\} > \gamma$, so there exists a [$\gamma'$, γ']-core in G, which contradicts [γ, γ] is the maximum equal cn-pair. □

A CORE-BASED APPROXIMATION ALGORITHM

Definition 6.7 (Key cn-pair). Given a graph G=(V, E) and its maximum equal cn-pair $[\gamma, \gamma]$, the cn-pair of an $[x, y]$-core is a **key cn-pair**, if one of the following conditions is satisfied:

(1) if $x \leq \gamma$, there does not exist any $[x, y']$-core in G, such that $y' > y$;

(2) if $y \leq \gamma$, there does not exist any $[x', y]$-core in G, such that $x' > x$.

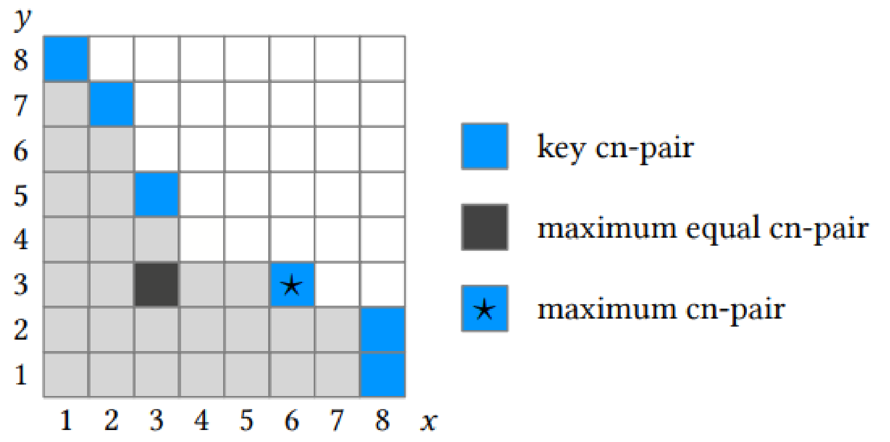


Figure 8: Illustrating the concepts of cn-pairs.

A CORE-BASED APPROXIMATION ALGORITHM

LEMMA 6.9. *Given a graph $G=(V, E)$ and its maximum equal cn-pair $[\gamma, \gamma]$, we have $\gamma \leq \sqrt{m}$.*

LEMMA 6.10. *Given a graph $G=(V, E)$, there are at most $2\sqrt{m}$ key cn-pairs in G.*

PROOF. According to Lemma 6.6 and Definition 6.7, there are at most 2γ key cn-pairs in G. Since we have $\gamma \leq \sqrt{m}$ by Lemma 6.9, there are at most $2\sqrt{m}$ key cn-pairs in G. □

A CORE-BASED APPROXIMATION ALGORITHM

Algorithm 6: Core-Approx

Input : G=(V, E)

Output : An approximate DDS $\widetilde{D}$, i.e., the $[x^*, y^*]$-core

1 $x^* \leftarrow 0$, $y^* \leftarrow 0$;

2 $[\gamma, \gamma] \leftarrow$ compute the $[\gamma, \gamma]$-core by iteratively peeling vertices which have the minimum indegrees or outdegrees;

3 **for** $x \leftarrow 1$ **to** γ **do**

4 $y \leftarrow$GetMaxY(G, x);

5 **if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;

6 **for** $y \leftarrow 1$ **to** γ **do**

7 $x \leftarrow$GetMaxX(G, y);

8 **if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;

9 **return** compute the $[x^*, y^*]$-core;

10 **Function** GetMaxY(G, x):

11 $S \leftarrow V, T \leftarrow V, y_{\max} \leftarrow 0, y \leftarrow \lfloor \frac{x^* y^*}{x} \rfloor + 1$;

12 **if** $y > \max_{u \in T}\{d_G^+(u)\}$ **then return** $y_{\max}$;

13 **while** $|E| > 0$ **do**

14   **while** $\exists u \in T, d_G^+(u) < y$ **do**

15    $E \leftarrow E \setminus \{(v, u) | v \in S\}, T \leftarrow T \setminus \{u\}$;

16   **while** $\exists v \in S, d_G^-(v) < x$ **do**

17     $E \leftarrow E \setminus \{(v, u) | u \in T\}, S \leftarrow S \setminus \{v\}$;

18   **if** $|E| > 0$ **then** $y_{\max} \leftarrow y$;

19   $y \leftarrow y + 1$;

20 **return** $y_{\max}$;

21 **Function** GetMaxX(G, y):

22 reuse lines 11-20 by interchanging u with v, S with T, x with y, and changing $y_{\max}$ to $x_{\max}$;

A CORE-BASED APPROXIMATION ALGORITHM

Algorithm 6: Core-Approx

Input : $G=(V,E)$

Output: An approximate DDS $\widetilde{D}$, i.e., the $[x^*, y^*]$-core

1 $x^* \leftarrow 0, y^* \leftarrow 0$;
2 $[\gamma,\gamma] \leftarrow$ compute the $[\gamma,\gamma]$-core by iteratively peeling vertices which have the minimum indegrees or outdegrees;
3 **for** $x \leftarrow 1$ **to** γ **do**
4 $\quad y \leftarrow$GetMaxY(G, x);
5 $\quad$**if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;
6 **for** $y \leftarrow 1$ **to** γ **do**
7 $\quad x \leftarrow$GetMaxX(G, y);
8 $\quad$**if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;
9 **return** compute the $[x^*, y^*]$-core;
10 **Function** GetMaxY(G, x):
11 $\quad S \leftarrow V, T \leftarrow V, y_{\max} \leftarrow 0, y \leftarrow \lfloor \frac{x^*y^*}{x} \rfloor + 1$;
12 $\quad$**if** $y > \max_{u \in T}\{d_G^+(u)\}$ **then return** $y_{\max}$;
13 $\quad$**while** $|E| > 0$ **do**
14 $\quad\quad$**while** $\exists u \in T, d_G^+(u) < y$ **do**
15 $\quad\quad\quad E \leftarrow E \setminus \{(v,u)|v \in S\}, T \leftarrow T \setminus \{u\}$;
16 $\quad\quad\quad$**while** $\exists v \in S, d_G^-(v) < x$ **do**
17 $\quad\quad\quad\quad E \leftarrow E \setminus \{(v,u)|u \in T\}, S \leftarrow S \setminus \{v\}$;
18 $\quad\quad$**if** $|E| > 0$ **then** $y_{\max} \leftarrow y$;
19 $\quad\quad y \leftarrow y + 1$;
20 $\quad$**return** $y_{\max}$;
21 **Function** GetMaxX(G, y):
22 $\quad$reuse lines 11-20 by interchanging u with v, S with T, x with y, and changing $y_{\max}$ to $x_{\max}$;

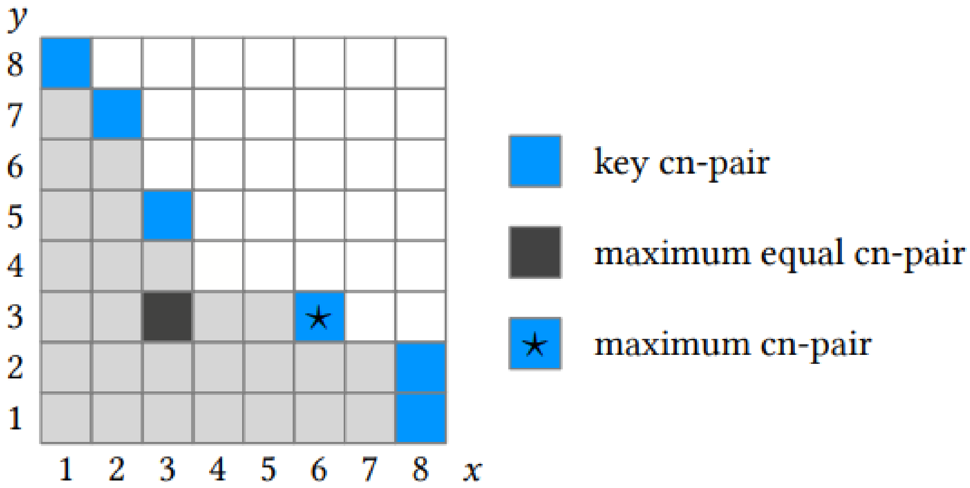


Figure 8: Illustrating the concepts of cn-pairs.

- **time complexity:** $O(\sqrt{m}(n+m))$

Evaluation

Table 4: Datasets used in our experiments.

Dataset	Category	$\|V\|$	$\|E\|$
MO [21]	Human Social	217	2,672
TC [2]	Infrastructure	1,226	2,615
OF [42]	Infrastructure	2,939	30,501
AD [38]	Social	6,541	51,127
AM [35]	E-commerce	403,394	3,387,388
AR [40]	E-commerce	3,376,972	5,838,041
BA [41]	Hyperlink	2,141,300	17,794,839
TW [9]	Social	52,579,682	1,963,263,821

Setup

- Exact [32] is the state-of-the-art exact algorithm, which is also recapped in Section 4.1.
- KS-Approx [32] is an approximation algorithm whose appproximation ratio was misclaimed, which is also recapped in Section 4.2.
- FKS-Approx [1] is the fixed version provided by the authors of [32]. Its time complexity is $O(n \cdot (n+m))$, with an approximation ratio of 2.
- PM-Approx [4] is a parameterized approximation algorithm. Note that we use its default parameters in [4] in our experiments (δ=2, ϵ=1).
- BS-Approx [10] is a 2-approximation algorithm.

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

[4] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment* 5, 5 (2012), 454–465.

[1] 2019. Supplementary Note. https://i.cs.hku.hk/~chma2/sup-sigmod2020.pdf. (2019).

[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

Exact Algorithms

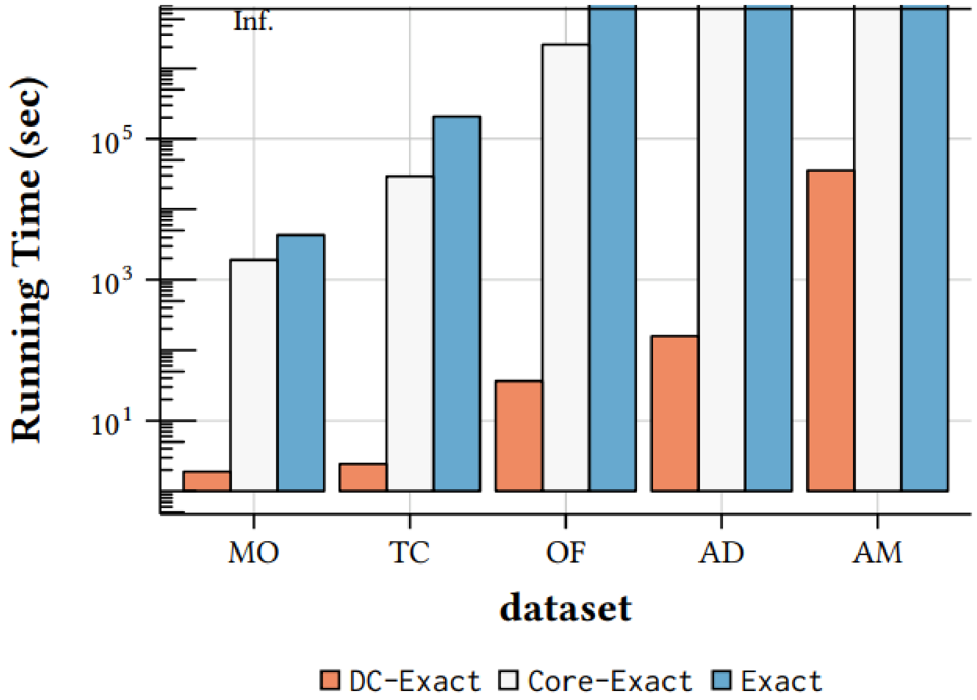


Figure 9: Efficiency of exact algorithms.

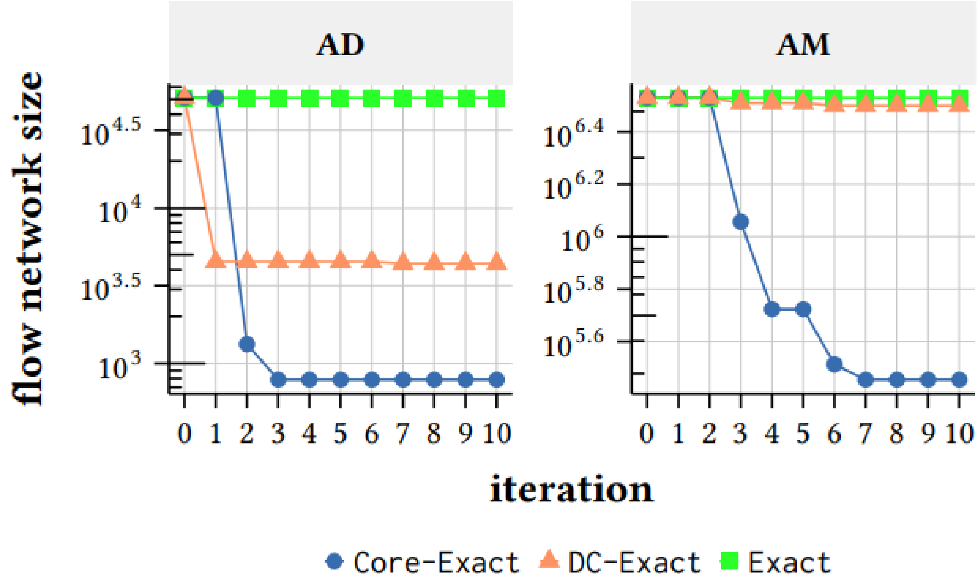


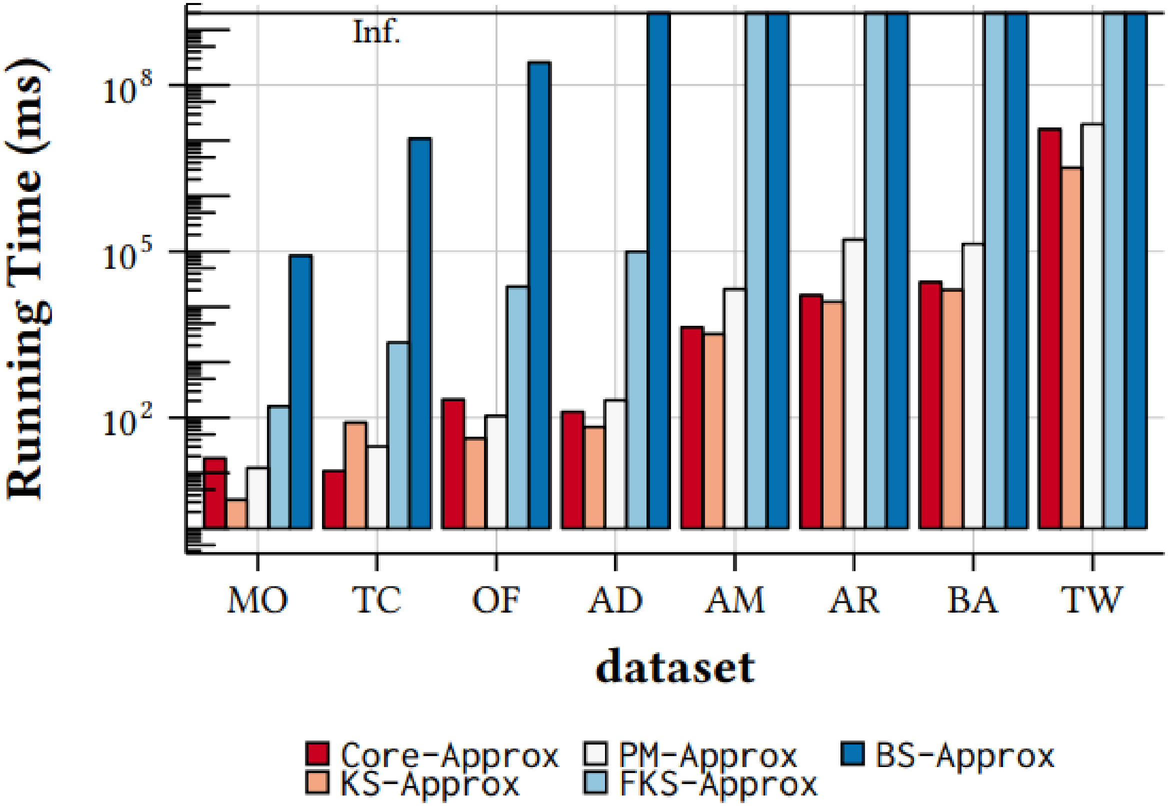
Figure 10: Flow network sizes in exact algorithms.

Exact Algorithms

Table 5: The total numbers of values of a examined in `DC-Exact` and `Exact`.

Dataset	n^2 (Exact)	k (DC-Exact)	$\frac{n^2}{k}$
MO	4.71×10^4	16	2.94×10^3
TC	1.50×10^6	23	6.54×10^4
OF	8.64×10^6	35	2.47×10^5
AD	4.28×10^7	81	5.28×10^5
AM	1.63×10^{11}	13	1.25×10^{10}

Approximation Algorithms



Approximation Algorithms

Table 6: Analyzing 2-approximation algorithms.

Dataset	MO	TC	OF	AD	AM	AR	BA	TW
n	217	1,226	2,939	6,541	4.03×10^5	3.38×10^6	2.14×10^6	5.26×10^7
n^2	4.71×10^4	1.50×10^6	8.64×10^6	4.28×10^7	1.63×10^{11}	1.14×10^{13}	4.59×10^{12}	2.76×10^{15}
δ	8	3	27	18	10	26	60	2221
$\frac{n}{\delta}$	27	408	108	363	4.03×10^4	1.30×10^5	3.57×10^4	2.37×10^4
$\frac{n^2}{\delta}$	5.89×10^3	5.01×10^5	3.20×10^5	2.38×10^6	1.63×10^{10}	4.39×10^{11}	7.64×10^{10}	1.24×10^{12}

Approximation Algorithms

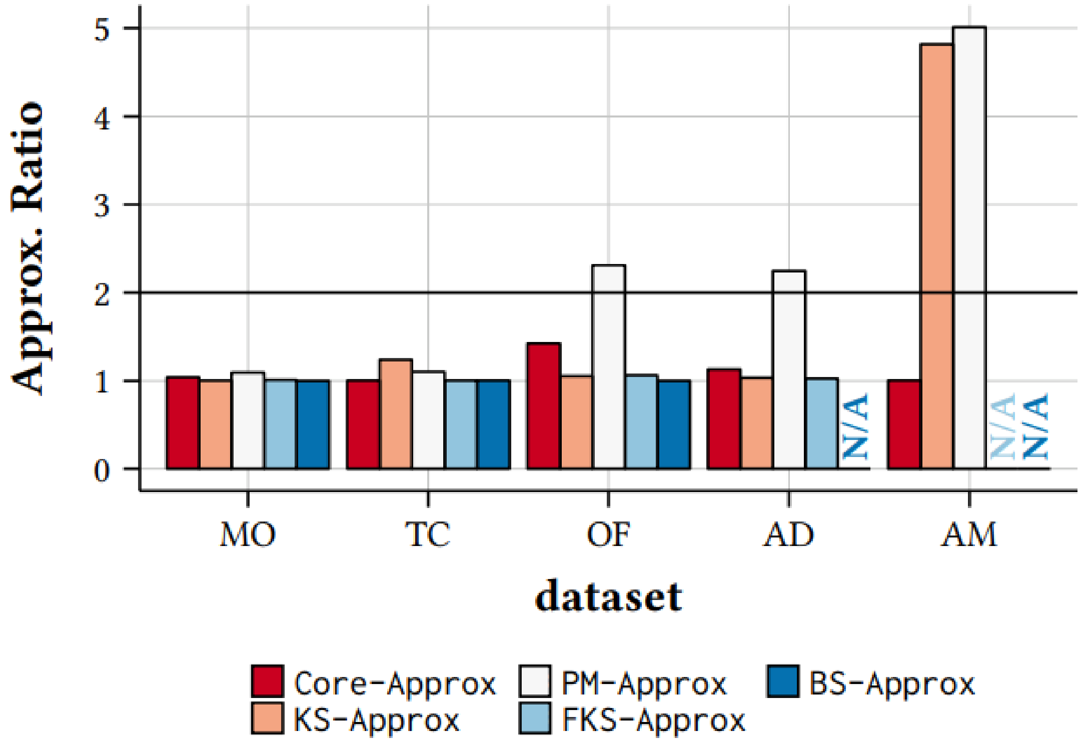


Figure 12: The actual approximation ratios of all the five approximation algorithms.

Table 7: The values of $\frac{|S^*|}{|T^*|}$ on the first five datasets.

Dataset	MO	TC	OF	AD	AM
$\frac{|S^*|}{|T^*|}$	1.04	5.67×10^{-2}	1.01	2.32	2.47×10^{3}

Comparing DC-Exact and Core-Approx

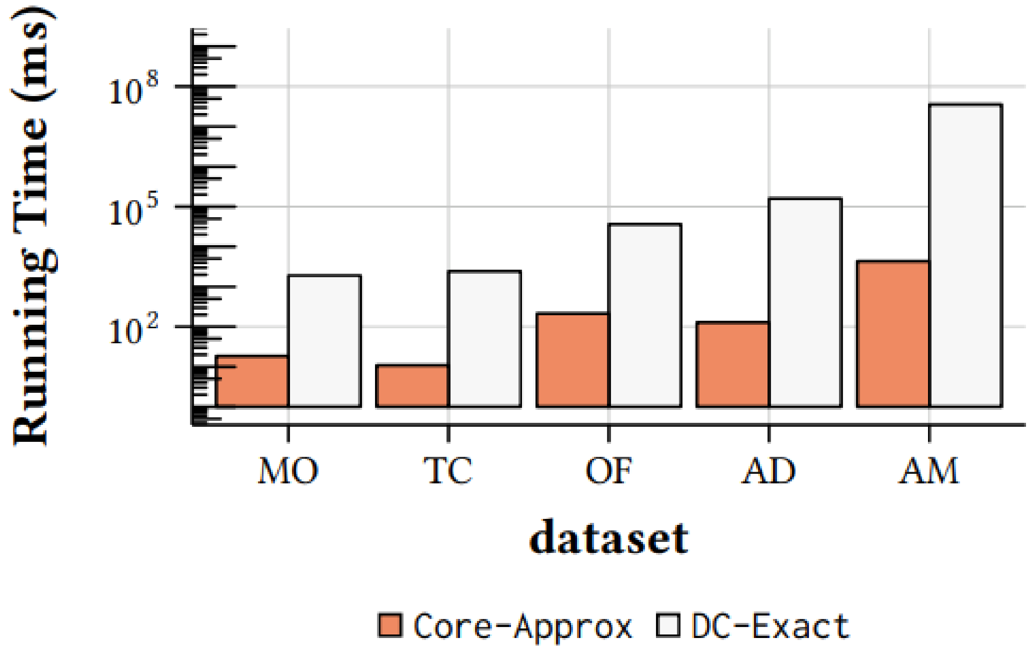


Figure 13: Efficiency of Core-Approx **and** DC-Exact.

Conclusion

- They first review existing algorithms and discuss their limitations.
- To boost the efficiency of finding DDS, they introduce a novel dense subgraph model, namely $[x, y] - core$, on directed graphs, and establish bounds on the density of the $[x, y] - core$.
- They then propose a core-based exact algorithm, and further optimize it by incorporating a divide-and-conquer strategy.
- Besides, they find that the $[x^*, y^*] - core$, where x^*y^* is the maximum value of xy for all the $[x, y] - cores$, is a good approximation solution to the DDS problem, with theoretical guarantee.
- Extensive experiments show that both exact and approximation algorithms are up to six orders of magnitude faster than state-of-the-art approaches.

THANK YOU

Xiaojia Xu