Fast Algorithms for the Maximum Clique Problem on Massive Sparse Graphs

Bharath Pattabiraman (Northwestern)
Mostofa Patwary (Northwestern)
Assefaw Gebremedhin (Purdue)
Wei-keng Liao (Northwestern)
Alok Choudhary (Northwestern)

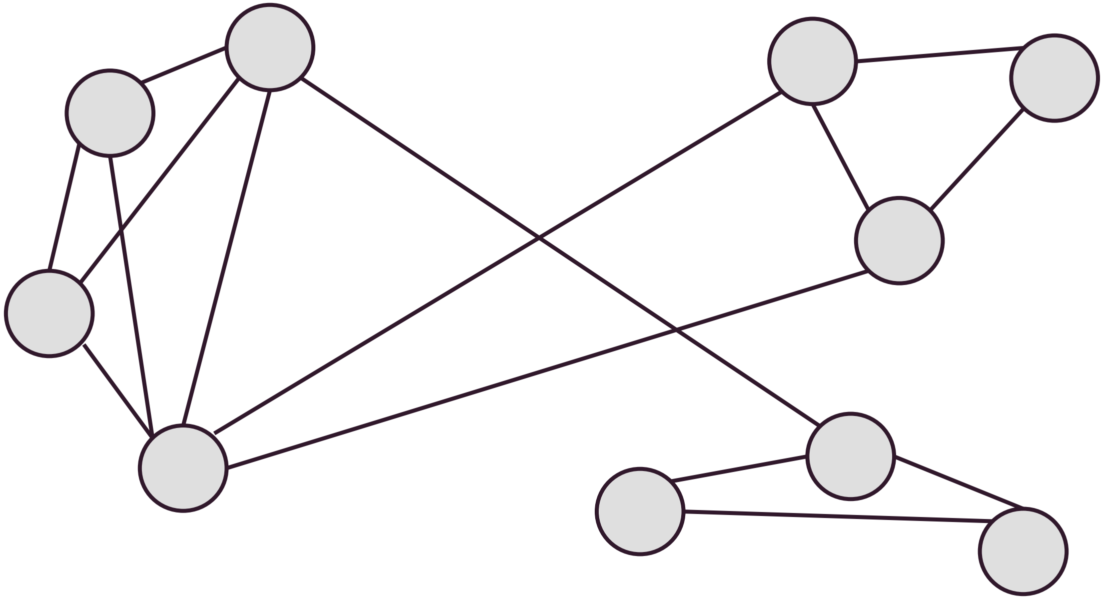
Outline

- Introduction
- Motivation
- Existing Algorithms
- New Algorithm
- Performance Comparison
- Future Work

Clique Problem

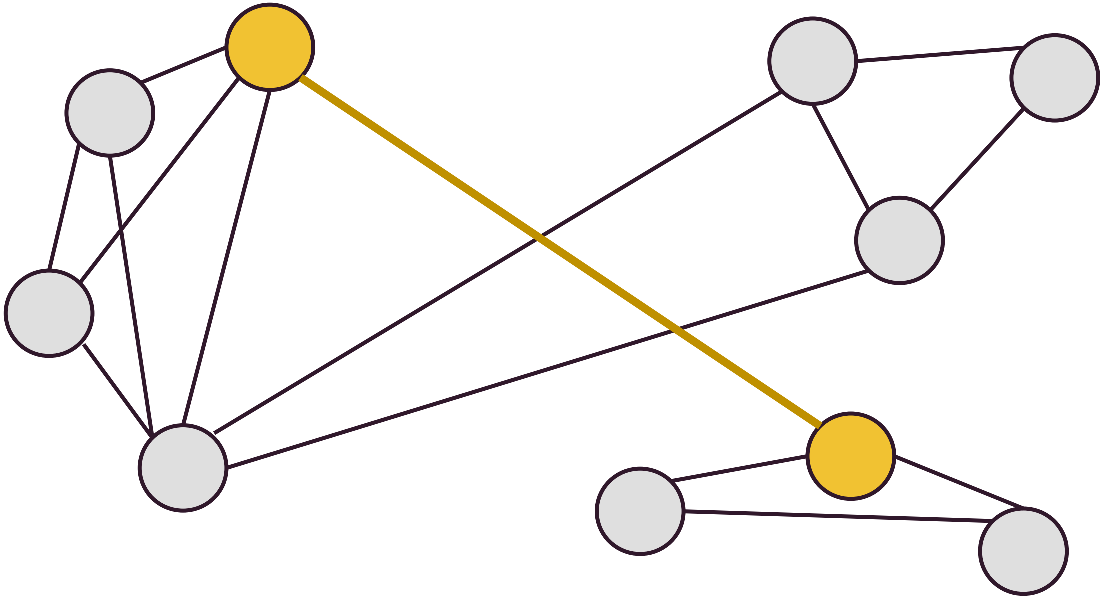
- $G = (V,E)$ is an undirected graph
- ***Clique*** - a subset of V such that every node is connected to every other in the subset
- ***Maximal Clique*** - a clique that cannot be enlarged by adding more vertices i.e. one that is not a subset of a larger clique
- ***Maximum Clique*** - the (maximal) clique with largest number of vertices

Clique Problem



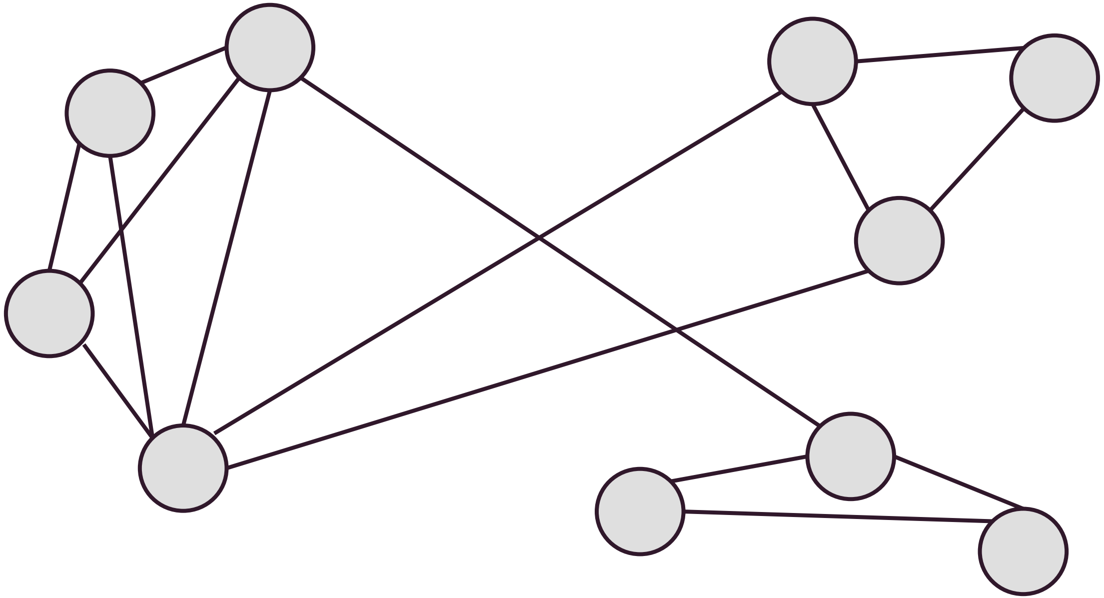
- Cliques of size 2 ?
  - every connected pair of vertices
- Maximal cliques of size 2 ?

Clique Problem



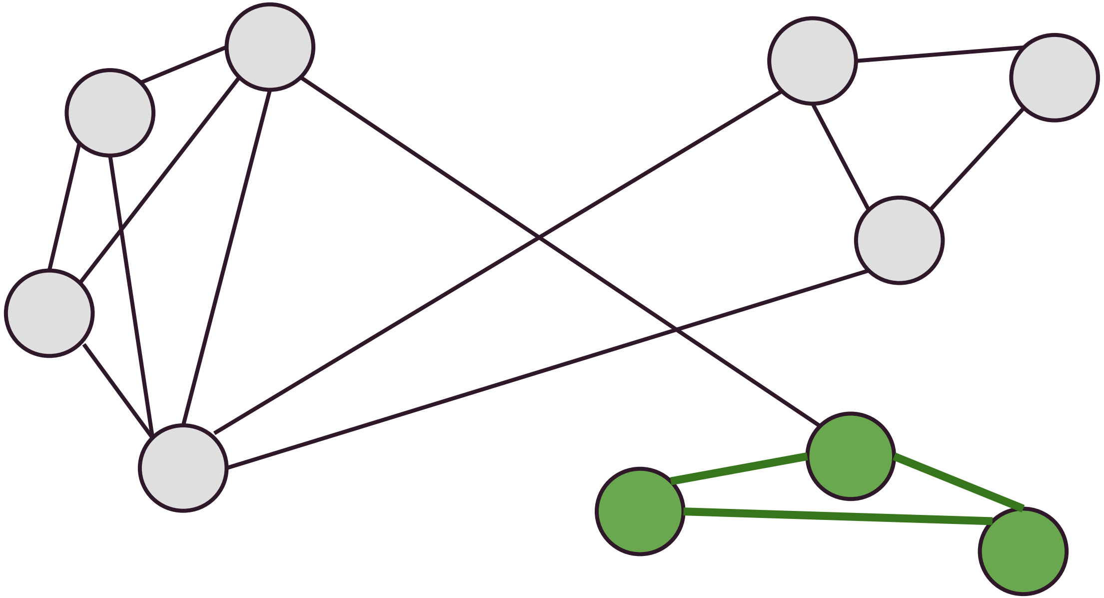
- Cliques of size 2 ?
  - every connected pair of vertices
- Maximal cliques of size 2 ?

Clique Problem



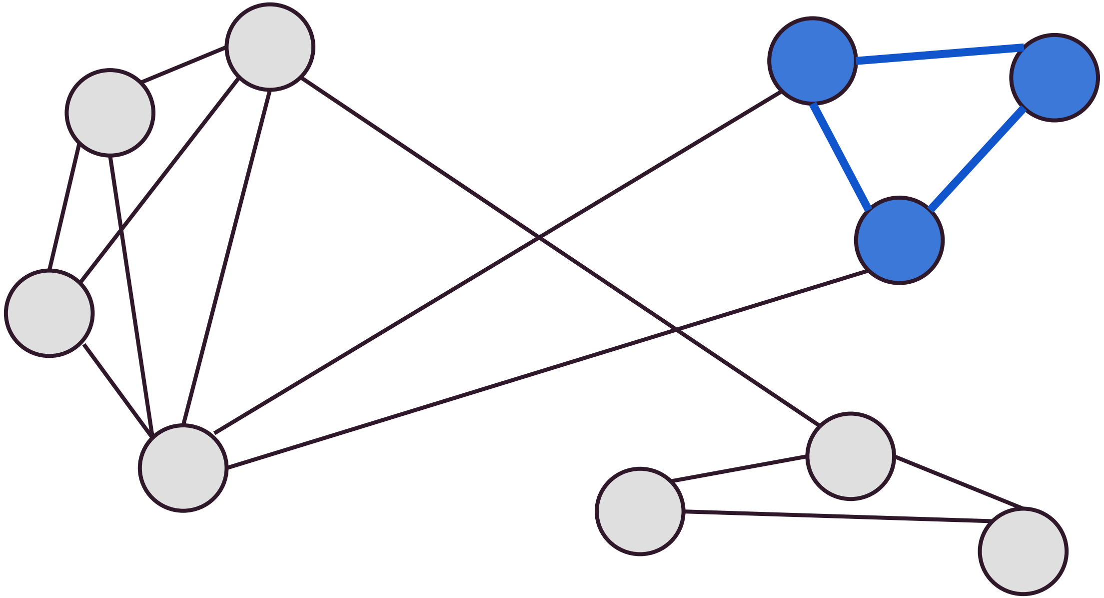
- Maximal cliques of size 3 ?

Clique Problem



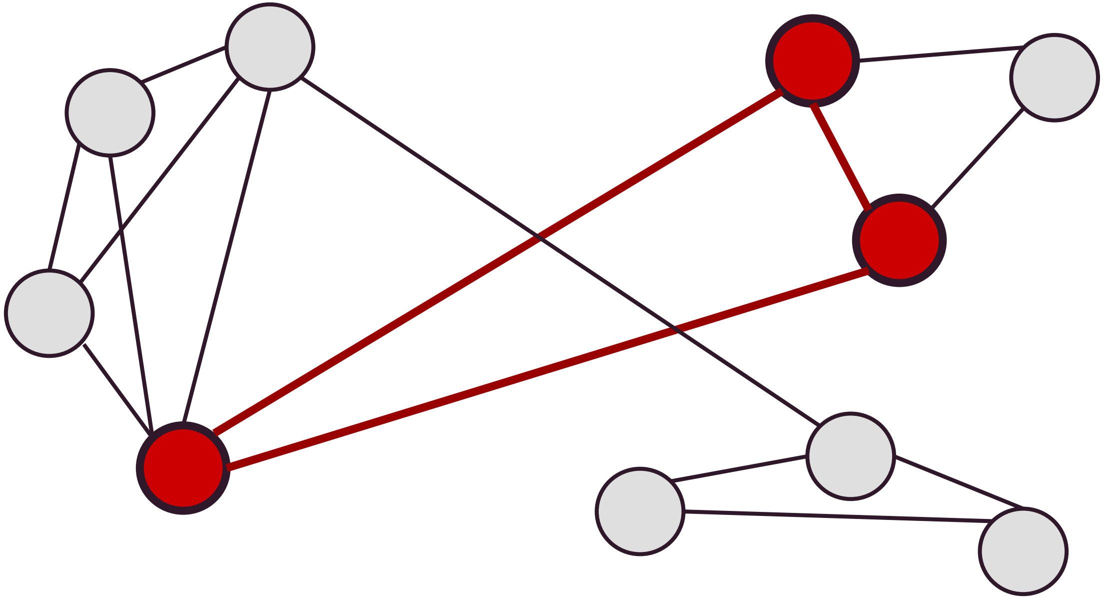
- Maximal cliques of size 3 ?

Clique Problem



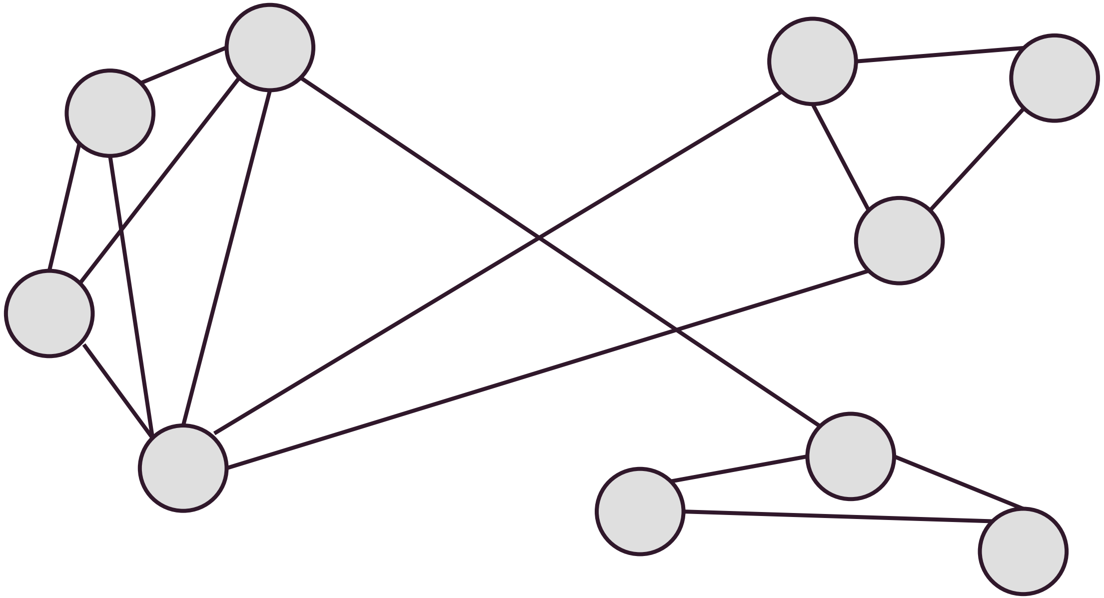
- Maximal cliques of size 3 ?

Clique Problem



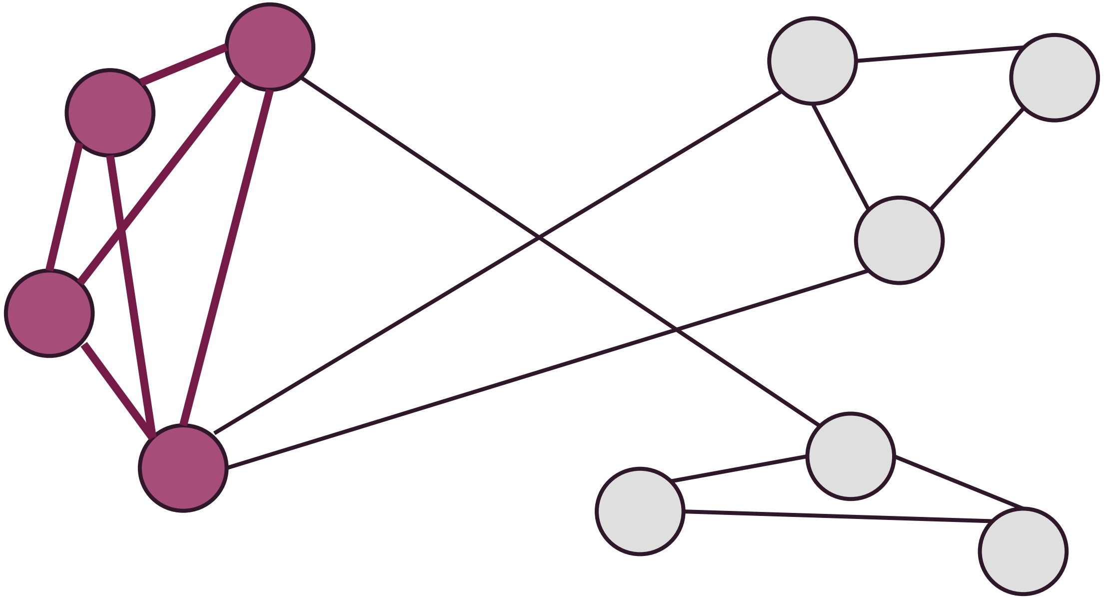
- Maximal cliques of size 3 ?

Clique Problem



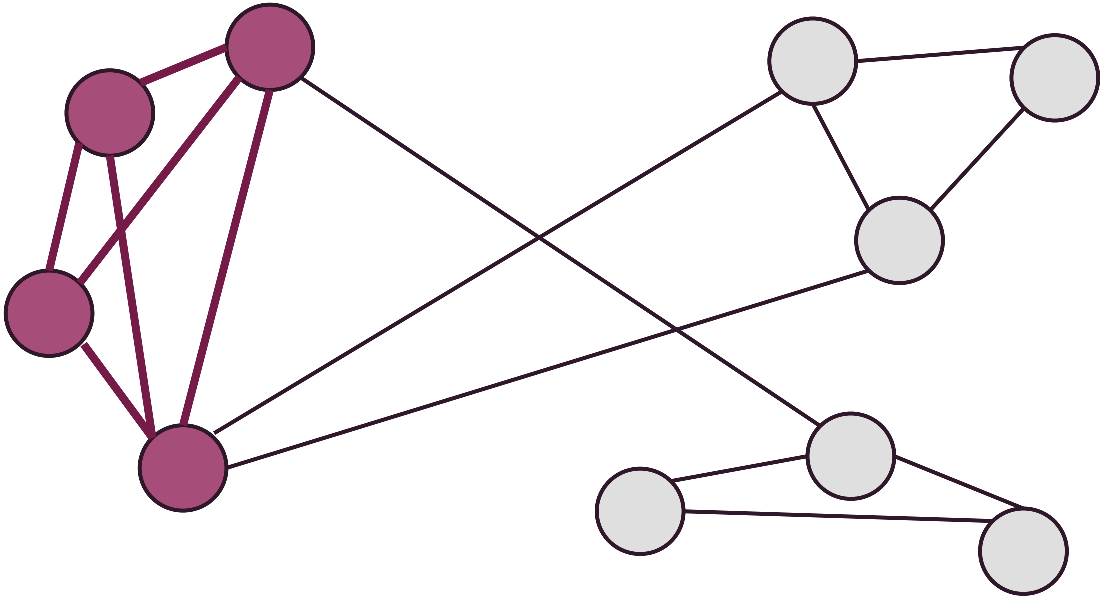
- Maximal cliques of size 4 ?

Clique Problem



- Maximal cliques of size 4 ?

Clique Problem



- Maximal cliques of size 4 ?
- Also the **maximum** clique

Applications

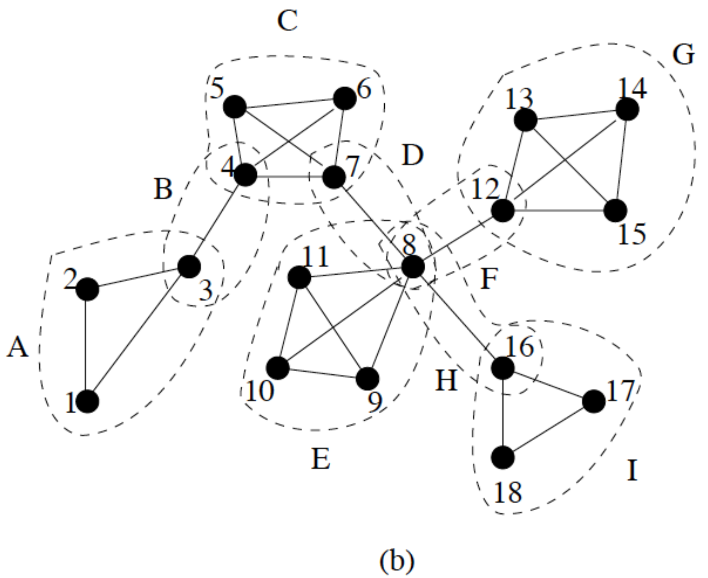


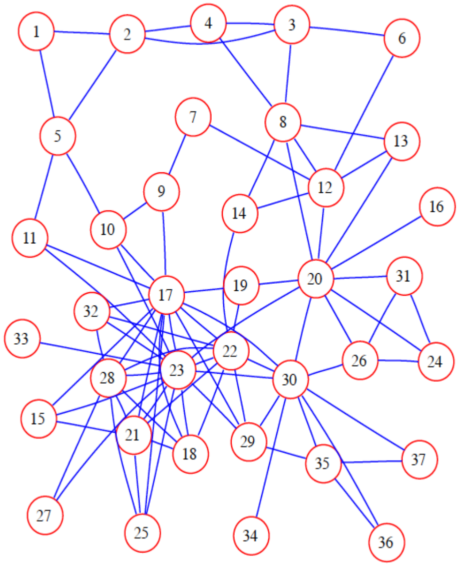
Figure 1: An Example of Clusters

Krishna, P., Vaidya, N., Chatterjee, M., Pradhan, D.: A cluster-based approach for routing in dynamic networks. In: ACM SIGCOMM Computer Communication Review, pp. 49–65 (1997)

- Clustering

Applications

1 Wail Alshehri
2 Satam Suqami
3 Nabil al-Marabh
4 Raed Hijazi
5 Waleed Alshehri
6 Ahmed Alghamdi
7 Mohand Alshehri
8 Saeed Alghamdi
9 Fayez Ahmed
10 Mustafa Ahmed Al-Hisawi
11 Abdul Aziz Al-Omari
12 Hamza Alghamdi
13 Ahmed Alnami
14 Ahmed Al Haznawi
15 Mamoun Darkazanli
16 Mohamed Abdi
17 Marwan Al-Shehhi
18 Zakariya Essabar
19 Salem Alhazmi



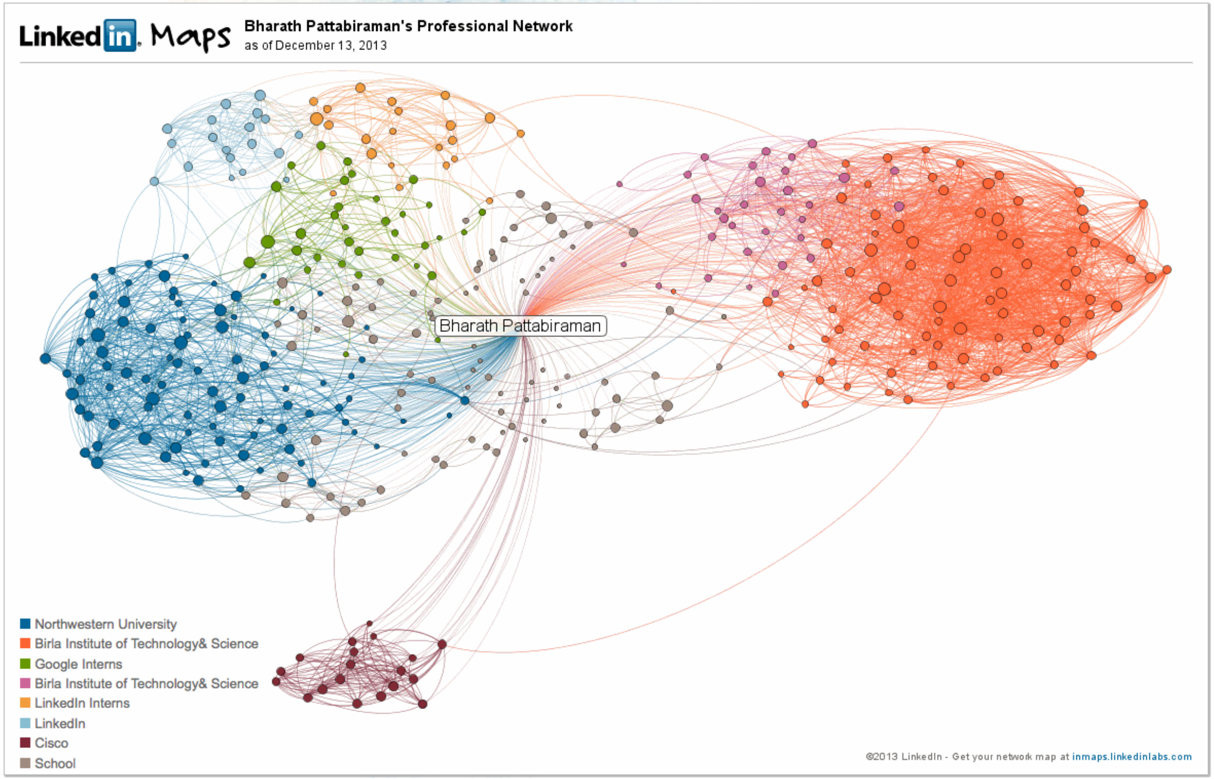
20 Nawaf Alhazmi
21 Said Bahaji
22 Ziad Jarrah
23 Mohamed Atta
24 Abdussattar Shaikh
25 Mounir El Motassadeq
26 Khalid Al-Mihdhar
27 Zacarias Moussaoui
28 Ramzi Bin al-Shibh
29 Lofti Raissi
30 Hani Hanjour
31 Osama Awadallah
32 Agus Budiman
33 Ahmed Khalil Ibrahim Samir Al-Ani
34 Majed Moqed
35 Rayed Mohammed Abdullah
36 Faisal Al Salmi
37 Bandar Alhazmi

Fig. 1 The network surrounding the tragic events of September 11, 2001.

- Clustering
- Social Network Analysis

Balabhaskar Balasundaram, Sergiy Butenko, Illya V. Hicks Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem

Applications



- Clustering
- Social Network Analysis

http://inmaps.linkedinlabs.com

Applications

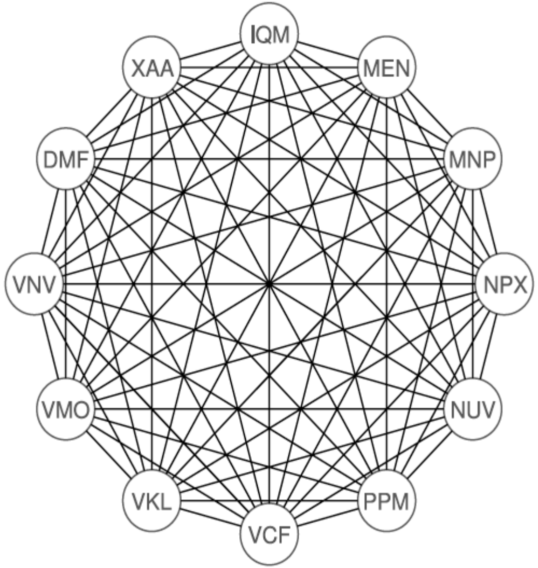


Fig. 1. The maximum closed clique in the stock market database with correlation coefficient threshold 0.90 and minimum relative support threshold 100%.

- Clustering
- Social Network Analysis
- Financial Network Analysis

Out-of-Core Coherent Closed Quasi-Clique Mining from Large Dense Graph Databases
ZHIPING ZENG, JIANYONG WANG, and LIZHU ZHOU GEORGE KARYPIS

Applications

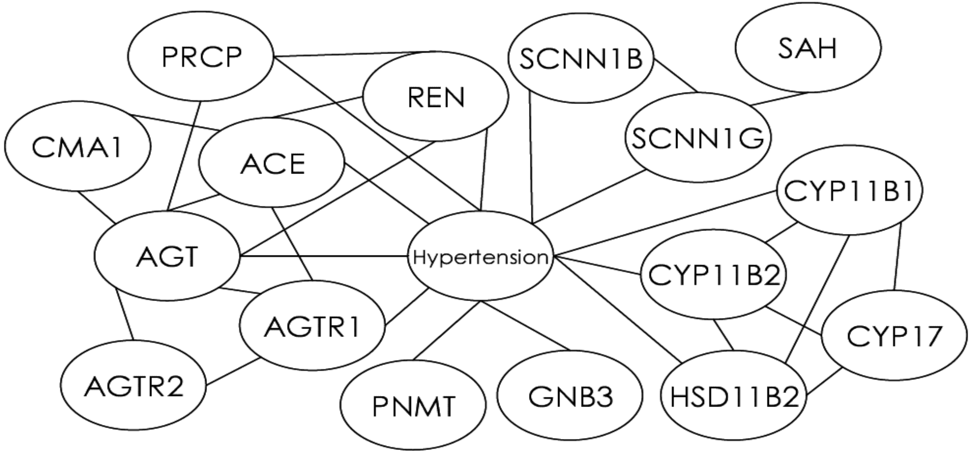


Figure 1
Example of a biomedical relational graph. Hypertension and hypertension-related genes are represented by nodes, and the associations between them are represented by edges.

Clique-based data mining for related genes in a biomedical database
Tsutomu Matsunaga*[1], Chikara Yonemori[1], Etsuji Tomita[2,3] and
Masaaki Muramatsu[4,5]

- Clustering
- Social Network Analysis
- Financial Network Analysis
- Biomedical data analysis and Bioinformatics

Algorithms

- Maximum clique problem
  - NP-complete
  - Still infeasible for large instances
  - Practical tricks to obtain acceptable runtimes
  - Heuristic approaches

Related Work

- Branch and bound algorithms
  - enumerate all candidate solutions, discard fruitless candidates (a.k.a **pruning**) using estimated upper bounds of the max clique size
  - Carraghan and Pardalos 1990
  - Ostergard 2002
  - Tomita and Seki 2003 - MCQ (vertex coloring as upper bound)
  - Konc and Janezic 2007 - MCQD (improved MCQ)

Related Work

- Base Algorithm of most published work
  - Carraghan and Pardalos 1990
  - Branch and Bound algorithm
  - Variant of depth first search on each vertex
  - Store the size of largest clique encountered, and use for pruning fruitless candidates

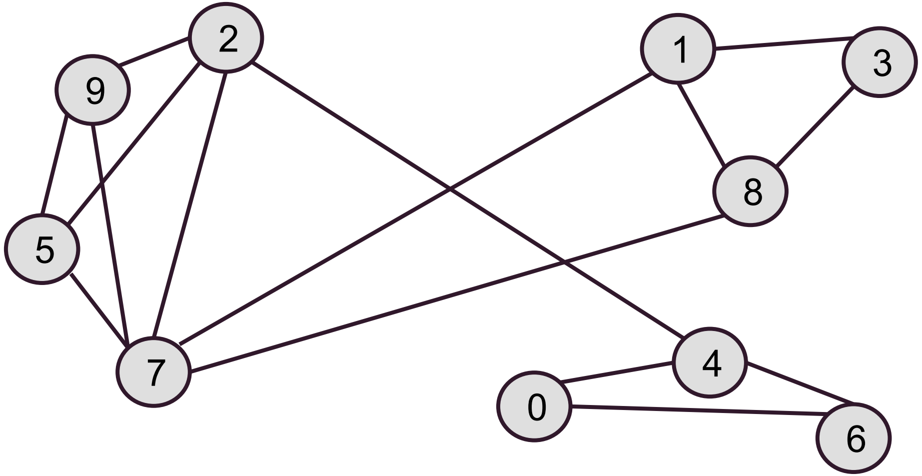
Pardalos Algorithm

```
function clique(U, size)
    if |U| = 0 then
        if size > max then
            max := size
            New record; save it.
        end if
        return
    end if
    while U ≠ ∅ do
        i := min{j | v_j ∈ U}
        U := U \ {v_i}
        clique(U ∩ N(v_i), size + 1)
    end while
    return
function old
    max := 0
    clique(V, 0)
    return
```

- pick a vertex from the candidate list
- add it to current clique
- updated candidate list = intersection of current candidate list and neighbors of added vertex
- recurse until all cliques are examined

Pardalos Algorithm

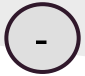
|V| = 10, |E| = 15



Max clique size = 0
Max clique set = {}

Current clique size = 0
Current clique set = {}
Current Node = ---
Current Neighbors = ---
Candidate set = {0,1,...,9}

Recursion Tree



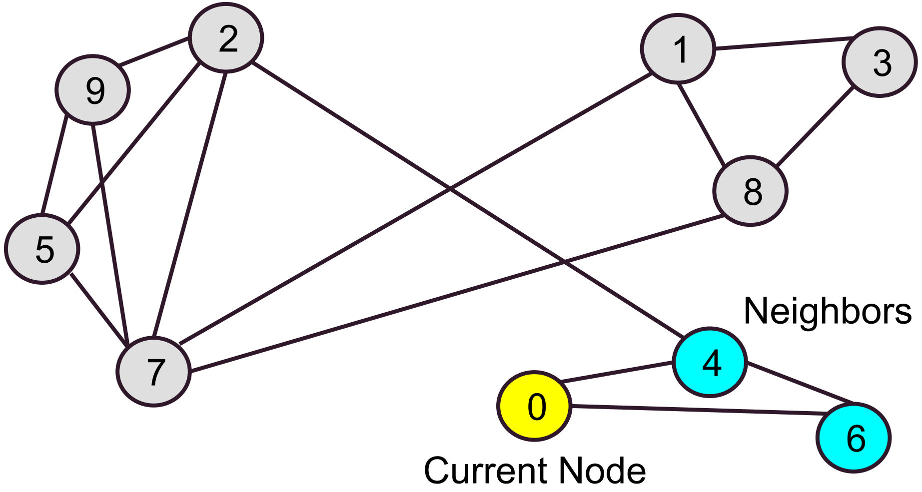
Current Step:

Next Step:
Pick Current Node
Update Current Neighbors

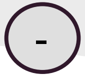
Current Clique

Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree



Max clique size = 0
Max clique set = {}

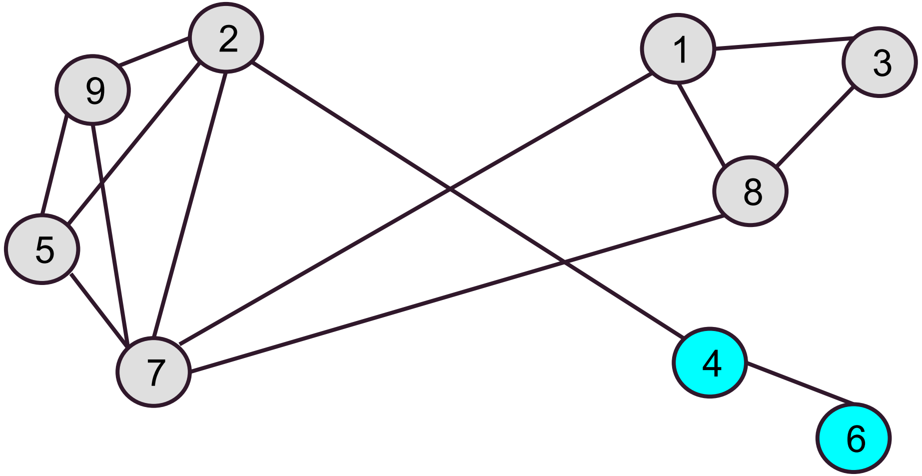
Current clique size = 0
Current clique set = {}
Current Node = Node 0
Current Neighbors = {4,6}
Candidate set = {0,1,...,9}

Current Step:
Pick Current Node
Update Current Neighbors
Next Step:
Update clique set and size
Update Candidate set

Current Clique

Pardalos Algorithm

|V| = 10, |E| = 15



Max clique size = 0
Max clique set = {}

Current clique size = 1
Current clique set = {0}
Current Node = ---
Current Neighbors = ---
Candidate set = {4,6}

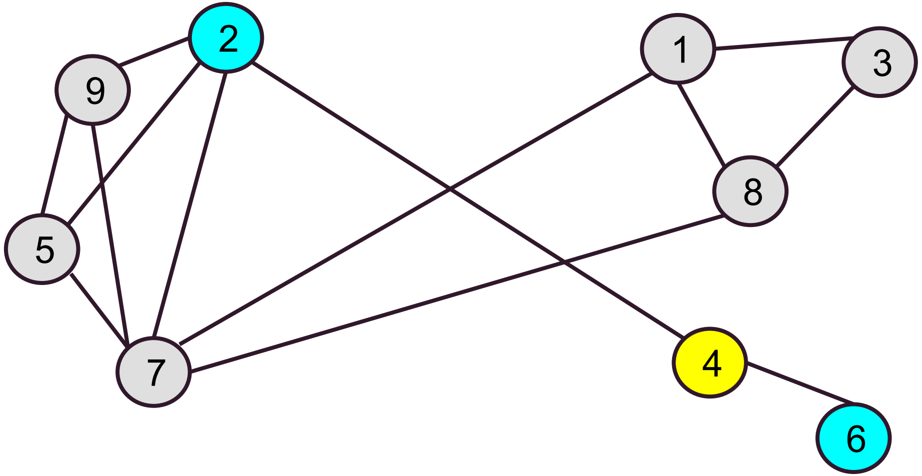
Recursion Tree

<u>Current Step</u>:
Update clique set and size
Update Candidate set
<u>Next Step</u>:
Pick Current Node
Update Current Neighbors

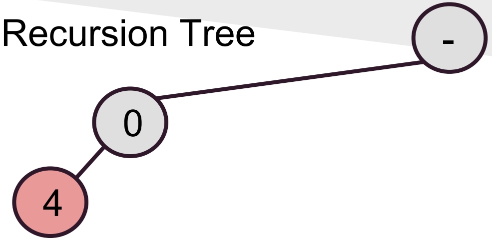
Current Clique

Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree



Max clique size = 0
Max clique set = {}

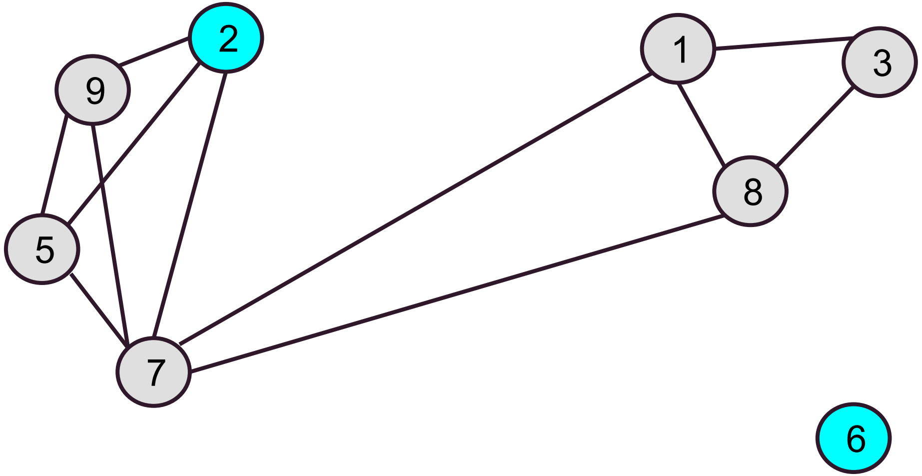
Current clique size = 1
Current clique set = {0}
Current Node = Node 4
Current Neighbors = {2,6}
Candidate set = {4,6}

Current Step:
Pick Current Node
Update Current Neighbors
Next Step:
Update clique set and size
Update Candidate set

Current Clique

Pardalos Algorithm

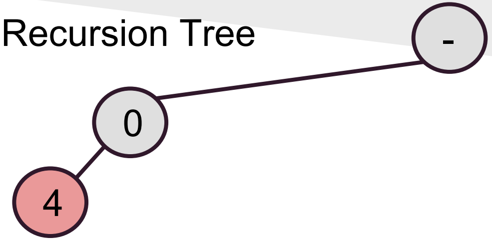
|V| = 10, |E| = 15



Max clique size = 0
Max clique set = {}

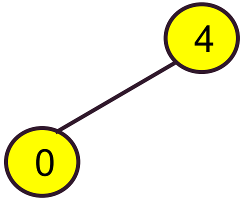
Current clique size = 2
Current clique set = {0,4}
Current Node = ---
Current Neighbors = ---
Candidate set = {6}

Recursion Tree



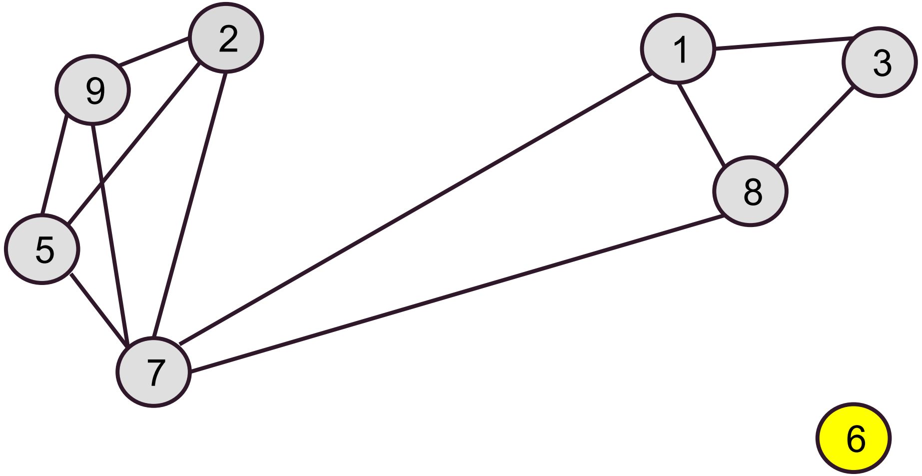
Current Step:
Update clique set and size
Update Candidate set
Next Step:
Pick Current Node
Update Current Neighbors

Current Clique



Pardalos Algorithm

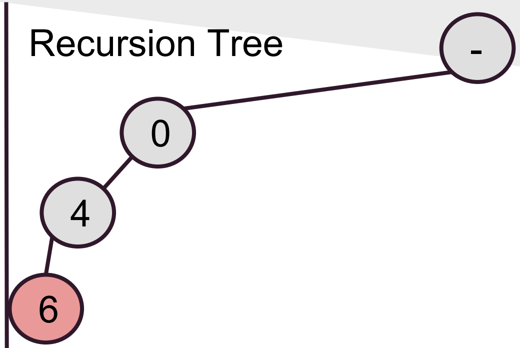
|V| = 10, |E| = 15



Max clique size = 0
Max clique set = {}

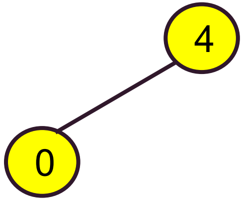
Current clique size = 2
Current clique set = {0,4}
Current Node = Node 6
Current Neighbors = {}
Candidate set = {6}

Recursion Tree



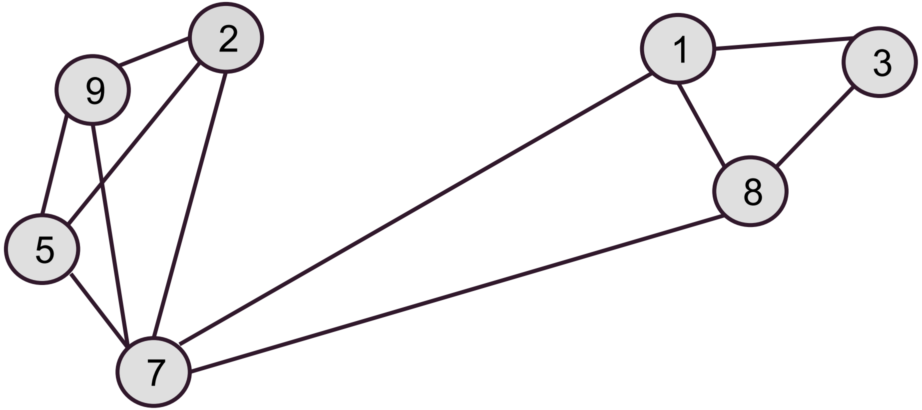
<u>Current Step</u>:
Pick Current Node
Update Current Neighbors
<u>Next Step</u>:
Update clique set and size
Update Candidate set

Current Clique

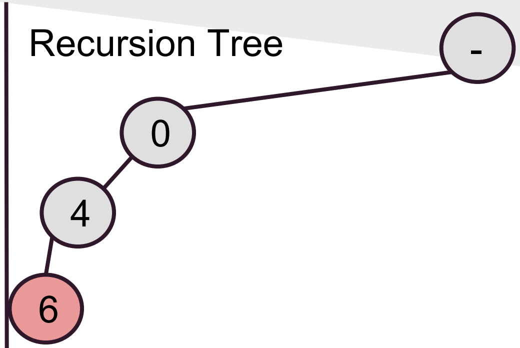


Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree

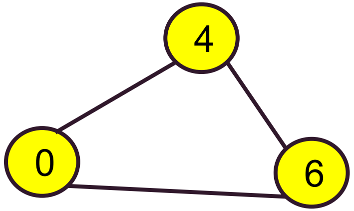


Max clique size = 0
Max clique set = {}

Current clique size = 3
Current clique set = {0,4,6}
Current Node = ---
Current Neighbors = {}
Candidate set = {}

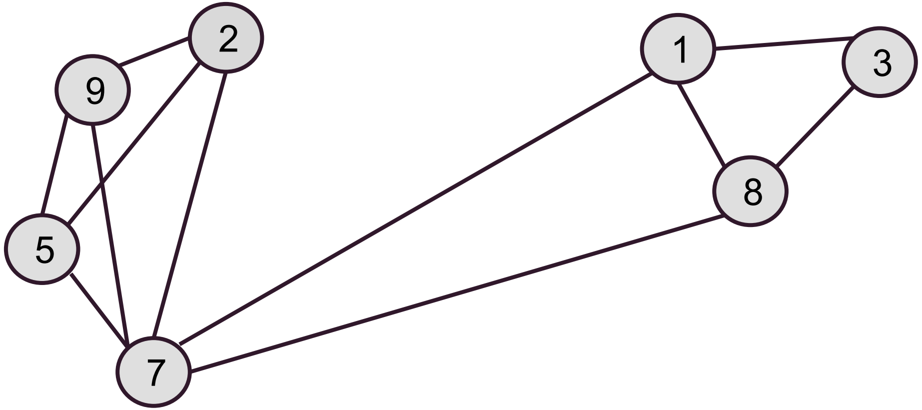
<u>Current Step</u>:
Update clique set and size
Update Candidate set
<u>Next Step</u>:
Update Max Clique

Current Clique

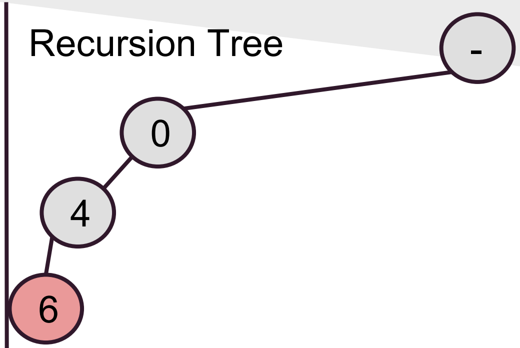


Pardalos Algorithm

|V| = 10, |E| = 15



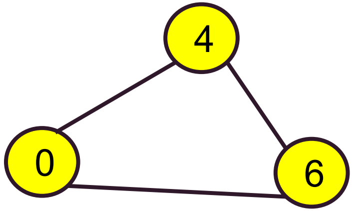
Recursion Tree



Current Step:
Update Max Clique

Next Step:

Current Clique

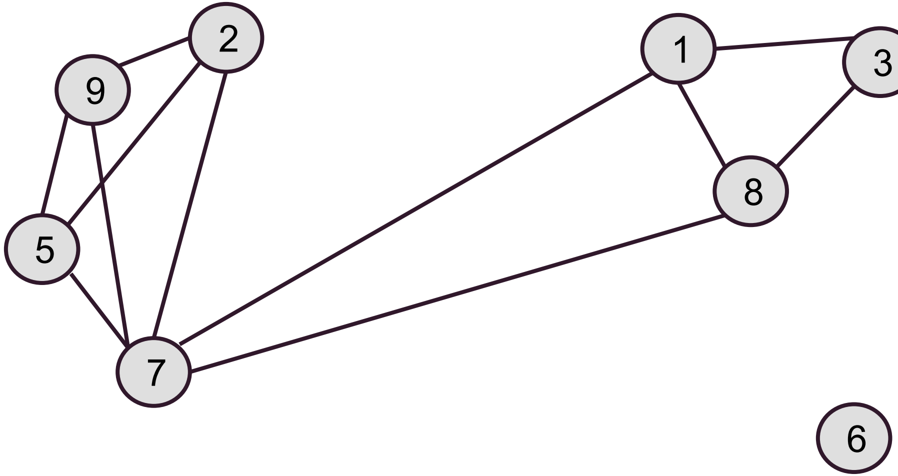


Max clique size = 3
Max clique set = {0,4,6}

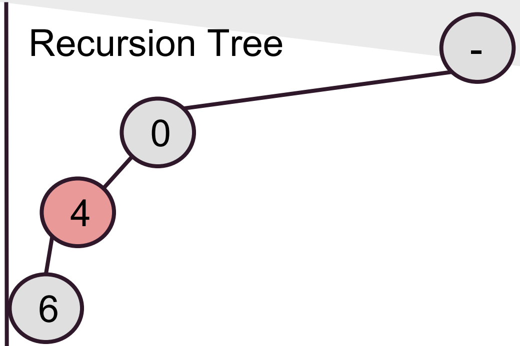
Current clique size = 3
Current clique set = {0,4,6}
Current Node = ---
Current Neighbors = {}
Candidate set = {}

Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree



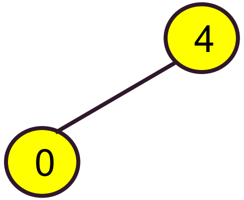
Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 0
Current clique set = {}
Current Node = ---
Current Neighbors = ---
Candidate set = {6}

Current Step:

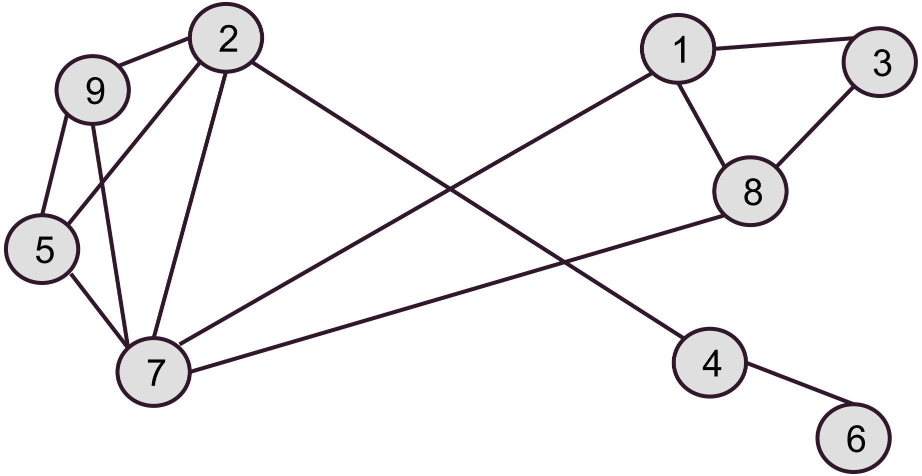
Next Step:

Current Clique



Pardalos Algorithm

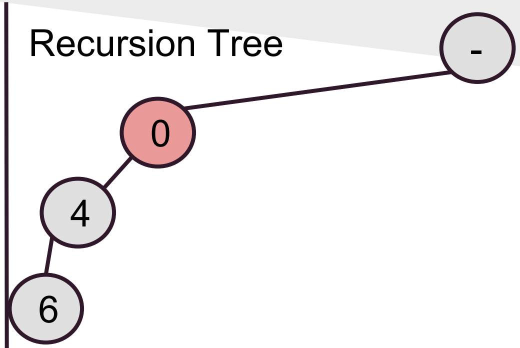
|V| = 10, |E| = 15



Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 0
Current clique set = {}
Current Node = ---
Current Neighbors = ---
Candidate set = {4,6}

Recursion Tree



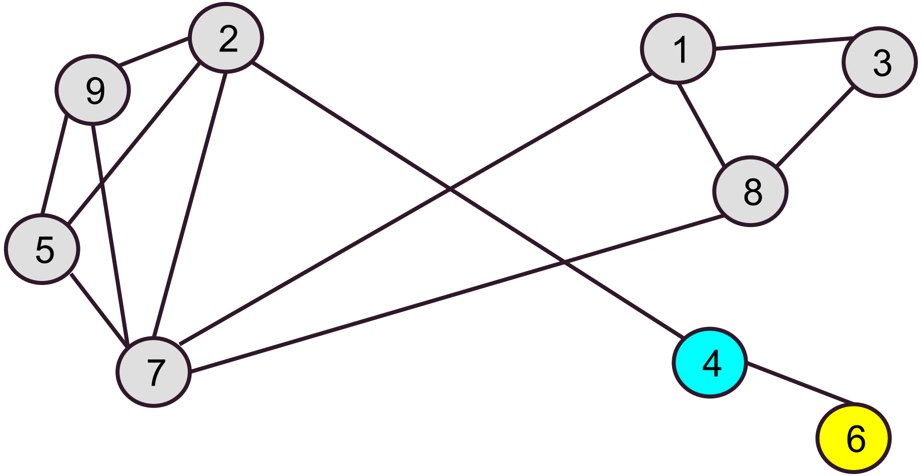
Current Step:

Next Step:

Current Clique

Pardalos Algorithm

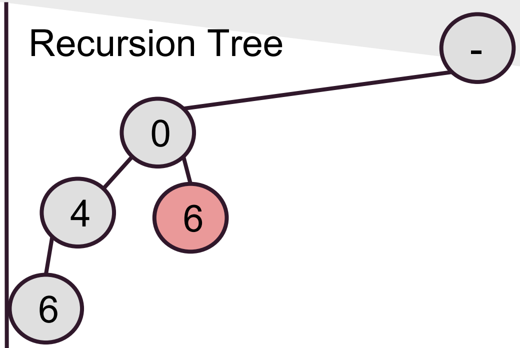
|V| = 10, |E| = 15



Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 1
Current clique set = {0}
Current Node = Node 6
Current Neighbors = {4}
Candidate set = {4,6}

Recursion Tree

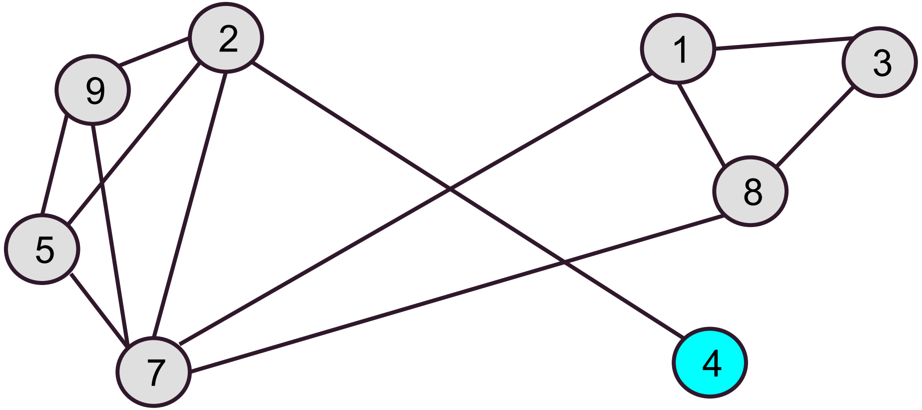


<u>Current Step</u>:
Pick Current Node
Update Current Neighbors
<u>Next Step</u>:
Update clique set and size
Update Candidate set

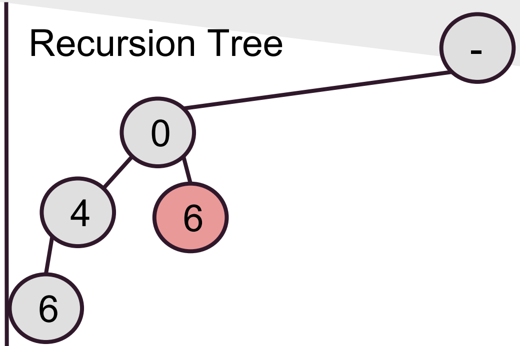
Current Clique

Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree



Max clique size = 3
Max clique set = {0,4,6}

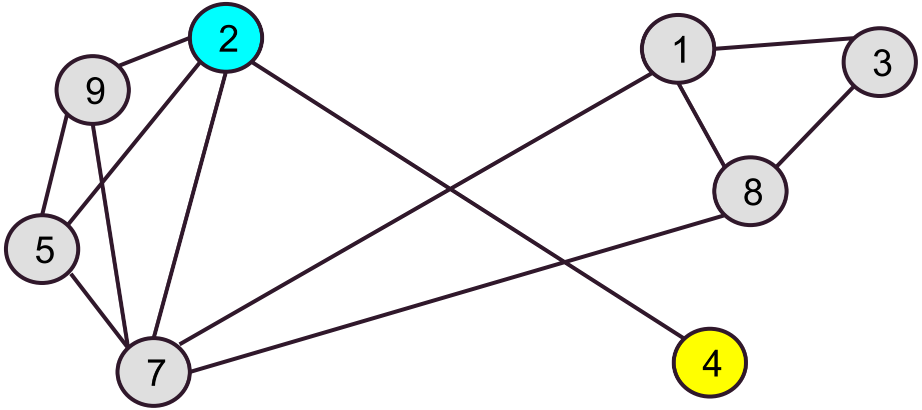
Current clique size = 2
Current clique set = {0,6}
Current Node = ---
Current Neighbors = ---
Candidate set = {4}

<u>Current Step</u>:
Update clique set and size
Update Candidate set
<u>Next Step</u>:
Pick Current Node
Update Current Neighbors

Current Clique

Pardalos Algorithm

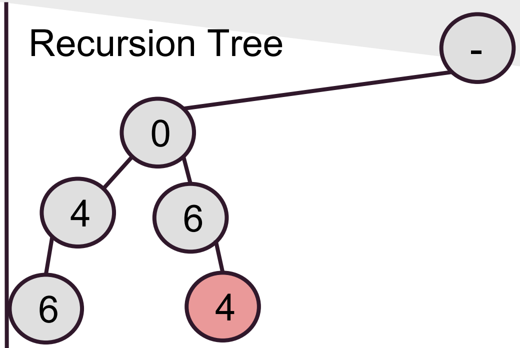
|V| = 10, |E| = 15



Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 2
Current clique set = {0,6}
Current Node = Node 4
Current Neighbors = {2}
Candidate set = {4}

Recursion Tree

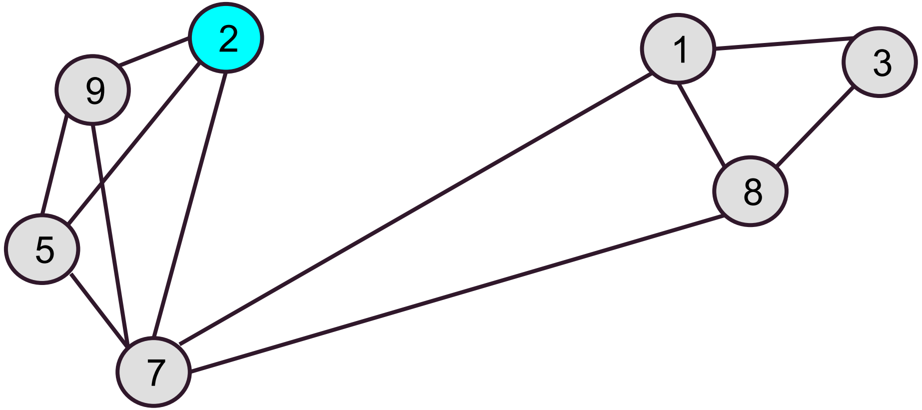


<u>Current Step</u>:
Pick Current Node
Update Current Neighbors
<u>Next Step</u>:
Update clique set and size
Update Candidate set

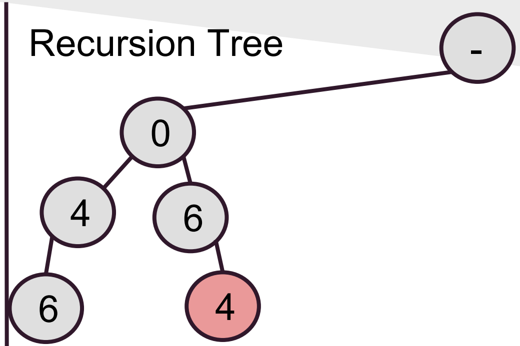
Current Clique

Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree

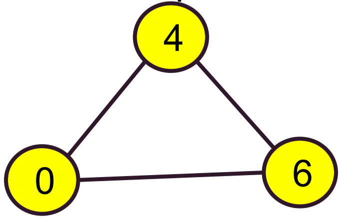


Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 3
Current clique set = {0,6,4}
Current Node = ---
Current Neighbors = ---
Candidate set = {}

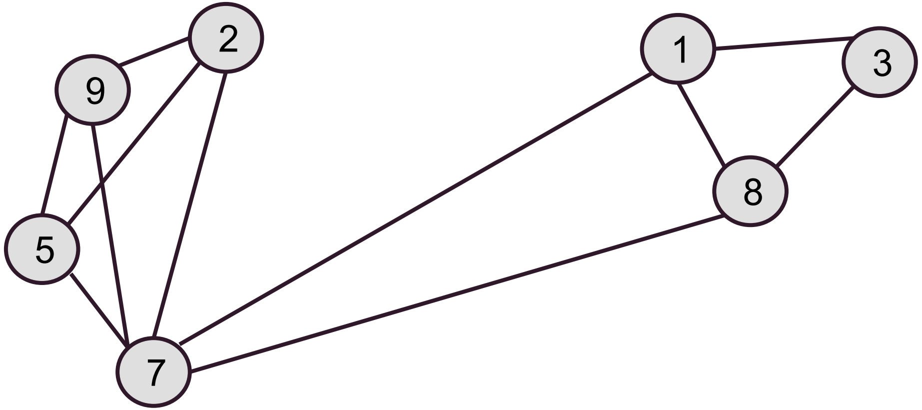
Current Step:
Pick Current Node
Update Current Neighbors
Next Step:
Update Max Clique

Current Clique

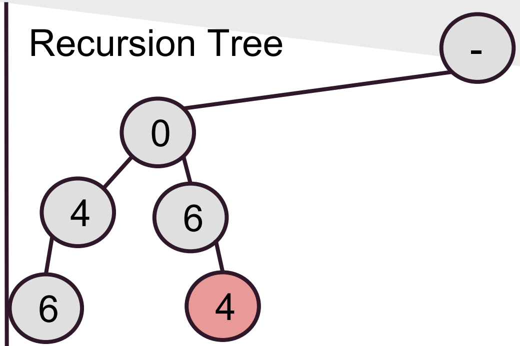


Pardalos Algorithm

|V| = 10, |E| = 15



Recursion Tree



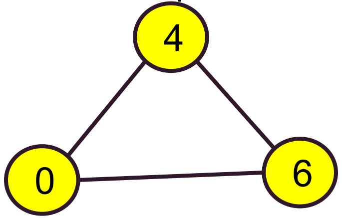
Max clique size = 3
Max clique set = {0,4,6}

Current clique size = 3
Current clique set = {0,6,4}
Current Node = ---
Current Neighbors = ---
Candidate set = {}

<u>Current Step</u>:
Update Max Clique

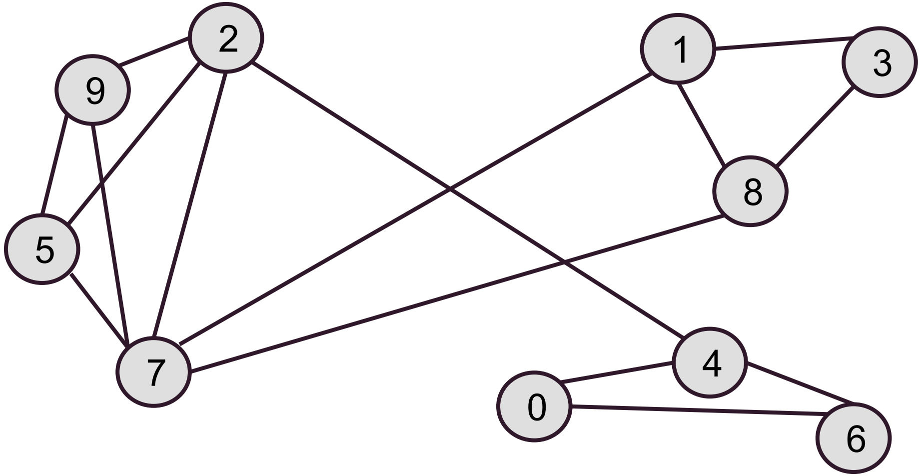
<u>Next Step</u>:

Current Clique



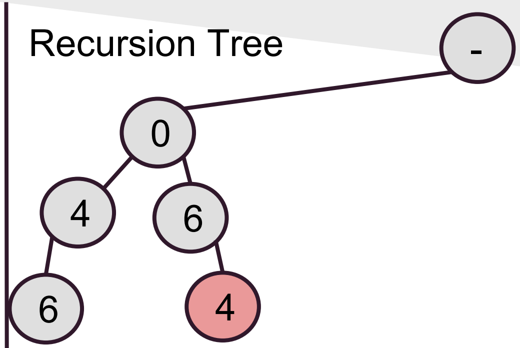
Pardalos Algorithm

|V| = 10, |E| = 15



Skipping...

Recursion Tree



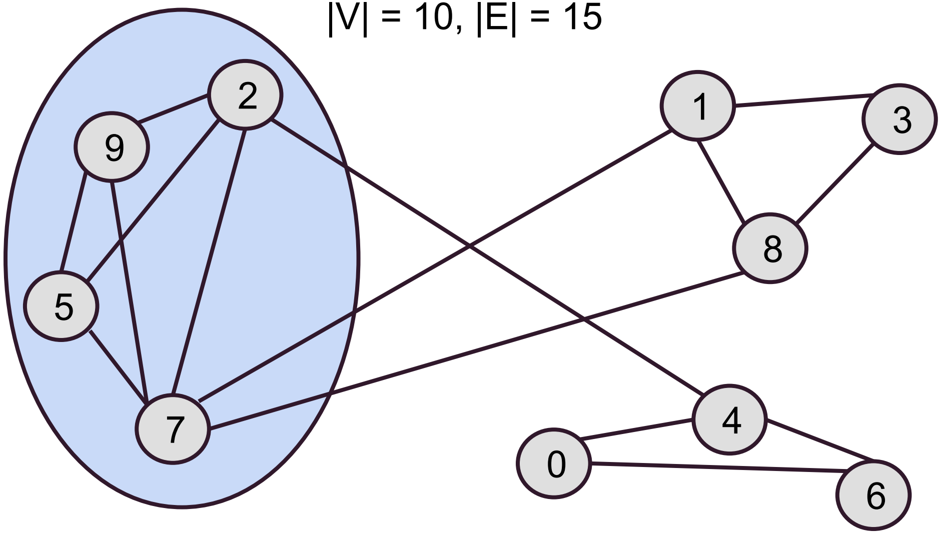
<u>Current Step</u>:

<u>Next Step</u>:

Current Clique

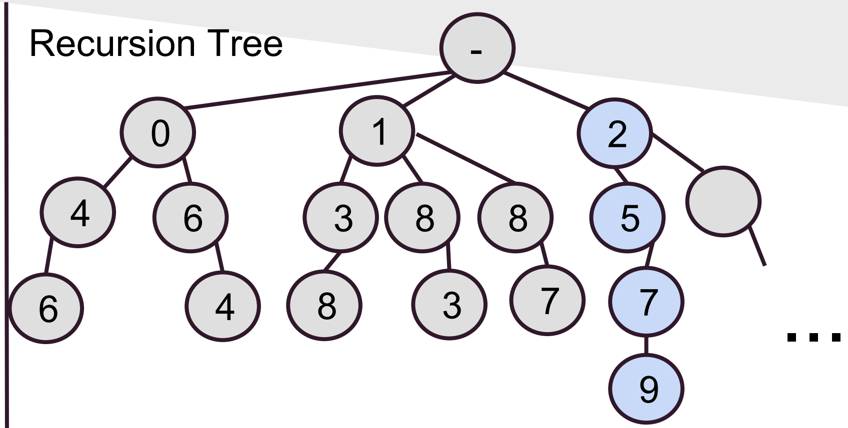
Pardalos Algorithm

|V| = 10, |E| = 15



Max clique size = 4
Max clique set = {2,5,7,9}

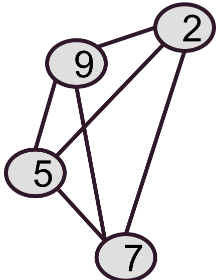
Recursion Tree



Current Step:

Next Step:

Current Clique



Modified Pardalos Algorithm

procedure $\text{MAXCLIQUE}(G = (V, E), lb)$
    $max \leftarrow lb$ ignore for now
    for i : 1 to n **do**
        $U \leftarrow \emptyset$
        for each $v_j \in N(v_i)$ **do**
            $U \leftarrow U \cup \{v_j\}$
        $\text{CLIQUE}(G, U, 1)$

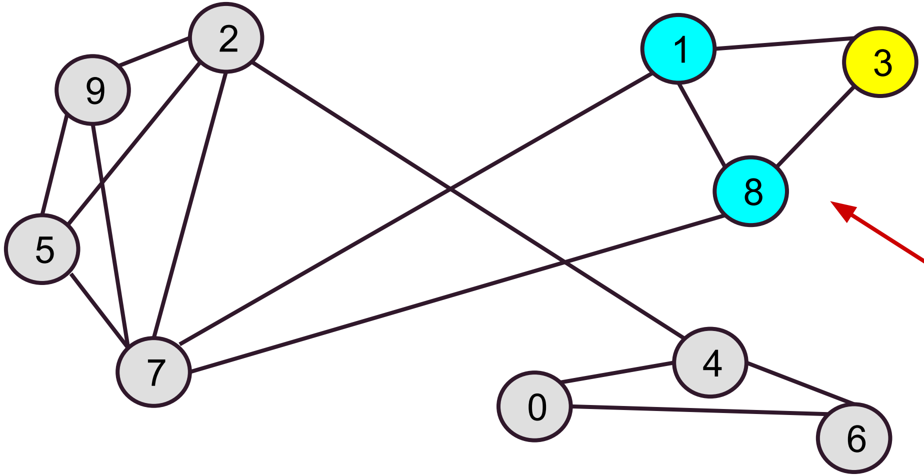
procedure $\text{CLIQUE}(G = (V, E), U, size)$
    if $U = \emptyset$ **then**
        if $size > max$ **then**
            $max \leftarrow size$
        return
    while $|U| > 0$ **do**
        Select any vertex u from U
        $U \leftarrow U \setminus \{u\}$
        $\text{CLIQUE}(G, U \cap N'(u), size + 1)$

FindMaximumClique(G)
    max_clq = lower_bound;
    For each vertex v_i
        Remove v_i from G
        FindMaximalCliqueOfV(Neighbors(v_i), 1)

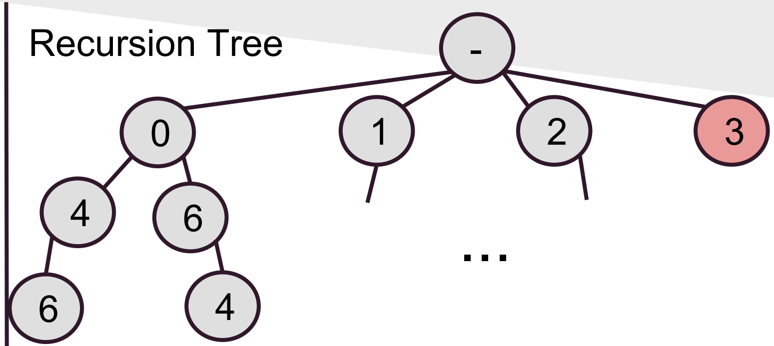
FindMaximalCliqueOfV(U, size)
    if U is empty then
        if size > max_clq
            max_clq = size
        return
    For each vertex v_j in U
        Remove v_j from U
        U_{new} = Neighbors(v_j) $\cap$ U
        FindMaximalCliqueOfV(U_{new}, size+1)

Pardalos Algorithm

$|V| = 10, |E| = 15$



Recursion Tree



Current Step:

Next Step:

Has only 2 neighbors. Even if all neighbors were included, only a size 3 clique can be formed (<= 4)

Max clique size = 4
Max clique set = {2,5,7,9}

Current clique size = 0
Current clique set = {}
Current Node = Node 3
Current Neighbors = {1,8}
Candidate set = {3,...,9}

Modified Pardalos Algorithm

procedure MAXCLIQUE($G = (V, E), lb$)
  $max \leftarrow lb$
  for $i : 1$ to n **do**
    $U \leftarrow \emptyset$
    for each $v_j \in N(v_i)$ **do**
      $U \leftarrow U \cup \{v_j\}$
    CLIQUE($G, U, 1$)

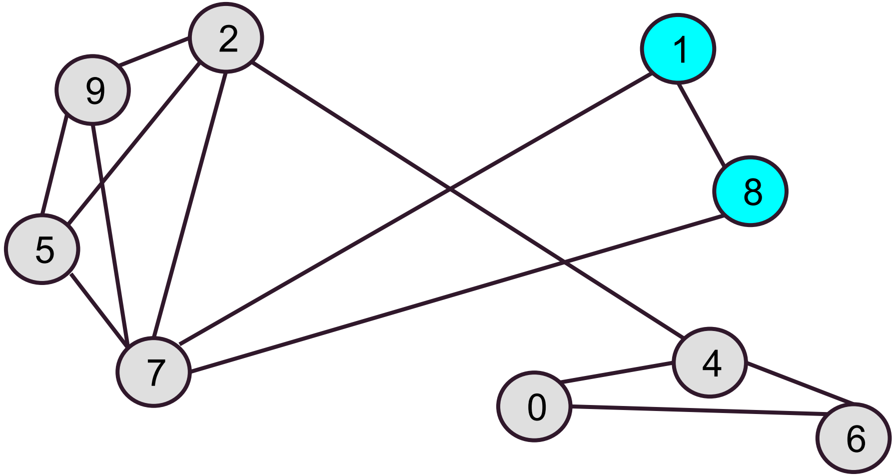
procedure CLIQUE($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    CLIQUE($G, U \cap N'(u), size + 1$)

procedure MAXCLIQUE($G = (V, E), lb$)
  $max \leftarrow lb$
  for $i : 1$ to n **do**
    if $d(v_i) \geq max$ **then**   ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        $U \leftarrow U \cup \{v_j\}$
      CLIQUE($G, U, 1$)

procedure CLIQUE($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    CLIQUE($G, U \cap N'(u), size + 1$)

Pardalos Algorithm

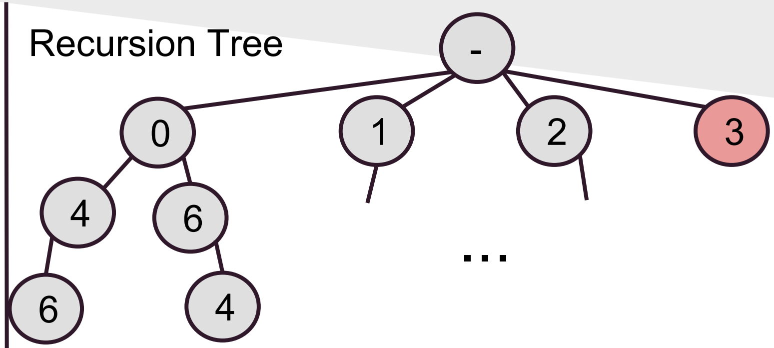
|V| = 10, |E| = 15



Max clique size = 4
Max clique set = {2,5,7,9}

Current clique size = 1
Current clique set = {3}
Current Node = ---
Current Neighbors = ---
Candidate set = {1,8}

Recursion Tree



All cliques containing Node 3 already examined. Can discard this node and edges from any future computation.

Modified Pardalos Algorithm

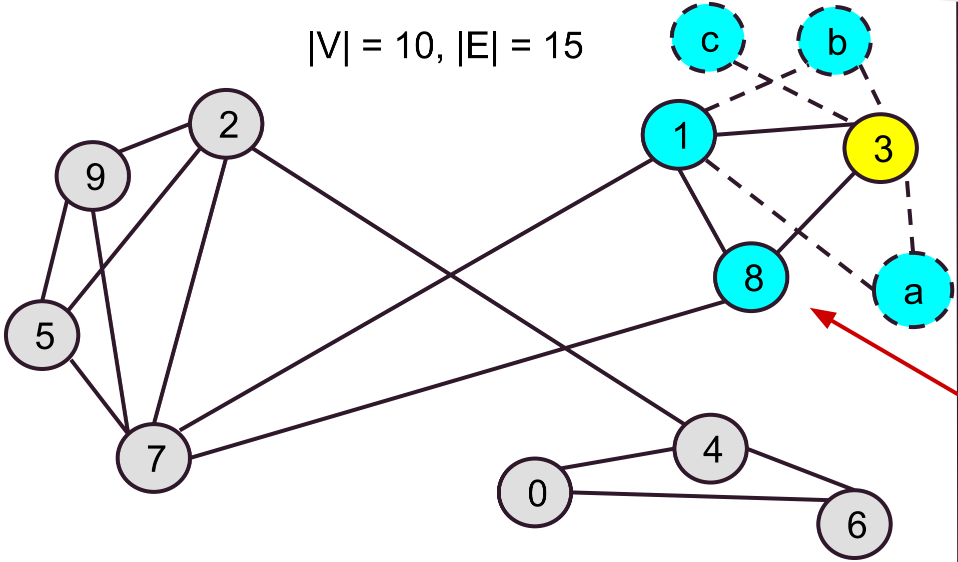
procedure MAXCLIQUE($G = (V, E), lb$)
  $max \leftarrow lb$
  for i : 1 to n **do**
    if $d(v_i) \geq max$ **then** ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        $U \leftarrow U \cup \{v_j\}$
      CLIQUE($G, U, 1$)

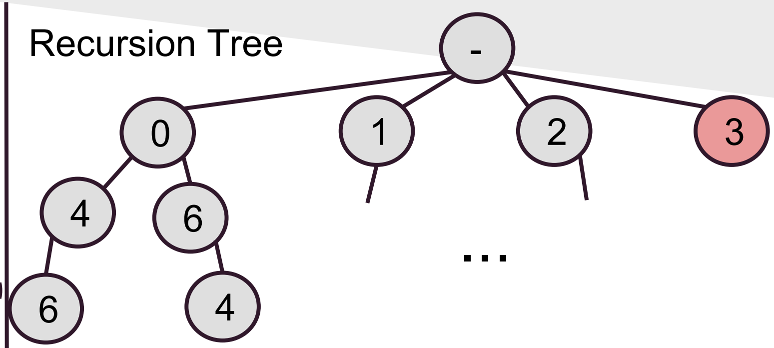
procedure CLIQUE($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    CLIQUE($G, U \cap N'(u), size + 1$)

procedure MAXCLIQUE($G = (V, E), lb$)
  $max \leftarrow lb$
  for i : 1 to n **do**
    if $d(v_i) \geq max$ **then** ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        if $j > i$ **then** ▷ Pruning 2
          $U \leftarrow U \cup \{v_j\}$
      CLIQUE($G, U, 1$)

procedure CLIQUE($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    CLIQUE($G, U \cap N'(u), size + 1$)

Pardalos Algorithm





Max clique size = 4
Max clique set = {2,5,7,9}

Current clique size = 0
Current clique set = {}
Current Node = Node 3
Current Neighbors = {1,8}
Candidate set = {3,...,9}

Current Step:

Next Step:

Each neighbor also should also have >= 4 neighbors to be a part of a clique of larger than 4 nodes. In this case only Node 1 can possibly form a clique of size >= 4.

Modified Pardalos Algorithm

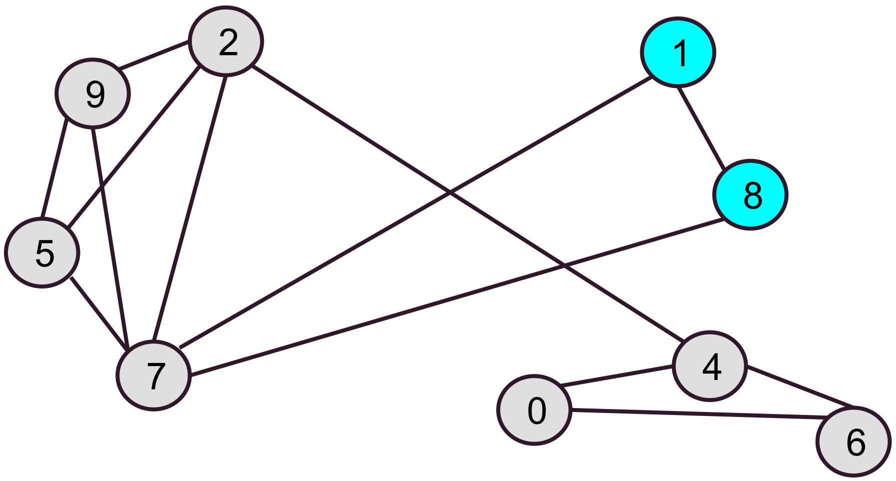
procedure MaxClique($G = (V, E), lb$)
  $max \leftarrow lb$
  for $i : 1$ to n **do**
    if $d(v_i) \geq max$ **then** ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        if $j > i$ **then** ▷ Pruning 2
          $U \leftarrow U \cup \{v_j\}$
      Clique($G, U, 1$)

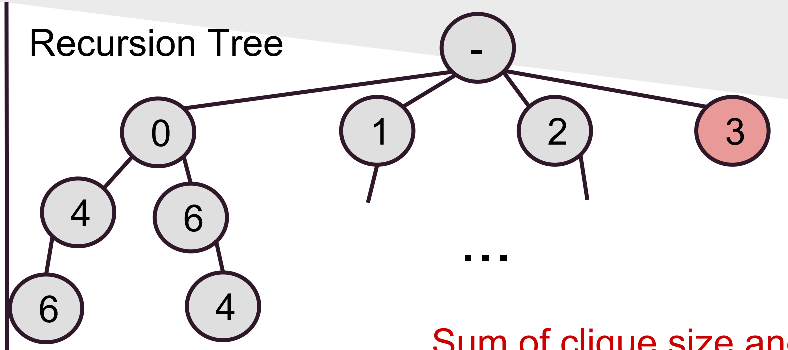
procedure Clique($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    Clique($G, U \cap N'(u), size + 1$)

procedure MaxClique($G = (V, E), lb$)
  $max \leftarrow lb$
  for $i : 1$ to n **do**
    if $d(v_i) \geq max$ **then** ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        if $j > i$ **then** ▷ Pruning 2
          if $d(v_j) \geq max$ **then** ▷ Pruning 3
            $U \leftarrow U \cup \{v_j\}$
      Clique($G, U, 1$)

procedure Clique($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    Clique($G, U \cap N'(u), size + 1$)

Pardalos Algorithm





Sum of clique size and size of candidate set must be larger than max clique

Max clique size = 4
Max clique set = {2,5,7,9}

Current clique size = 1
Current clique set = {3}
Current Node = ---
Current Neighbors = ---
Candidate set = {1,8}

Modified Pardalos Algorithm

```
procedure MaxClique(G = (V, E), lb)
    max ← lb
    for i : 1 to n do
        if d(v_i) ≥ max then                ▷ Pruning 1
            U ← ∅
            for each v_j ∈ N(v_i) do
                if j > i then               ▷ Pruning 2
                    if d(v_j) ≥ max then    ▷ Pruning 3
                        U ← U ∪ {v_j}
            Clique(G, U, 1)
procedure Clique(G = (V, E), U, size)
    if U = ∅ then
        if size > max then
            max ← size
        return
    while |U| > 0 do
        Select any vertex u from U
        U ← U \ {u}
        Clique(G, U ∩ N'(u), size + 1)
```

```
procedure MaxClique(G = (V, E), lb)
    max ← lb
    for i : 1 to n do
        if d(v_i) ≥ max then                ▷ Pruning 1
            U ← ∅
            for each v_j ∈ N(v_i) do
                if j > i then               ▷ Pruning 2
                    if d(v_j) ≥ max then    ▷ Pruning 3
                        U ← U ∪ {v_j}
            Clique(G, U, 1)
procedure Clique(G = (V, E), U, size)
    if U = ∅ then
        if size > max then
            max ← size
        return
    while |U| > 0 do
        if size + |U| ≤ max then            ▷ Pruning 4
            return
        Select any vertex u from U
        U ← U \ {u}
        Clique(G, U ∩ N'(u), size + 1)
```

Modified Pardalos Algorithm

```
procedure MaxClique(G = (V, E), lb)
    max ← lb
    for i : 1 to n do
        if d(v_i) ≥ max then                ▷ Pruning 1
            U ← ∅
            for each v_j ∈ N(v_i) do
                if j > i then               ▷ Pruning 2
                    if d(v_j) ≥ max then    ▷ Pruning 3
                        U ← U ∪ {v_j}
            Clique(G, U, 1)
procedure Clique(G = (V, E), U, size)
    if U = ∅ then
        if size > max then
            max ← size
        return
    while |U| > 0 do
        if size + |U| ≤ max then            ▷ Pruning 4
            return
        Select any vertex u from U
        U ← U \ {u}
        Clique(G, U ∩ N'(u), size + 1)
```

```
procedure MaxClique(G = (V, E), lb)
    max ← lb
    for i : 1 to n do
        if d(v_i) ≥ max then                ▷ Pruning 1
            U ← ∅
            for each v_j ∈ N(v_i) do
                if j > i then               ▷ Pruning 2
                    if d(v_j) ≥ max then    ▷ Pruning 3
                        U ← U ∪ {v_j}
            Clique(G, U, 1)
procedure Clique(G = (V, E), U, size)
    if U = ∅ then
        if size > max then
            max ← size
        return
    while |U| > 0 do
        if size + |U| ≤ max then            ▷ Pruning 4
            return
        Select any vertex u from U
        U ← U \ {u}
        N'(u) := {w | w ∈ N(u) ∧ d(w) ≥ max}  ▷ Pruning 5
        Clique(G, U ∩ N'(u), size + 1)
```

Modified Pardalos Algorithm

procedure $\textsc{MaxClique}(G = (V, E), lb)$
    $max \leftarrow lb$
    for i : 1 to n **do**
        if $d(v_i) \geq max$ **then** ▷ Pruning 1
            $U \leftarrow \emptyset$
            for each $v_j \in N(v_i)$ **do**
                if $j > i$ **then** ▷ Pruning 2
                    if $d(v_j) \geq max$ **then** ▷ Pruning 3
                        $U \leftarrow U \cup \{v_j\}$
            $\textsc{Clique}(G, U, 1)$
procedure $\textsc{Clique}(G = (V, E), U, size)$
    if $U = \emptyset$ **then**
        if $size > max$ **then**
            $max \leftarrow size$
        return
    while $|U| > 0$ **do**
        if $size + |U| \leq max$ **then** ▷ Pruning 4
            return
        Select any vertex u from U
        $U \leftarrow U \setminus \{u\}$
        $\textsc{Clique}(G, U \cap N'(u), size + 1)$

procedure $\textsc{MaxClique}(G = (V, E), lb)$
    $\boxed{max \leftarrow lb}$
    for i : 1 to n **do**
        if $d(v_i) \geq max$ **then** ▷ Pruning 1
            $U \leftarrow \emptyset$
            for each $v_j \in N(v_i)$ **do**
                if $j > i$ **then** ▷ Pruning 2
                    if $d(v_j) \geq max$ **then** ▷ Pruning 3
                        $U \leftarrow U \cup \{v_j\}$
            $\textsc{Clique}(G, U, 1)$
procedure $\textsc{Clique}(G = (V, E), U, size)$
    if $U = \emptyset$ **then**
        if $size > max$ **then**
            $max \leftarrow size$
        return
    while $|U| > 0$ **do**
        if $size + |U| \leq max$ **then** ▷ Pruning 4
            return
        Select any vertex u from U
        $U \leftarrow U \setminus \{u\}$
        $N'(u) := \{w | w \in N(u) \wedge d(w) \geq max\}$ ▷
Pruning 5
        $\textsc{Clique}(G, U \cap N'(u), size + 1)$

New Algorithms

Algorithm 1 Algorithm for finding the maximum clique of a given graph. *Input*: Graph $G = (V, E)$, lower bound on clique lb (default, 0). *Output*: Size of maximum clique.

procedure MAXCLIQUE($G = (V, E), lb$)
  $max \leftarrow lb$
  for i : 1 **to** n **do**
    if $d(v_i) \geq max$ **then** ▷ Pruning 1
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        if $j > i$ **then** ▷ Pruning 2
          if $d(v_j) \geq max$ **then** ▷ Pruning 3
            $U \leftarrow U \cup \{v_j\}$
      CLIQUE($G, U, 1$)

– *Subroutine*

procedure CLIQUE($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  while $|U| > 0$ **do**
    if $size + |U| \leq max$ **then** ▷ Pruning 4
      return
    Select any vertex u from U
    $U \leftarrow U \setminus \{u\}$
    $N'(u) := \{w | w \in N(u) \wedge d(w) \geq max\}$ ▷ Pruning 5
    CLIQUE($G, U \cap N'(u), size + 1$)

Algorithm 2 Heuristic for finding the maximum clique in a graph. *Input*: Graph $G = (V, E)$. *Output*: Approximate size of maximum clique.

procedure MAXCLIQUEHEU($G = (V, E)$)
  for i : 1 **to** n **do**
    if $d(v_i) \geq max$ **then**
      $U \leftarrow \emptyset$
      for each $v_j \in N(v_i)$ **do**
        if $d(v_j) \geq max$ **then**
          $U \leftarrow U \cup \{v_j\}$
      CLIQUEHEU($G, U, 1$)

– *Subroutine*

procedure CLIQUEHEU($G = (V, E), U, size$)
  if $U = \emptyset$ **then**
    if $size > max$ **then**
      $max \leftarrow size$
    return
  Select a vertex $u \in U$ of maximum degree in G
  $U \leftarrow U \setminus \{u\}$
  $N'(u) := \{w | w \in N(u) \wedge d(w) \geq max\}$
  CLIQUEHEU($G, U \cap N'(u), size + 1$)

Experiments

- Testbed
  - Real world graphs
  - Synthetic graphs
  - DIMACS graphs

Testbed

- Real world graphs
  - Obtained from Florida Matrix Collection* - a large and actively growing set of sparse matrices that arise in real applications

Graph	Description
cond-mat-2003 [26]	A collaboration network of scientists posting preprints on the condensed matter archive at www.arxiv.org in the period
email-Enron [23]	A communication network representing email exchanges.
dictionary28 [4]	Pajek network of words.
Fault_639 [14]	A structural problem discretizing a faulted gas reservoir with tetrahedral Finite Elements and triangular Interface Elements.
audikw_1 [11]	An automotive crankshaft model of TETRA elements.
bone010 [39]	A detailed micro-finite element model of bones representing the porous bone micro-architecture.
af_shell [11]	A sheet metal forming simulation network.
as-Skitter [23]	An Internet topology graph from trace routes run daily in 2005.
roadNet-CA [23]	A road network of California. Nodes represent intersections and endpoints and edges represent the roads connecting them.
kkt_power [11]	An Optimal Power Flow (nonlinear optimization) network.

* http://www.cise.ufl.edu/research/sparse/matrices/

Testbed

- Synthetic graphs
  - Generated using the RMAT algorithm*

A. Random graphs (5 graphs) – Erdős-Rényi random graphs generated using R-MAT with the parameters (0.25, 0.25, 0.25, 0.25). Denoted with prefix *rmat_er*.
B. Skewed Degree, Type 1 graphs (5 graphs) – graphs generated using R-MAT with the parameters (0.45, 0.15, 0.15, 0.25). Denoted with prefix *rmat_sd1*.
C. Skewed Degree, Type 2 graphs (5 graphs) – graphs generated using R-MAT with the parameters (0.55, 0.15, 0.15, 0.15). Denoted with prefix *rmat_sd2*.

- DIMACS graphs
  - From the Second DIMACS Implementation Challenge
  - Established benchmark for the maximum clique problem

* D. Chakrabarti and C. Faloutsos, *Graph mining: Laws, generators, and algorithms*, ACM Comput. Surv. 38 (2006).

Testbed

Table 2. Structural properties (the number of vertices, $|V|$; edges, $|E|$; and the maximum degree, Δ) of the graphs, G in the testbed: DIMACS Challenge graphs (upper left); UF Collection (lower and middle left); RMAT graphs (right).

G	$\|V\|$	$\|E\|$	Δ	G	$\|V\|$	$\|E\|$	Δ
cond-mat-2003	31,163	120,029	202	*rmat_sd1_1*	131,072	1,046,384	407
email-Enron	36,692	183,831	1,383	*rmat_sd1_2*	262,144	2,093,552	558
dictionary28	52,652	89,038	38	*rmat_sd1_3*	524,288	4,190,376	618
Fault_639	638,802	13,987,881	317	*rmat_sd1_4*	1,048,576	8,382,821	802
audikw_1	943,695	38,354,076	344	*rmat_sd1_5*	2,097,152	16,767,728	1,069
bone010	986,703	35,339,811	80	*rmat_sd2_1*	131,072	1,032,634	2,980
af_shell10	1,508,065	25,582,130	34	*rmat_sd2_2*	262,144	2,067,860	4,493
as-Skitter	1,696,415	11,095,298	35,455	*rmat_sd2_3*	524,288	4,153,043	6,342
roadNet-CA	1,971,281	2,766,607	12	*rmat_sd2_4*	1,048,576	8,318,004	9,453
kkt_power	2,063,494	6,482,320	95	*rmat_sd2_5*	2,097,152	16,645,183	14,066
rmat_er_1	131,072	1,048,515	82	*hamming6-4*	64	704	22
rmat_er_2	262,144	2,097,104	98	*johnson8-4-4*	70	1,855	53
rmat_er_3	524,288	4,194,254	94	*keller4*	171	9,435	124
rmat_er_4	1,048,576	8,388,540	97	*c-fat200-5*	200	8,473	86
rmat_er_5	2,097,152	16,777,139	102	*brock200_2*	200	9,876	114

Algorithms - Comparison

- Carraghan and Pardalos 1990 - Self-implemented
- Ostergard 2002 - *cliquer* software package
  - http://users.tkk.fi/pat/cliquer.html
- MCQD+CS 2007 - *MaxCliqueDyn* software package
  - http://www.sicmm. org/˜konc/maxclique/

Experiments

- Setup
  - Linux workstation (64-bit Red Hat Enterprise Server release)
  - 6.22 GHz Intel Xeon E7540 processor
  - Implemented in C++
  - gcc version 4.4.6 with -O3 optimization.
  - Single threaded

Results - real-world graphs

Graph	ω	τ_{CP}	$\tau_{cliquer}$	τ_{MCQD} $+CS$	τ_{A1}	$P1$	$P2$	$P3$	$P5$	ω_{A2}	τ_{A2}
cond-mat-2003	25	4.875	11.17	2.41	**0.011**	29K	48K	6,527	17K	25	<0.01
email-Enron	20	7.005	15.08	3.70	**0.998**	32K	155K	4,060	8M	18	0.261
dictionary28	26	7.700	32.74	7.69	**<0.01**	52K	4,353	2,114	107	26	<0.01
Fault_639	18	14571.20	4437.14	-	**20.03**	36	13M	126	1,116	18	5.80
audikw_1	36	*	9282.49	-	**190.17**	4,101	38M	59K	721K	36	58.38
bone010	24	*	10002.67	-	**393.11**	37K	34M	361K	44M	24	24.39
af_shell10	15	*	21669.96	-	**50.99**	19	25M	75	2,105	15	10.67
as-Skitter	67	24385.73	*	-	**3838.36**	1M	6M	981K	737M	66	27.08
roadNet-CA	4	*	*	-	**0.44**	1M	1M	370K	4,302	4	0.08
kkt_power	11	*	*	-	**2.26**	1M	4M	401K	2M	11	1.83

LEGEND:

- *CP* - Pardalos 1990
- *cliquer* - Ostergard 2001
- *MCQD+CS* - Konc & Janezic 2007
- A1 - Our new algorithm
- * - More than 25,000 sec

- *P1, P2, P3, P5* - Nodes/Computation Pruned
- ω_{A2} - Max clique by Heuristic
- τ_{A2} - Time taken by Heuristic
- ω - Actual max clique
- \- - Implementation couldn't handle

Results - real-world graphs

Graph	ω	τ_{CP}	$\tau_{cliquer}$	$\tau_{MCQD+CS}$	τ_{A1}	$P1$	$P2$	$P3$	$P5$	ω_{A2}	τ_{A2}
rmat_er_1	3	256.37	215.18	49.79	**0.38**	780	1M	915	8,722	3	0.12
rmat_er_2	3	1016.70	865.18	-	**0.78**	2,019	2M	2,351	23K	3	0.24
rmat_er_3	3	4117.35	3456.39	-	**1.87**	4,349	4M	4,960	50K	3	0.49
rmat_er_4	3	16419.80	13894.52	-	**4.16**	9,032	8M	10K	106K	3	1.44
rmat_er_5	3	*	*	-	**9.87**	18K	16M	20K	212K	3	2.57
rmat_sd1_1	6	225.93	214.99	50.08	**1.39**	39K	1M	23K	542K	6	0.45
rmat_sd1_2	6	912.44	858.80	-	**3.79**	90K	2M	56K	1M	6	0.98
rmat_sd1_3	6	3676.14	3446.02	-	**8.17**	176K	4M	106K	2M	6	1.78
rmat_sd1_4	6	14650.40	13923.93	-	**25.61**	369K	8M	214K	5M	6	4.05
rmat_sd1_5	6	*	*	-	**46.89**	777K	16M	455K	12M	6	9.39
rmat_sd2_1	26	427.41	213.23	**48.17**	242.20	110K	853K	88K	614M	26	32.83
rmat_sd2_2	35	4663.62	**851.84**	-	3936.55	232K	1M	195K	1B	35	95.89
rmat_sd2_3	39	13626.23	**3411.14**	-	10647.84	470K	3M	405K	1B	37	245.51
rmat_sd2_4	43	*	**13709.52**	-	*	*	*	*	*	42	700.05
rmat_sd2_5	N	*	*	-	*	*	*	*	*	51	1983.21

LEGEND:

CP - Pardalos 1990
cliquer - Ostergard 2001
MCQD+CS - Konc & Janezic 2007
A1 - Our new algorithm
* - More than 25,000 sec

P1, P2, P3, P5 - Nodes/Computation Pruned
ω_{A2} - Max clique by Heuristic
τ_{A2} - Time taken by Heuristic
ω - Actual max clique
\- - Implementation couldn't handle

Results - real-world graphs

Graph	ω	τ_{CP}	$\tau_{cliquer}$	$\tau_{MCQD+CS}$	τ_{A1}	$P1$	$P2$	$P3$	$P5$	ω_{A2}	τ_{A2}
hamming6-4	4	**<0.01**	**<0.01**	**<0.01**	<0.01	0	704	0	0	4	<0.01
johnson8-4-4	14	0.19	**<0.01**	**<0.01**	0.23	0	1,855	0	0	14	<0.01
keller4	11	22.19	0.15	**0.02**	23.35	0	9,435	0	0	11	<0.01
c-fat200-5	58	0.60	0.33	**0.01**	0.93	0	8,473	0	0	58	0.04
brock200_2	12	0.98	0.02	**<0.01**	1.10	0	9,876	0	0	10	<0.01

LEGEND:

CP	-	Pardalos 1990
cliquer	-	Ostergard 2001
MCQD+CS	-	Konc & Janezic 2007
A1	-	Our new algorithm
*	-	More than 25,000 sec
P1, P2, P3, P5	-	Nodes/Computation Pruned
ω_{A2}	-	Max clique by Heuristic
τ_{A2}	-	Time taken by Heuristic
ω	-	Actual max clique
-	-	Implementation couldn't handle

Results - summary

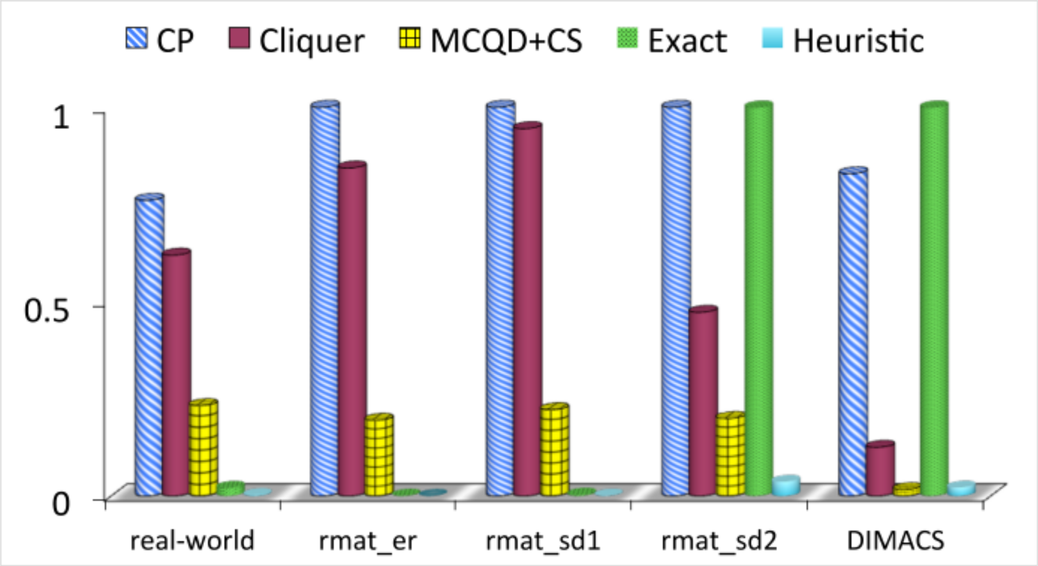


Fig. 1. Runtime (normalized, mean) comparison between various algorithms. For each category of graph, first, all runtimes for each graph were normalized by the runtime of the slowest algorithm for that graph, and then the mean was calculated for each algorithm. Graphs were considered only if the runtimes for at least three algorithms was less than the 25,000 seconds limit set.

Results - summary

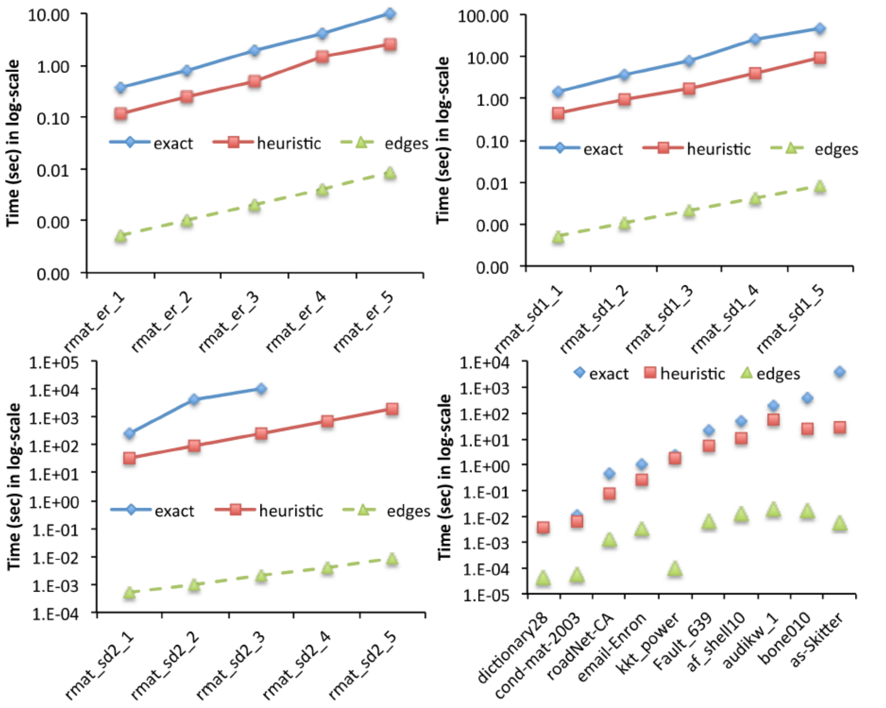


Fig. 2. Run time plots of the new exact and heuristic algorithms. The third curve, labeled *edges*, shows the quantity, number of edges in the graph divided by the clock frequency of the computing platform used in the experiment.

Summary

- New algorithm
  - Very effective and orders of magnitude times faster on large sparse graphs compared to existing algorithms
  - For certain synthetic graphs and DIMACS graphs, slower that existing algorithms
- Heuristic
  - Delivers optimal solution for 83% of graphs in testbed
  - When sub-optimal, accuracy ranges between 0.83 - 0.99

Future Work

- Thorough analysis on effect of pruning steps
- Effect of vertex ordering
- Use heuristic-based approximate lower bound to improve pruning
- Compare with more recent algorithms (implementation not publicly available)
- Compare heuristic with others