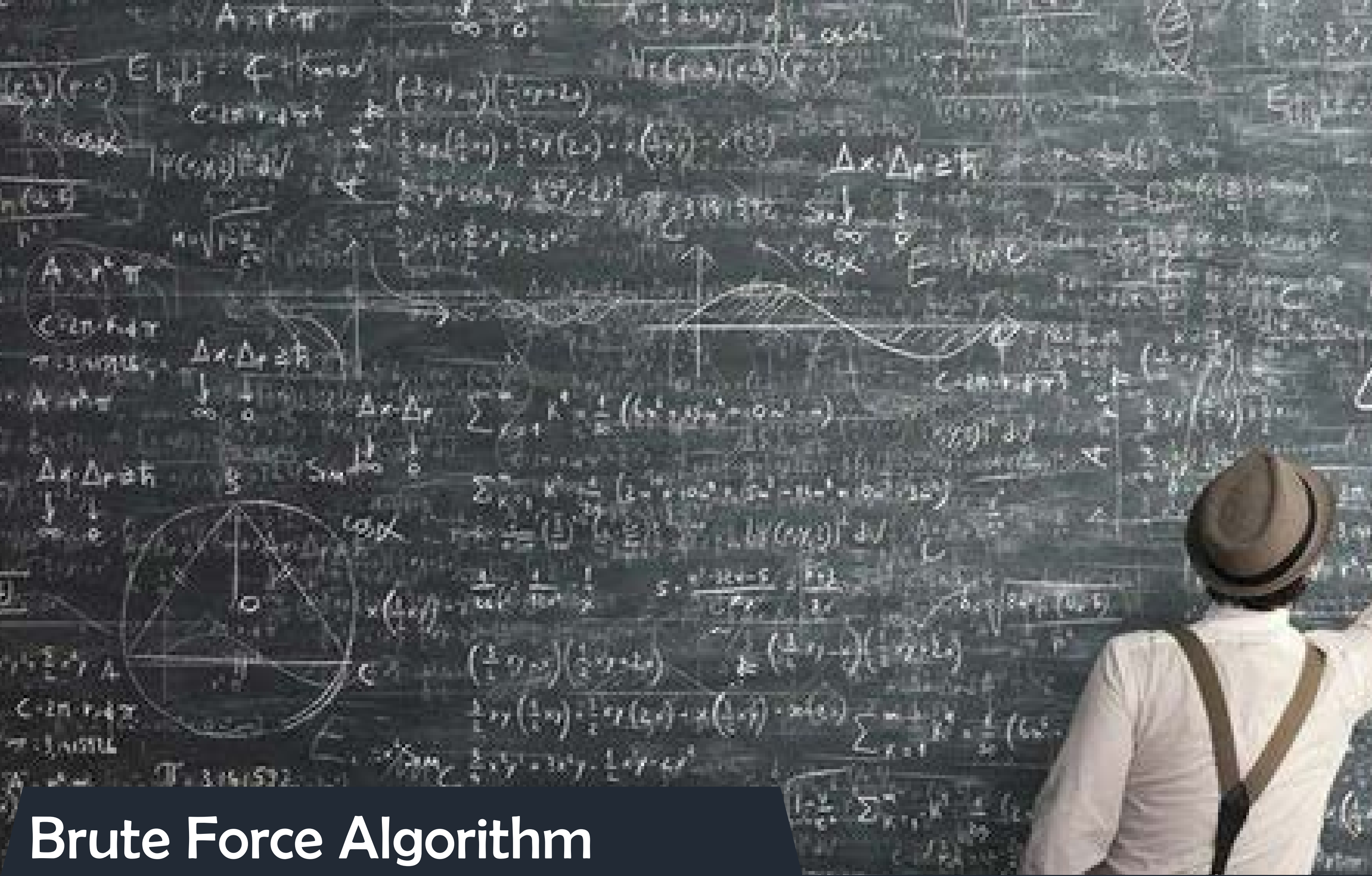


Brute Force Algorithm

Ja-Hee Kim

Agenda

Introduction

Techniques

Examples

Introduction

Brute Force

- Based on the problem's statement and definitions of the concepts involved.
- A **straightforward** approach, usually based directly on the problem's statement and definitions of the concepts involved

Strength and Weakness

- Strengths
  - wide applicability
  - **simplicity**
  - yields reasonable algorithms for some important problems

  (e.g., matrix multiplication, sorting, searching, string matching)
- Weaknesses
  - Usually yields **inefficient** algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive as some other design techniques

Techniques

Basic techniques

- Optimization problems
- Generate and test
- Backtracking
- Fixing parameters
- Meet in the middle

Optimization problems

- for a given solution set *S* and a value function *f*, find an $x \in S$, which maximize *f(x)*.
- Example: Find the max value
  - Algorithm: Scan the array to find its maximum element and return it with the maximum element.

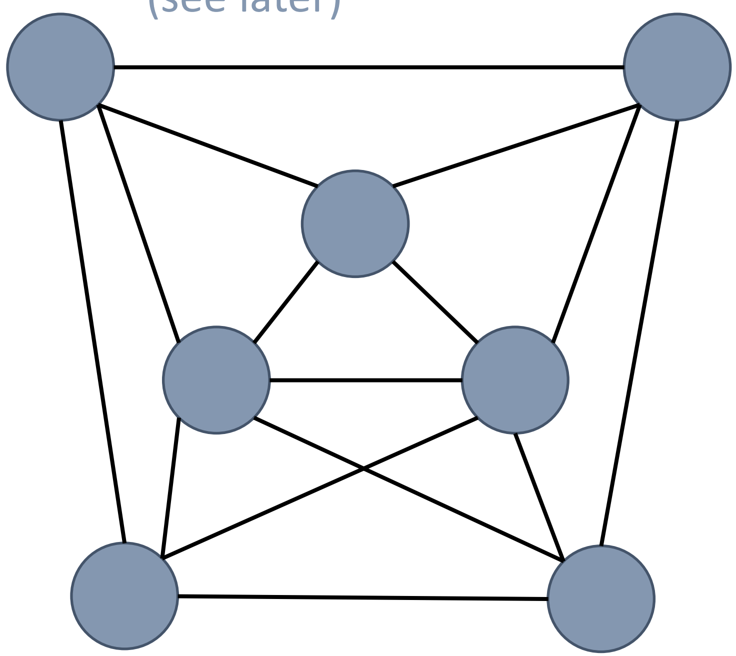
$A[0], \ldots, A[min], \ldots, A[n-1]$

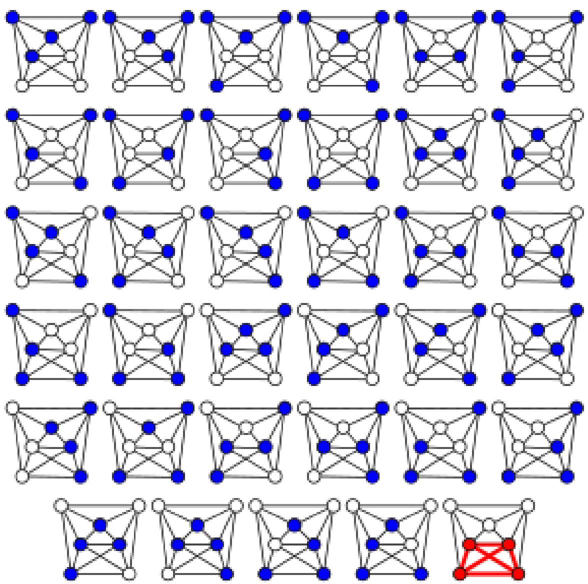
  - Time efficiency: **$\Theta(n)$**
  - Space efficiency: **$\Theta(1)$, so in place**
  - Stability: **yes**

Generate and test

- generating all solution and test all of them.
- Example: Clique Problem
    - finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph.

(see later)



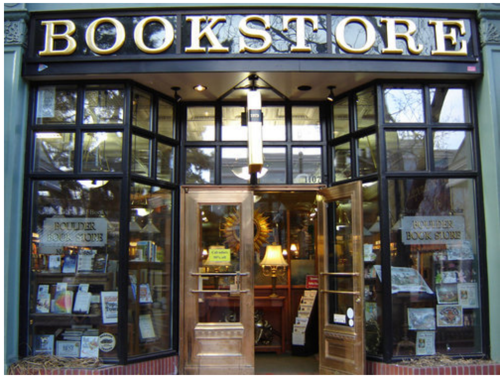


Backtracking

- Constrained problem
- Step
  - It incrementally builds candidates for solutions
  - Abandons a candidate as soon as the candidate cannot satisfy the constraint.
- Example: labyrinth
  - Input size n : the number of intersection
  - Time complexity: $\Theta(n^3)$
  - Space complexity: $\Theta(n)$

Fixing parameters

- Instead in testing the solution set itself, it try to fix some parameters.
- Example: Buying books
  - Problem description:
    - to buy *n* books from *m* book shops.
    - Each book is sold by at least one bookstore
    - The price of each book can vary between the different stores.
    - If you order anything from a certain bookstore, you must pay for postage, which
      - may vary between bookstores
      - is the same no matter how many books you decide to order.
    - **Compute the smallest amount of money you need to pay for all the books.**
  - Time complexity: $\Theta(n^m)$



Changed problem: Deciding the bookstore(parameter) where you buy books.
Time complexity: $\mathbf{\Theta((mn)2^m)}$

Meet in the middle

- To fix half of the parameter space and build some fast structure s.t. when testing the other half of the parameter space.
- Example: subset sum.
  - Given a set of integers S, is there some subset $A \subseteq S$ with a sum equal to T?
  - *n*: the number of elements in set S
  - Time complexity: $\Theta(2^n)$

Time complexity: $\Theta(n2^{n/2})$

```
procedure SubsetSum(set S, target T)
    N ← |S|
    left ← N/2
    right ← N − left
    Lset ← the left first elements of S
    Rset ← S \ Lset
    Lsums ← new set
    for each L ⊆ Lset do
        Lsums.insert(∑_{l∈L} l)
    for each R ⊆ Rset do
        sum ← ∑_{r∈R} r
        if Lsums.contains(T − sum) then
            output true
            return
    output false
```

constant

$\Theta(2^{n/2})$

$\Theta(n/2)$

$\Theta(\frac{n}{2} 2^{\frac{n}{2}})$

$\Theta(2^{n/2})$

$\Theta(n/2)$

$\Theta(\frac{n}{2} 2^{\frac{n}{2}})$

Constant or $\Theta(n/2)$

Examples: maximal cliques (Bron Kerbosch with pivot)

Examples

- Computing *n! : O(n)*
- Multiplying two matrices of size *n: O(n)*
- Sequential Searching for a key of a given value in a list of length *n: O(n)*
- Simple sort: selection sort, bubble sort: $O(n^2)$
- String matching: *O(nm)*
  - https://www.youtube.com/watch?v=pTBJhXrEmxo
  - Pattern: a string of m characters to search for. (ex: 001011)-length: m
  - text: a (longer) string of n characters to search in (ex:10010101101001100101111010)-length: n
  - problem: find a substring in the text that matches the pattern

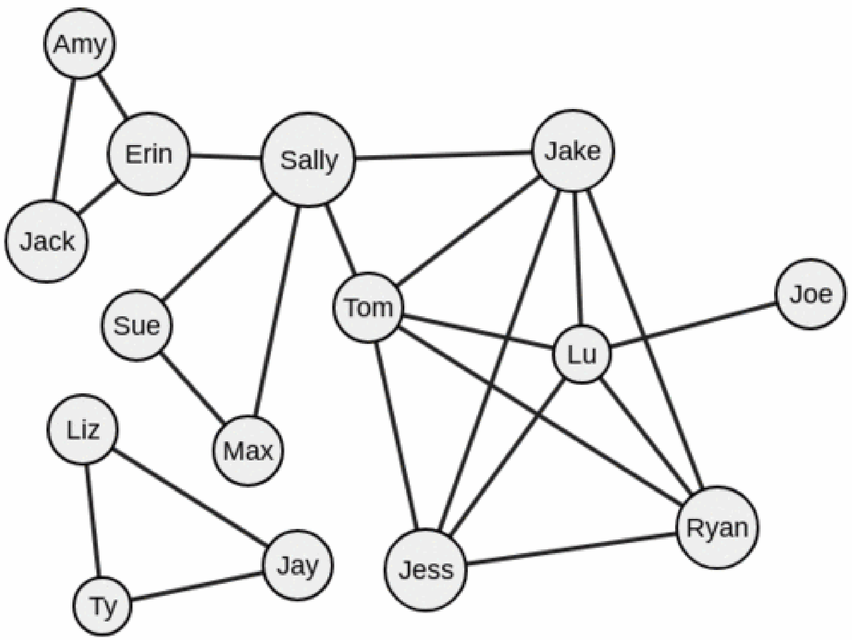
Example: Maximal cliques

- Terminologies
  - Cliques: complete subgraphs of a graph.
  - Maximal clique: a clique that cannot be extended by including any more adjacent vertices

- Input: a graph

- Output: the list of maximal cliques

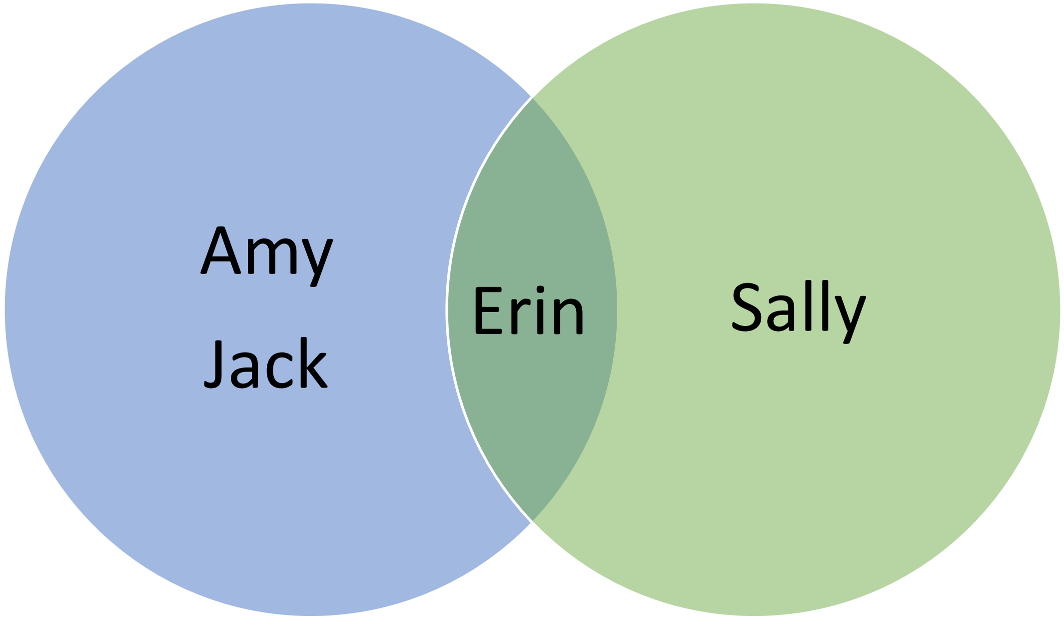
Problem description



Sally is having a party. She invites Max, Sue, Tom and Jake. Then Tom invites Ryan, who brings Jess, and Jess brings Lu who knows Tom's friends. Jake is popular, so Jake also knows Jess and Ryan. Then there is Joe. Joe knows Lu, but doesn't know Sally, or anyone else really, although still 'technically' a guest at the party. Three randoms Liz, Ty, and Jay show up, because the party is a whomper. Then finally, Erin, who almost always is fashionably late, shows up with Amy and Jack.

Our task is to find all the maximal cliques, the groupings of people that all know each other.

Set operations



- Union: $A \cup B$ = {Jack, Amy, Sally, Erin}
- Intersection: $A \cap B$ = {Erin}
- Relative complement: $A \setminus B$ = {Amy, Jack}

Bron-Kerbosh algorithm

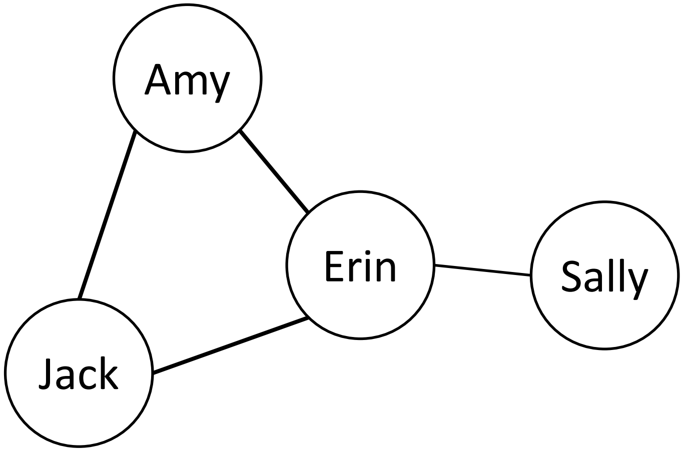
- Bron-Kerbosch operates on three sets: R, P, and X.
  - R := is the set of nodes of a maximal clique.
  - P := is the set of possible nodes in a maximal clique.
  - X := is the set of nodes that are excluded.
  - N(v): the neighbors of v(vertex)

- Pseudo code

```
BronKerbosch1(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  for each vertex v in P:
    BronKerbosch1( R ⋃ {v},  P ⋂ N(v),  X ⋂ N(v) )
    P := P \ {v}
    X := X ⋃ {v}
```

Example

BronKerbosch(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  for each vertex v in P:
    BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
    P := P \ {v}
    X := X ⋃ {v}

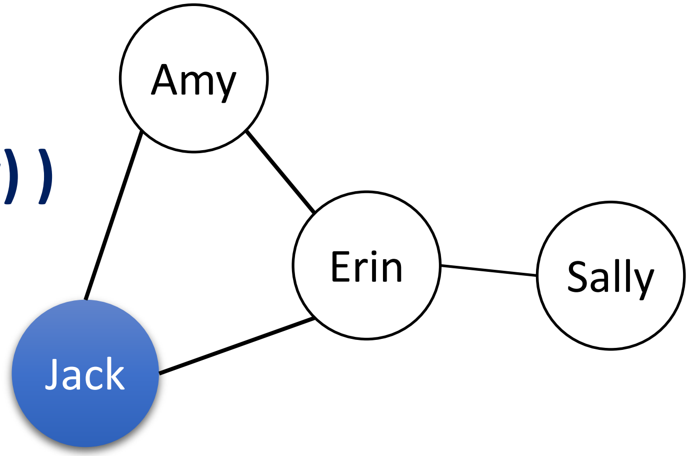


- R={}
- P={Jack, Amy, Erin, Sally}
- X={}
- v=Jack
- N(v)={Amy, Erin}

- R ⋃ {v}={Jack}
- P ⋂ N(v) = {Amy, Erin}
- X ⋂ N(v) ={}

Example

```
BronKerbosch(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  for each vertex v in P:
    BronKerbosch( R ⋃ {v}, P ⋂ N(v), X ⋂ N(v) )
    P := P \ {v}
    X := X ⋃ {v}
```

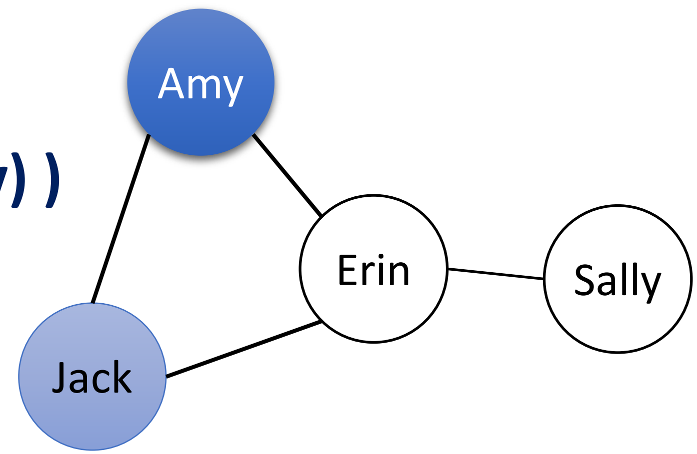


- R={Jack}
- P={Amy, Erin}
- X={}
- v=Amy
- N(v) = {Jack, Erin}

- R ⋃ {v}={Jack, Amy}
- P ⋂ N(v) = {Erin}
- X ⋂ N(v) ={}

Example

BronKerbosch(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  for each vertex v in P:
    BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
    P := P \ {v}
    X := X ⋃ {v}



- R={Jack, Amy}
- P={Erin}
- X={}
- v=Erin
- N(v) = {Amy, Jack, Sally}

- R ⋃ {v}={Jack, Amy, Erin}
- P ⋂ N(v) = {}
- X ⋂ N(v) ={}

Example

BronKerbosch(R, P, X):

if P and X are both empty:

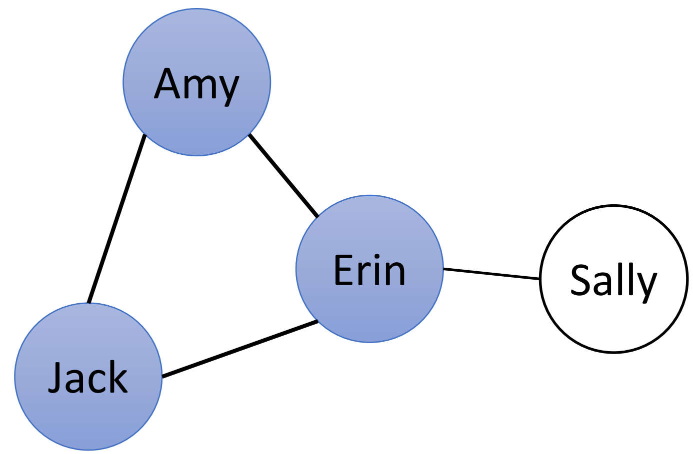
report R as a maximal clique

for each vertex v in P:

BronKerbosch($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)

$P := P \setminus \{v\}$

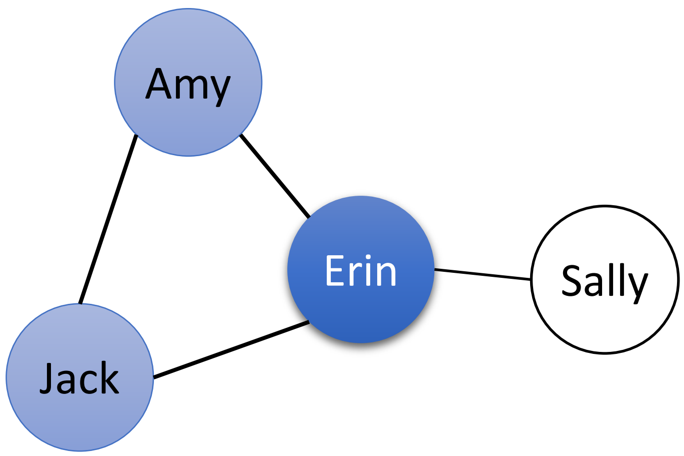
$X := X \cup \{v\}$



- R={Jack, Amy, Erin}
- P={}
- X={}

Example

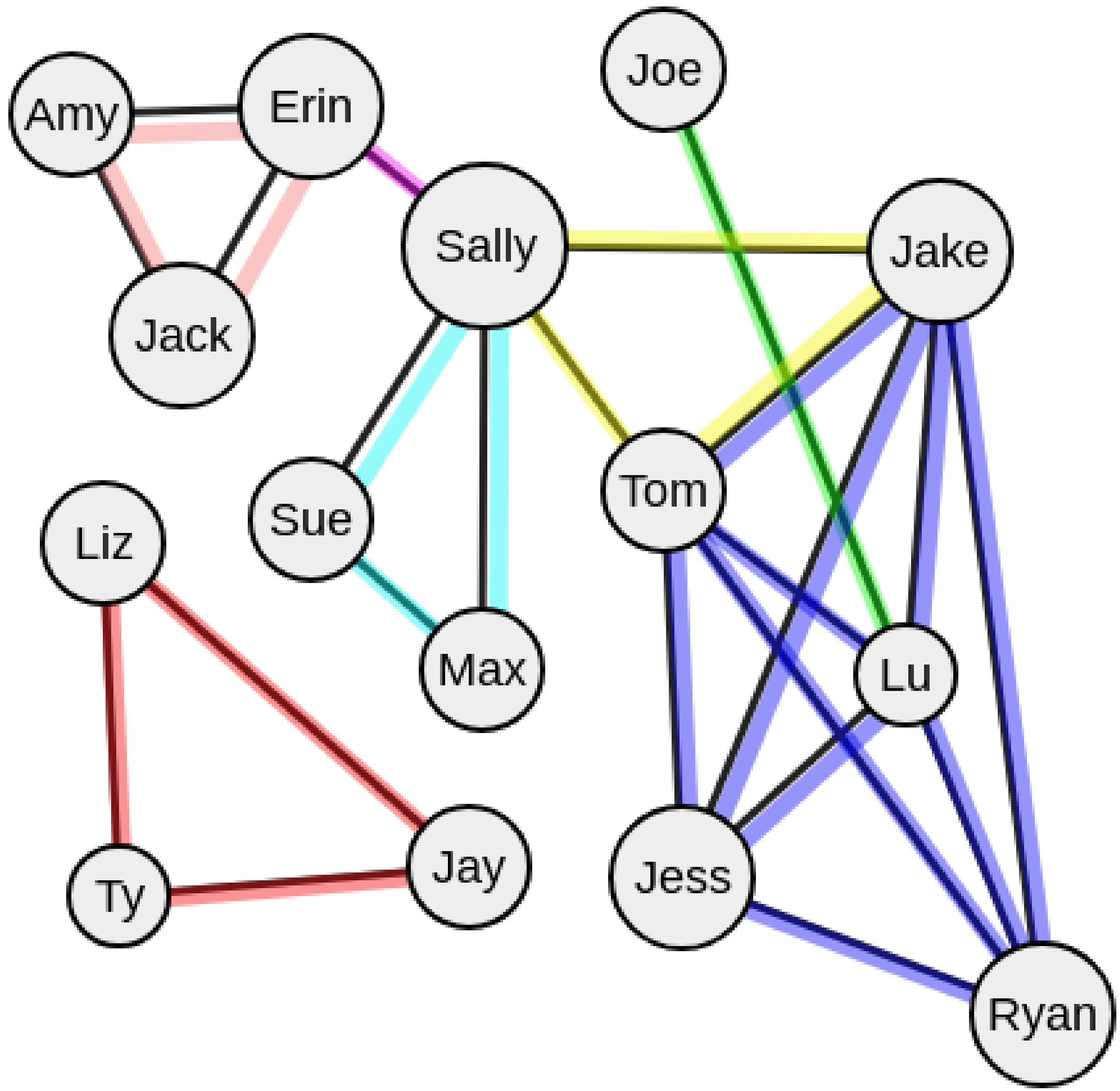
BronKerbosch(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  for each vertex v in P:
    BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
    P := P \ {v}
    X := X ⋃ {v}



- R={Jack, Amy}
- P={Erin}
- X={}
- v=Erin
- N(v) = {Amy, Jack, Sally}

- P={}
- X={Erin}

Result



Amy
Erin
Joe
Sally
Jake
Jack
Tom
Sue
Liz
Max
Lu
Jay
Ty
Jess
Ryan

Thanks