CSE 6140
CSE Algorithms

Final Project

Team X:

Shang-Tse Chen
Ruiming Lu
Chengwei Li
Lixi Zhao

12/01/2015

Georgia Institute of Technology

Outline

- **Branch-and-bound algorithm**
- **Approximate algorithm**
  - Edge Deletion
  - Maximum Degree Greedy
- **Local Search**
  - Simulated Annealing
  - Two stage Exchange with Weight Forgetting

Branch-and-Bound

Input: graph G

Output: opt_set, running_time

F ← {(∅,G)} // Frontier set of configurations

Upper bound = approximation_method(G)

while F ≠ ∅ do

  Choose (X)∈ F – the most "promising" configuration

  Expand (X), by making a choice(es)

  Let (X1), (X2), ..., (Xk,) be new configurations

  for each new configuration (Xi) do

    Check (Xi)

    if "solution found" then

      if cost(Xi) < Upper_bound then

        BEST ← (cost(Xi))

    if not "dead end" then

      if lb(Xi) < Upper_bound then

        F ← F ∪ {(Xi,Yi)}

return B

Initialize the Upper Bound:
approx_method(G)

Most "Promising"configuration:
Choose the node with has largest #edges

Expand:
1) choose node x, then C' include x;
2) not choose node x, then C' include u, where (u, v) and v=x

Lower bound:
|C'| + approx_method(G')/2

Implementation:
1) Recursion:
Karate.graph: ***5.3 ms***
Football.graph: ***Stack overflow***
2) Iteration:
Karate.graph: ***36.723 ms***
Football.graph: ***More than 1h***

Approximate Algorithms

Georgia Tech

Algorithm 1: Edge Deletion (ED)

Input : graph $G = (V, E)$
Output: vertex cover of G
$C := \emptyset$;
$E' := E$;
while $E' \neq \emptyset$ **do**
    Choose an arbitrary edge $(u, v) \in E'$;
    $C \leftarrow C \cup \{u, v\}$;
    remove from E' every edge incident on either u or v;
end
return C

Approximation ratio: 2

Algorithm 2: Maximum Degree Greedy (MDG)

Input : graph $G = (V, E)$
Output: vertex cover of G
$C := \emptyset$;
$E' := E$;
while $E' \neq \emptyset$ **do**
    Select a vertex u of maximum degree;
    $C \leftarrow C \cup \{u\}$;
    remove from E' every edge incident on u;
end
return C

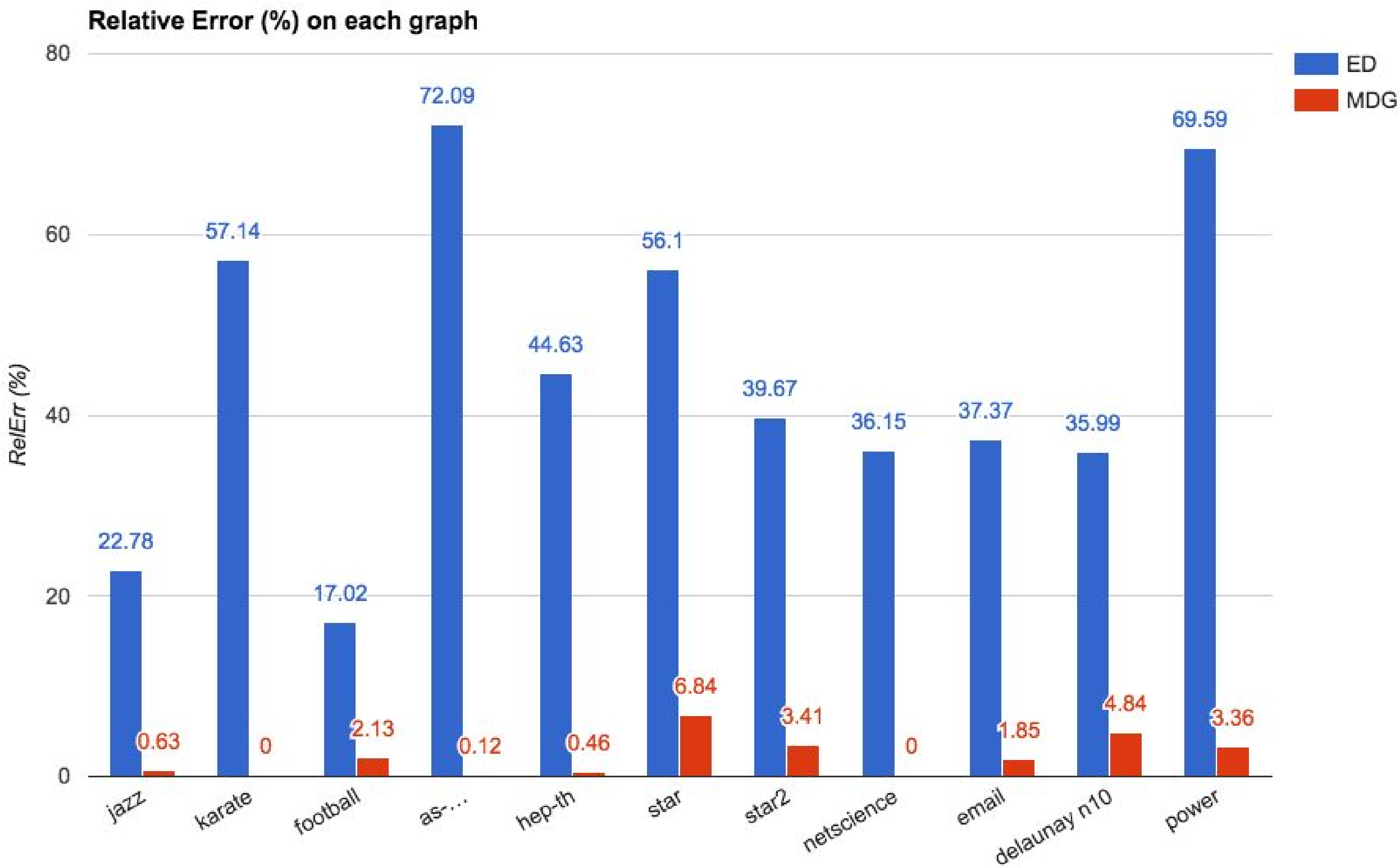
Approximation ratio: H(Δ)

Δ: maximum degree in G

H(Δ) = $1 + \frac{1}{2} + \cdots + \frac{1}{\Delta}$

≃ log(Δ)+0.57 when Δ is large

Approximate Algorithms (cont.)



Simulated Annealing (SA)

Algorithm 3: Simulated Annealing

```
Input  : graph G = (V, E), the cutoff time
Output: vertex cover of G
Construct initial vertex cover current;
while elapsed < cutoff do
    T := schedule(t);
    if current is a vertex cover then
        C := current ;
        remove one vertex from current;
        continue;
    end
    next := Random-Successor(current);
    Δ := cost(next) - cost(current);
    if Δ < 0 then
        current := next;
    else
        current := next with probability exp(−Δ/T)
    end
end
return C
```

Outline:

- Select a neighbor at random (swapping)
- If better than current state go there
- Otherwise, go there with some probability
- Probability goes down with time

Performance:

- Fast divergence in the beginning
- Very slow convergence afterwards

Efficient Simulated Annealing [X.Xu, J.Ma]

Key Idea: A vertex which has larger degree than the other vertices will be put into the cover with higher probability

The cost function

$$\text{cost} = \sum_{i=1}^{n} v_i + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \overline{v_i \vee v_j}$$

$$v_i = \begin{cases} 1 & \text{if vertex } i \text{ is in the cover} \\ 0 & \text{otherwise} \end{cases}$$

The acceptance function

$$p = e^{-\Delta/T} \quad \Longrightarrow \quad p = \begin{cases} e^{-\frac{\Delta(1-Deg(v_i))}{T}} & v^i = 1 \\ e^{-\frac{\Delta(1+Deg(v_i))}{T}} & v^i = 0 \end{cases}$$

Two stage exchange with Weight Forgetting

Algorithm 4: 2 Stage Exchange with weight Forgetting

Input : graph $G = (V, E)$, the $cutoff$ time
Output: vertex cover of G

initialize vertex cover C using approximation or greedy;
result $C^* \leftarrow C$;
uncovered graph $G^* \leftarrow \{\}$;
initialize weight for $E \in G$;
while $elapsed < cutoff$ **do**
    if G^* *is empty* **then**
        $C^* := C$;
        remove a vertex with minimum score from C;
        continue;
    end
    U := choose the neigbor with the largest calculated score;
    $C = C \setminus U$;
    subgraph $G^* = G^* \cup N(U)$,where $e \in N(U)$ is is the edge that no longer covered because of removal of U ;
    choose V of the largest score, where $V \in G^*$ and $V \neq U$;
    get rid of edges in G^* that got one end of V,adjust their weight;
    $C = C \cup V$;
    w(e)=w(e)+1 for $e \in G^*$;
    if $w(e) > \gamma$ **then**
        $w(e) = \lfloor w(e)^* \times \alpha \rfloor$
    end
end
return C^*

Key Ideas:

1 Two Stage Exchange:

selects the two vertices for exchanging separately in two stages.

(1)selects a vertex u in C with the highest dscore and removes it.

(2)selects a uniformly random uncovered edge e, and chooses one endpoint v of e with the higher dscore and add it back to C

A drawback of selecting two vertices to be exchanged simultaneously is that, the evaluation of a pair of vertices not only depends on the evaluations (such as dscore) of the two vertices, but also involves the relationship between the two vertices, like "do they belong to a same edge".

Two stage exchange with Weight Forgetting

Key idea:

2 Edge Weight Forgetting

(1) each edge is associated to a positive integer number as its weight, and each edge weight is initialized as one

(2) uncovered edges are increased by one in each iteration

(3) Decrease all weight if average reach some threshold

(3)Dscore calculated based on edge weight

(4)**When vertex choose, decrease its attached edge weight to make it less likely be choosen in the following round.**

Edge1.weight = 1000
Edge2.weight = 500
average reach threshold →
Edge1.weight = 100
Edge2.weight =50

Edge1.weight = 10
Edge2.weight = 50
over 100 iteration →
Edge1.weight = 100
Edge2.weight =150

Result :
Power.graph:
30s to achieve
optimal.

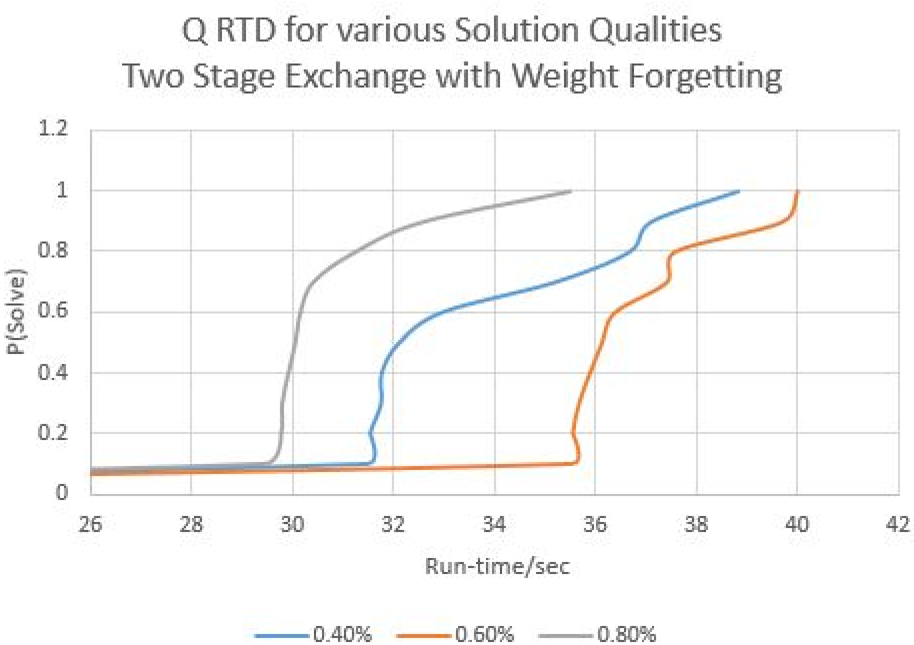
Local Search RTD

- **Influence of the Initialization**

Local Minimum Problem: Varies with initialization

Results differ with Approximation and Greedy initialization

- **QRTD Graph**



Thank you!