Towards a Theory of Greedy Algorithms

A Survey

Virginia Vassilevska

Theory Lunch

What is a Greedy Algorithm?

What is a Greedy Algorithm?

Wikipedia: "Greedy algorithms are algorithms which follow the problem solving meta-heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum."

What is a Greedy Algorithm?

Wikipedia: “Greedy algorithms are algorithms which follow the problem solving meta-heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum.”

LANL Megamath: “A greedy algorithm might also be called a *single-minded* algorithm, or an algorithm that gobbles up all of its favorites first.”

What is a Greedy Algorithm?

Wikipedia: “Greedy algorithms are algorithms which follow the problem solving meta-heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum.”

LANL Megamath: “A greedy algorithm might also be called a *single-minded* algorithm, or an algorithm that gobbles up all of its favorites first.”

Implicitly, in all definitions it is assumed that once a decision has been made, it cannot be reversed.

Properties of a Greedy Algorithm

Properties of a Greedy Algorithm

- views the instance as a set of items and the output as set of decisions, one per item;

Properties of a Greedy Algorithm

- views the instance as a set of items and the output as set of decisions, one per item;
- defines a criterion for best choices, which orders the items;

Properties of a Greedy Algorithm

- views the instance as a set of items and the output as set of decisions, one per item;
- defines a criterion for best choices, which orders the items;
- in the order chosen, makes irrevocable decisions for the data items one at a time;

Properties of a Greedy Algorithm

- views the instance as a set of items and the output as set of decisions, one per item;
- defines a criterion for best choices, which orders the items;
- in the order chosen, makes irrevocable decisions for the data items one at a time;
- when making a decision for the current item only considers the current and previous items, but not later items.

The Priority Model

Borodin, Nielsen, and Rackoff 2003 introduce priority algorithms.

The Priority Model

Borodin, Nielsen, and Rackoff 2003 introduce priority algorithms.

The instance I is a set of items of the same type, Γ.

The Priority Model

Borodin, Nielsen, and Rackoff 2003 introduce priority algorithms.

The instance I is a set of items of the same type, Γ.

Γ is a description of the item necessary for solving the particular problem.
For example, for vertex coloring, Γ may be $(v, N(v))$ where v is the name of a vertex and $N(v)$ is a list of the names of its neighbors.

The Priority Model cont.

The Priority Model cont.

The algorithm makes a decision σ_i for each item $\gamma_i \in I$. The set of possible decisions Σ depends on the particular problem.

The Priority Model cont.

The algorithm makes a decision σ_i for each item $\gamma_i \in I$. The set of possible decisions Σ depends on the particular problem.

For example, for k-Vertex Coloring, $\Sigma = \{1, \ldots, k\}$.

The Priority Model cont.

The algorithm makes a decision σ_i for each item $\gamma_i \in I$. The set of possible decisions Σ depends on the particular problem.

For example, for k-Vertex Coloring, $\Sigma = \{1, \ldots, k\}$.

For Vertex Cover we can have Γ be $(v, N(v))$ and $\Sigma = \{0, 1\}$.

The Priority Model cont.

The algorithm makes a decision σ_i for each item $\gamma_i \in I$. The set of possible decisions Σ depends on the particular problem.

For example, for k-Vertex Coloring, $\Sigma = \{1, \ldots, k\}$.

For Vertex Cover we can have Γ be $(v, N(v))$ and $\Sigma = \{0, 1\}$.

Or, can have Γ be (u, v) for (u, v) an edge, and $\Sigma = \{1, -1, 0\}$.

The Priority Model cont.

The algorithm makes a decision σ_i for each item $\gamma_i \in I$. The set of possible decisions Σ depends on the particular problem.

For example, for k-Vertex Coloring, $\Sigma = \{1, \ldots, k\}$.

For Vertex Cover we can have Γ be $(v, N(v))$ and $\Sigma = \{0, 1\}$.

Or, can have Γ be (u, v) for (u, v) an edge, and $\Sigma = \{1, -1, 0\}$.

We define Fixed and Adaptive priority algorithms.

Fixed Priority Algorithm

Input: $I = \{\gamma_1, \ldots, \gamma_n\} \subseteq \Gamma$
Output: $\{(\gamma_i, \sigma_i) | \sigma_i \in \Sigma, i = 1, \ldots, n\}$

Determine ordering π of *all* possible items of type Γ.

Order I corresponding to π.

Initialize $S \leftarrow \emptyset$; ordered list $L \leftarrow I$.

Repeat until L is empty

- pick first item γ_i in L
- choose decision $\sigma_i \in \Sigma$, dependent on already decided items
- add (γ_i, σ_i) to S
- remove γ_i from L

Output S

Adaptive Priority Algorithm

Input: $I = \{\gamma_1, \ldots, \gamma_n\} \subseteq \Gamma$
Output: $\{(\gamma_i, \sigma_i) | \sigma_i \in \Sigma, i = 1, \ldots, n\}$

Initialize $S \leftarrow \emptyset$; ordered list $L \leftarrow I$.

Repeat until L is empty

- pick ordering π of *all* items of type Γ, based only on observed items.
- order L corresponding to π
- pick first item γ_i in L and choose decision $\sigma_i \in \Sigma$, dependent on already decided items
- add (γ_i, σ_i) to S
- remove γ_i from L

Output S

Examples

Examples

The standard greedy approximation algorithm for Vertex Cover can be phrased as a fixed priority algorithm.

Examples

The standard greedy approximation algorithm for Vertex Cover can be phrased as a fixed priority algorithm.

Γ consists of items (u, v) corresponding to edges (u, v), $\Sigma = \{2, 0\}$.

Examples

The standard greedy approximation algorithm for Vertex Cover can be phrased as a fixed priority algorithm.

Γ consists of items (u, v) corresponding to edges (u, v), $\Sigma = \{2, 0\}$.

The algorithm starts with the items in *any order*; when deciding (u, v), decides 0 if either u or v has been taken in the vertex cover, and 2 if none of them have, thus putting both in the vertex cover.

Examples

The standard greedy approximation algorithm for Vertex Cover can be phrased as a fixed priority algorithm.

Γ consists of items (u, v) corresponding to edges (u, v), $\Sigma = \{2, 0\}$.

The algorithm starts with the items in *any order*; when deciding (u, v), decides 0 if either u or v has been taken in the vertex cover, and 2 if none of them have, thus putting both in the vertex cover.

The size of the vertex cover is then the sum of all decisions. The algorithm is a 2-approximation.

Clarkson's 2-approx for Weighted Vertex Cover is an adaptive priority algorithm.

Clarkson's 2-approx for Weighted Vertex Cover is an adaptive priority algorithm.

$C = \emptyset$

For all v, $W(v) = w(v)$, $D(v) = deg(v)$.

While $E \neq \emptyset$

Let v have minimum $W(v)/D(v)$.

For all $e = (u, v) \in E$:

$W(u) \leftarrow W(u) - W(u)/D(u)$

$D(u) \leftarrow D(u) - 1$

$E \leftarrow E \setminus \{e\}$

$W(v) \leftarrow 0$

$C \leftarrow C \cup \{v\}$

$V \leftarrow V - \{v\}$

Return C

Clarkson's 2-approx for Weighted Vertex Cover is an adaptive priority algorithm.

$C = \emptyset$

For all v, $W(v) = w(v)$, $D(v) = deg(v)$.

While $E \neq \emptyset$

Let v have minimum $W(v)/D(v)$.

For all $e = (u, v) \in E$:

$W(u) \leftarrow W(u) - W(u)/D(u)$

$D(u) \leftarrow D(u) - 1$

$E \leftarrow E \setminus \{e\}$

$W(v) \leftarrow 0$

$C \leftarrow C \cup \{v\}$

$V \leftarrow V - \{v\}$

Return C

Γ consists of items of the form $(v, w(v), N(v))$

L contains items of instance. $S = \emptyset$.

For all v, let $W(v) = w(v)$, $D(v) = deg(v)$.

While $L \neq \emptyset$

Sort L according to $W(v)/D(v)$.

Let $(v, w(v), N(v))$ be first in L.

For all neighbors u of v:

$W(u) \leftarrow W(u) - W(u)/D(u)$

$D(u) \leftarrow D(u) - 1$

If all neighbors of v have been accepted so far,

add $((v, w(v), N(v)), reject)$ to S.

Otherwise, add $((v, w(v), N(v)), accept)$ to S.

Remove $(v, w(v), N(v))$ from L.

Return S

Input Format Matters

Input Format Matters

Notice, in the definition of a priority algorithm, no constraint is placed on the running time of the order computation. But we think of greedy algorithms as efficient.

Input Format Matters

Notice, in the definition of a priority algorithm, no constraint is placed on the running time of the order computation. But we think of greedy algorithms as efficient.

Suppose Γ are the items of the form $(v, N(v), N^2(v))$ where where $N(v) = \{u_1, \ldots, u_k\}$ and $N^2(v) = (N(u_1), \ldots, N(u_k))$.

Input Format Matters

Notice, in the definition of a priority algorithm, no constraint is placed on the running time of the order computation. But we think of greedy algorithms as efficient.

Suppose Γ are the items of the form $(v, N(v), N^2(v))$ where where $N(v) = \{u_1, \ldots, u_k\}$ and $N^2(v) = (N(u_1), \ldots, N(u_k))$.

Then, a fixed priority algorithm can find a K-clique in a given graph, *i.e.* solve an NP-hard problem.

Input Format Matters

Notice, in the definition of a priority algorithm, no constraint is placed on the running time of the order computation. But we think of greedy algorithms as efficient.

Suppose Γ are the items of the form $(v, N(v), N^2(v))$ where where $N(v) = \{u_1, \ldots, u_k\}$ and $N^2(v) = (N(u_1), \ldots, N(u_k))$.

Then, a fixed priority algorithm can find a K-clique in a given graph, *i.e.* solve an NP-hard problem.

Hence, we may need to place some limitations on Γ.

Input Models for Graph Problems

Davis and Impagliazzo 2004 propose two input models:

Input Models for Graph Problems

Davis and Impagliazzo 2004 propose two input models:

- Node Model: Γ defines data items of the form $(v, w(v), N(v))$, where v is the node name, $w(v)$ is the node's weight and $N(v)$ is a list of the names of the neighbors of v.

Input Models for Graph Problems

Davis and Impagliazzo 2004 propose two input models:

- Node Model: Γ defines data items of the form $(v, w(v), N(v))$, where v is the node name, $w(v)$ is the node's weight and $N(v)$ is a list of the names of the neighbors of v.
- Edge Model: Γ defines data items of the form $((u,v), c(u,v), w(u), w(v))$ where (u,v) is an edge, $c(u,v)$ is a weight of the edge, and $w(u)$ and $w(v)$ are weights for its endpoints.

Input Models for Graph Problems

Davis and Impagliazzo 2004 propose two input models:

- Node Model: Γ defines data items of the form $(v, w(v), N(v))$, where v is the node name, $w(v)$ is the node's weight and $N(v)$ is a list of the names of the neighbors of v.

- Edge Model: Γ defines data items of the form $((u, v), c(u, v), w(u), w(v))$ where (u, v) is an edge, $c(u, v)$ is a weight of the edge, and $w(u)$ and $w(v)$ are weights for its endpoints.

Borodin *et al.* 2005 propose

- Edge Adjacency Model: Γ defines data items of the form $(v, w(v), e_1, \ldots, e_{deg(v)})$, where v is the node name, $w(v)$ is the node's weight and the rest is a list of the names of the edges incident on v.

Input Models cont.

Input Models cont.

Node Model can simulate Edge Adjacency model.

Input Models cont.

Node Model can simulate Edge Adjacency model.

Borodin *et al.* show that the Edge Adjacency model can be strictly weaker than the Node Model.

Input Models cont.

Node Model can simulate Edge Adjacency model.

Borodin *et al.* show that the Edge Adjacency model can be strictly weaker than the Node Model.

Edge Model can simulate Node Model?

Input Models cont.

Node Model can simulate Edge Adjacency model.

Borodin *et al.* show that the Edge Adjacency model can be strictly weaker than the Node Model.

Edge Model can simulate Node Model?

Edge Adjacency $<$ Node $\leq$ Edge.

Approximation Lower Bounds for Priority Algorithms

Approximation Lower Bounds for Priority Algorithms

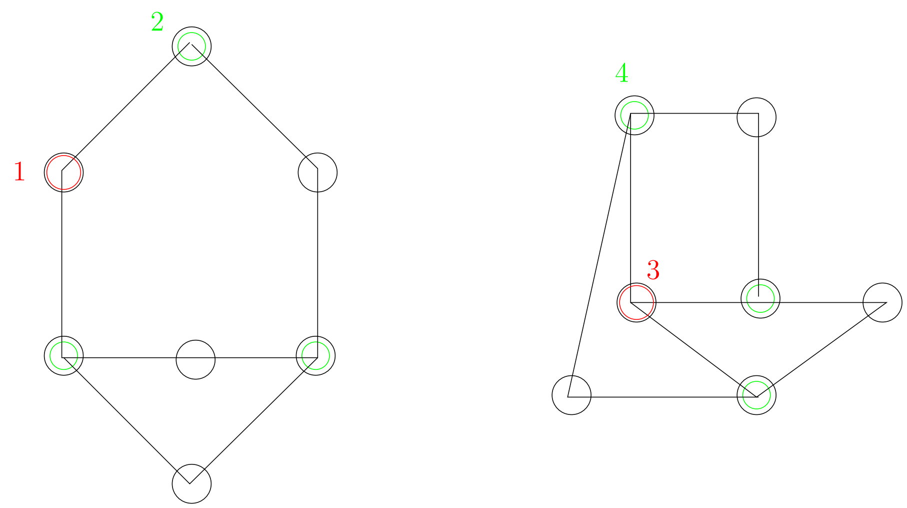
The weakness of priority algorithms is that they do not see the whole input, and must decide each item without knowing the remainder of the instance.

Approximation Lower Bounds for Priority Algorithms

The weakness of priority algorithms is that they do not see the whole input, and must decide each item without knowing the remainder of the instance.

The lower bound technique lets the adversary fool the algorithm by allowing several versions of the input and restricting to the worst input after each decision of the algo.

Vertex Cover Node Model Approximation Lower Bound

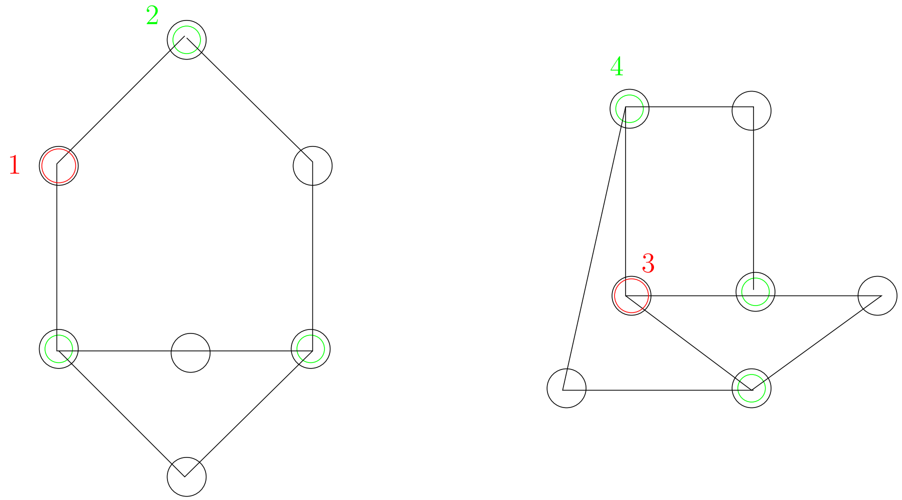


The adversary keeps all possible labelings of the above two graphs as possible instances.

Consider the first item the algorithm decides. There are four cases:

- accept degree 2 item $(A, (B, C))$: Pick labeling of left graph so that vertex 1 is labeled A and its neighbors are labeled B and C. Remove all other graph versions.
- reject degree 2 item $(A, (B, C))$: Pick labeling of left graph so that vertex 2 is labeled A and its neighbors are labeled B and C. Remove all other graph versions.

Vertex Cover Node Model Approximation Lower Bound



- accept degree 3 item $(A, (B, C, D))$: Pick labeling of right graph so that vertex 3 is labeled A and its neighbors are labeled B, C and D. Remove all other graph versions.

- reject degree 3 item $(A, (B, C, D))$: Pick labeling of right graph so that vertex 4 is labeled A and its neighbors are labeled B, C and D. Remove all other graph versions.

No adaptive priority algo (node model) can approximate VC to a better ratio than $4/3$.

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Repeat until G_t is empty:

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Repeat until G_t is empty:

Algorithm picks item γ_t and decision for it σ_t;

removes γ_t from G_t, adds (γ_t, σ_t) to PS and γ_t to PI.

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Repeat until G_t is empty:

Algorithm picks item γ_t and decision for it σ_t;

removes γ_t from G_t, adds (γ_t, σ_t) to PS and γ_t to PI.

Adversary replaces G_t with some $G_{t+1} \subseteq G_t$. Set $t = t + 1$.

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Repeat until G_t is empty:

Algorithm picks item γ_t and decision for it σ_t;

removes γ_t from G_t, adds (γ_t, σ_t) to PS and γ_t to PI.

Adversary replaces G_t with some $G_{t+1} \subseteq G_t$. Set $t = t + 1$.

Adversary presents solution S to PI.

Approximation Lower Bound Technique

Given $\rho > 0$, we give a game between the algorithm and the adversary:

Initialize empty partial instance PI, partial solution PS.

Adversary picks any subset G_1 of items of type Γ. Set $t = 1$.

Repeat until G_t is empty:

Algorithm picks item γ_t and decision for it σ_t;

removes γ_t from G_t, adds (γ_t, σ_t) to PS and γ_t to PI.

Adversary replaces G_t with some $G_{t+1} \subseteq G_t$. Set $t = t + 1$.

Adversary presents solution S to PI.

Algorithm wins if PI is not a valid instance, S is not a valid solution, or the approximation ratio is better than ρ.

2–Inapproximability for Weighted Vertex Cover in the Node Model

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

If the Algorithm observes an item $(v, w(v), N(v))$, the Adversary removes the other item corresponding to v.

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

If the Algorithm observes an item $(v, w(v), N(v))$, the Adversary removes the other item corresponding to v.

The Adversary waits until one of the following occurs:

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

If the Algorithm observes an item $(v, w(v), N(v))$, the Adversary removes the other item corresponding to v.

The Adversary waits until one of the following occurs:

- Algorithm accepts item of weight n^2: Adversary fixes weights of nodes on the other side to 1, and unseen same side items to weights of n^2.

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

If the Algorithm observes an item $(v, w(v), N(v))$, the Adversary removes the other item corresponding to v.

The Adversary waits until one of the following occurs:

- Algorithm accepts item of weight n^2: Adversary fixes weights of nodes on the other side to 1, and unseen same side items to weights of n^2.
- Algorithm rejects item. Adversary fixes unseen item weights on the same side to 1, and those on the other side to n^2.

2–Inapproximability for Weighted Vertex Cover in the Node Model

Adversary picks $K_{n,n}$, and lets Γ define items of the type $(v, w(v), N(v))$ where $w(v)$ is either 1 or n^2. Two items per vertex.

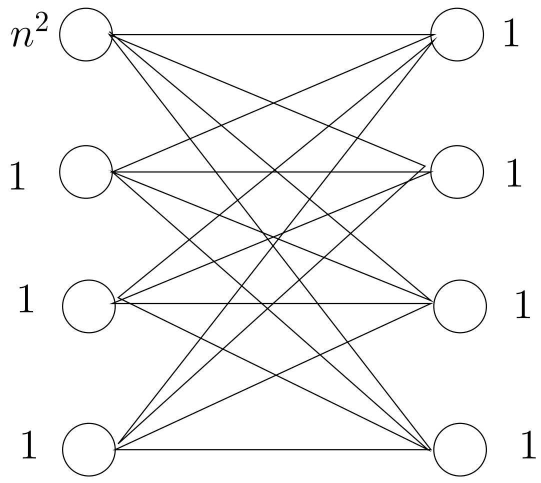
If the Algorithm observes an item $(v, w(v), N(v))$, the Adversary removes the other item corresponding to v.

The Adversary waits until one of the following occurs:

- Algorithm accepts item of weight n^2: Adversary fixes weights of nodes on the other side to 1, and unseen same side items to weights of n^2.
- Algorithm rejects item. Adversary fixes unseen item weights on the same side to 1, and those on the other side to n^2.
- Algorithm accepts $n-1$ items of weight 1 from the same side of $K_{n,n}$. Adversary fixes weight of last item in A to n^2, and all unseen nodes on the other side to weight 1.

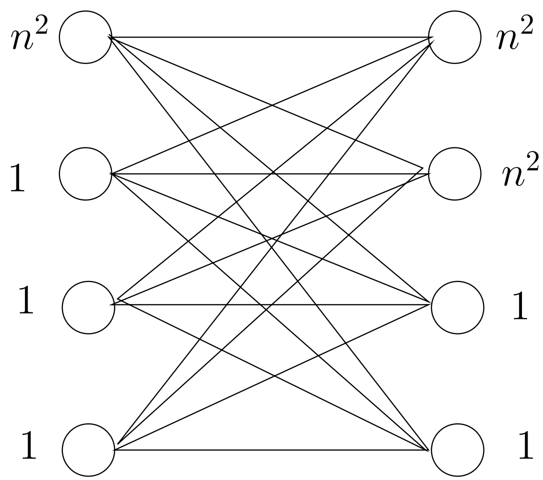
2–Inapproximability for Weighted Vertex Cover in the Node Model

- Algorithm accepts item of weight n^2: Adversary returns nodes on the other side as his solution of weight n. Algorithm has partial solution of weight at least n^2.



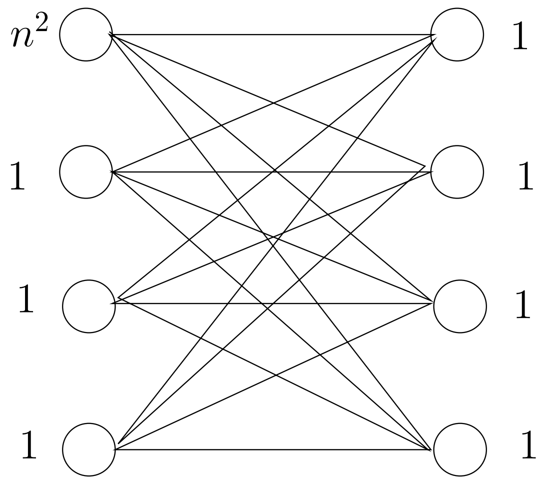
2–Inapproximability for Weighted Vertex Cover in the Node Model

- Algorithm rejects item. Adversary returns nodes from the same side as his solution of weight at most $n^2 + n - 1$. Algorithm must accept all items on the other side - at least two of these have weight n^2, and hence the algorithm solution is of weight at least $2n^2 + n - 2$.



2–Inapproximability for Weighted Vertex Cover in the Node Model

- Algorithm accepts $n-1$ items of weight 1 from the same side of $K_{n,n}$.
  Adversary returns nodes on the other side as his solution of weight n.
  The Algorithm can at best return a solution of weight $2n-1$.



Various Other Approximation Lower Bounds

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.
- Steiner tree with weights between 1 and 2: no adaptive priority algorithm can achieve a ratio better than 1.18. Best adaptive priority approx is 1.75.

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.
- Steiner tree with weights between 1 and 2: no adaptive priority algorithm can achieve a ratio better than 1.18. Best adaptive priority approx is 1.75.
- Degree-3 Independent Set - no adaptive priority algorithm in the node model can achieve a ratio better than $3/2$. Best approx is $5/3$.

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.
- Steiner tree with weights between 1 and 2: no adaptive priority algorithm can achieve a ratio better than 1.18. Best adaptive priority approx is 1.75.
- Degree-3 Independent Set - no adaptive priority algorithm in the node model can achieve a ratio better than $3/2$. Best approx is $5/3$.
- Weighted Independent Set on Cycles - no adap. prio. algorithm in the edge adj. model can achieve a ratio better than $3/2$. Best adap. prio. approx. is $3/2$.

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.
- Steiner tree with weights between 1 and 2: no adaptive priority algorithm can achieve a ratio better than 1.18. Best adaptive priority approx is 1.75.
- Degree-3 Independent Set - no adaptive priority algorithm in the node model can achieve a ratio better than $3/2$. Best approx is $5/3$.
- Weighted Independent Set on Cycles - no adap. prio. algorithm in the edge adj. model can achieve a ratio better than $3/2$. Best adap. prio. approx. is $3/2$.
- Vertex Coloring - any fixed priority algo in the edge adjacency model must use at least $d+1$ colors on bipartite graphs of max degree d.

Various Other Approximation Lower Bounds

- No fixed priority algo can approximate the (positive weights) shortest path problem within any ratio. No adaptive priority algo can approximate the shortest path problem within any ratio, when negative weights are allowed.
- Steiner tree with weights between 1 and 2: no adaptive priority algorithm can achieve a ratio better than 1.18. Best adaptive priority approx is 1.75.
- Degree-3 Independent Set - no adaptive priority algorithm in the node model can achieve a ratio better than $3/2$. Best approx is $5/3$.
- Weighted Independent Set on Cycles - no adap. prio. algorithm in the edge adj. model can achieve a ratio better than $3/2$. Best adap. prio. approx. is $3/2$.
- Vertex Coloring - any fixed priority algo in the edge adjacency model must use at least $d + 1$ colors on bipartite graphs of max degree d.
- Set Cover - no adaptive priority algo can achieve a ratio better than $\ln n - \ln \ln n + \Theta(1)$. No fixed priority algo can achieve a ratio better than $(1 - \epsilon)n$, for any $\epsilon > 0$.

Extensions

Extensions

Acceptance-first: priority algorithms in the accept/reject context which do not accept any items after the first rejected item.

Extensions

Acceptance-first: priority algorithms in the accept/reject context which do not accept any items after the first rejected item.

Memoryless: only remember accepted items.

Extensions

Acceptance-first: priority algorithms in the accept/reject context which do not accept any items after the first rejected item.

Memoryless: only remember accepted items.

Angelopoulos suggestion 2004: indistinguishible items receive the same priority - this increases the power of the adversary.

Extensions

Acceptance-first: priority algorithms in the accept/reject context which do not accept any items after the first rejected item.

Memoryless: only remember accepted items.

Angelopoulos suggestion 2004: indistinguishible items receive the same priority - this increases the power of the adversary.

BT algorithms: allows branching on different decisions and backtracking; generalizes both priority algorithms and dynamic programming. Alekhnovich *et al.* prove various lower bounds in this more general model.

Thank You!

References

- Davis, S. and Impagliazzo, R. 2004. Models of greedy algorithms for graph problems. SODA 2004.
- Allan Borodin and Joan Boyar and Kim S. Larsen, Priority Algorithms for Graph Optimization Problems, 2005
- Spyros Angelopoulos, Order-Preserving Transformations and Greedy-Like Algorithms. WAOA 2004: 197-210
- Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, Toniann Pitassi: Toward a Model for Backtracking and Dynamic Programming. IEEE Conference on Computational Complexity 2005: 308-322