Lecture 12. Online Algorithms

Guochuan Zhang

Zhejiang University

AOR, December 8, 2021

Combinatorial Optimization

A Standard Model

$$\max(\min) \quad F(x)$$

$$s.t. \quad x \in Q$$

where Q is a finite set

Problem-Solving

Our concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

What make things tough?

- Limited computing resources
- Incomplete information

Tradeoffs

- Running Times versus Performance Bounds
- Content of Information versus Performance Bounds

Problem-Solving

Our concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

What make things tough?

- Limited computing resources
- Incomplete information

Tradeoffs

- Running Times versus Performance Bounds
- Content of Information versus Performance Bounds

Problem-Solving

Our concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

What make things tough?

- Limited computing resources
- Incomplete information

Tradeoffs

- Running Times versus Performance Bounds
- Content of Information versus Performance Bounds

Problem-Solving

Our concerns

- Efficiency: How fast to obtain a solution?
- Effectiveness: How good the solution is?

What make things tough?

- Limited computing resources
- Incomplete information

Tradeoffs

- Running Times versus Performance Bounds
- Content of Information versus Performance Bounds

Roadmap

Purely Online

- Information arrival step by step
- None about future
- Irrevocable decisions

Semi Online

- Partial information available
- Changing previously made decisions

Roadmap

Purely Online

- Information arrival step by step
- None about future
- Irrevocable decisions

Semi Online

- Partial information available
- Changing previously made decisions

Roadmap

Purely Online

- Information arrival step by step
- None about future
- Irrevocable decisions

Semi Online

- Partial information available
- Changing previously made decisions

Roadmap

Purely Online

- Information arrival step by step
- None about future
- Irrevocable decisions

Semi Online

- Partial information available
- Changing previously made decisions

Roadmap

Purely Online

- Information arrival step by step
- None about future
- Irrevocable decisions

Semi Online

- Partial information available
- Changing previously made decisions

Roadmap

Online with Advice

- Algorithms get some bits of advice
- How much knowledge of the future needs to achieve a certain competitive ratio

Online with Imperfect Information

- Untrusted advice
- Predicted information

Roadmap

Online with Advice

- Algorithms get some bits of advice
- How much knowledge of the future needs to achieve a certain competitive ratio

Online with Imperfect Information

- Untrusted advice
- Predicted information

Standard Online Model

Settings

- (Adversary) Input $I = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$
- (Algorithm) Output $O = \{y_1, y_2, \ldots, y_n\}$

Competitive Analysis

Let A be an online algorithm, OPT be an optimal offline algorithm with full information in advance.

- For an input I, let $A(I)$ denote the objective value of the output produced by A, and $OPT(I)$ be the optimal value
- For minimization problems, A is c-competitive if there exists a constant β, for any I, $A(I) \le c \cdot OPT(I) + \beta$
- The infimum of c is called competitive ratio

Online Problems I

Online Search on a Line

- Given an infinite line and a start point S, find the unique point D whose location is unknown

Online Bidding

- A player is to guess an unknown (hidden) integral value $u \geq 1$. The cost is the sum of bids the player has submitted until the bid value is at least u

Ski Rental

- In a skiing season, a ski beginner has to decide on each day whether to buy skis or continue renting one, where renting skis for 1\$ a time or buying skis for d\$.

Online Problems I

Online Search on a Line

- Given an infinite line and a start point S, find the unique point D whose location is unknown

Online Bidding

- A player is to guess an unknown (hidden) integral value $u \geq 1$. The cost is the sum of bids the player has submitted until the bid value is at least u

Ski Rental

- In a skiing season, a ski beginner has to decide on each day whether to buy skis or continue renting one, where renting skis for 1\$ a time or buying skis for d\$.

Online Problems I

Online Search on a Line

- Given an infinite line and a start point S, find the unique point D whose location is unknown

Online Bidding

- A player is to guess an unknown (hidden) integral value $u \geq 1$. The cost is the sum of bids the player has submitted until the bid value is at least u

Ski Rental

- In a skiing season, a ski beginner has to decide on each day whether to buy skis or continue renting one, where renting skis for 1\$ a time or buying skis for d\$.

Online Problems II

Paging Problem

- Two levels of memory: fast memory consisting of k pages (cache), slow memory consisting of n pages ($k \ll n$)
- Accessing a page contained in the cache has no cost
- When accessing a page not in the cache, it must first be brought at a cost of 1 to the cache
- Manage the cache to minimize the whole cost in dealing with a sequence of requests

Online Problems II

k-Server Problem

- Given a weighted network $G = (V, E)$, k servers are initially located on some of the vertices.
- Vertex requests $\sigma = (r_1, r_2, \ldots, r_n)$ arrive one by one.
- As soon as a request appears, one has to send one of the servers to the vertex if no servers are there.
- The goal is to minimize the total distance moved to serve all requests.

Online Problems II

Online Scheduling and Load Balancing

- A set of m identical machines
- A list of jobs arrives one by one, with a processing time p_j
- Upon arrival of a job, it has to be assigned immediately on one of the m machines without any information of subsequent jobs
- The goal is to give a schedule of all coming jobs so that the makespan $\max_i \sum_{j \in M_i} p_j$ is minimized

Online Bin Packing

- Upon arrival of an item, it is irrevocably packed in to a bin
- With a smallest number of bins to accommodate all items

Online Problems II

Online Scheduling and Load Balancing

- A set of m identical machines
- A list of jobs arrives one by one, with a processing time p_j
- Upon arrival of a job, it has to be assigned immediately on one of the m machines without any information of subsequent jobs
- The goal is to give a schedule of all coming jobs so that the makespan $\max_i \sum_{j \in M_i} p_j$ is minimized

Online Bin Packing

- Upon arrival of an item, it is irrevocably packed in to a bin
- With a smallest number of bins to accommodate all items

Online Problems III

Online Throughput Maximization

- A set of m identical machines
- A list of jobs with the same length p and different deadline arrives over time
- The deadline of a job is unknown until its arrival
- Decide if an available job is accepted. If yes, schedule it in due, otherwise throw it away
- Accept as many jobs as possible

A Summary on the Models

- Information is the answer to seek
- Information is released in a list, or over time, or based on dependencies

Online Problems III

Online Throughput Maximization

- A set of m identical machines
- A list of jobs with the same length p and different deadline arrives over time
- The deadline of a job is unknown until its arrival
- Decide if an available job is accepted. If yes, schedule it in due, otherwise throw it away
- Accept as many jobs as possible

A Summary on the Models

- Information is the answer to seek
- Information is released in a list, or over time, or based on dependencies

A Framework

Positive Results

Try to design a good, better online algorithms in terms of competitive ratio

Negative Results

Try to construct bad examples showing no online algorithms can do better

Best Possible/Optimal Online Algorithms

- Matched bounds
- But the competitive ratio might be too big comparing with offline algorithms

A Framework

Positive Results

Try to design a good, better online algorithms in terms of competitive ratio

Negative Results

Try to construct bad examples showing no online algorithms can do better

Best Possible/Optimal Online Algorithms

- Matched bounds
- But the competitive ratio might be too big comparing with offline algorithms

A Framework

Positive Results

Try to design a good, better online algorithms in terms of competitive ratio

Negative Results

Try to construct bad examples showing no online algorithms can do better

Best Possible/Optimal Online Algorithms

- Matched bounds
- But the competitive ratio might be too big comparing with offline algorithms

The k-Server Problem

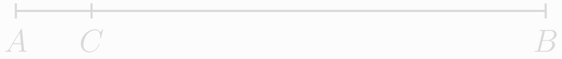
First Glance

A Greedy Policy

- General rule: for any request-answer system, a greedy online algorithm processes each request in order to minimize the cost (or maximize the profit) on the input sequence seen so far.
- Serve each request with a nearest server.

Bad Instance

- Consider two servers on a 3-node graph, where $d(A, C) < d(B, C)$.
- The input sequence is $(A, B, C, A, C, A, C, A, \dots)$.



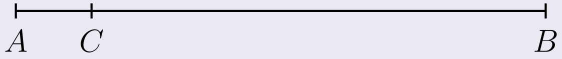
First Glance

A Greedy Policy

- General rule: for any request-answer system, a greedy online algorithm processes each request in order to minimize the cost (or maximize the profit) on the input sequence seen so far.
- Serve each request with a nearest server.

Bad Instance

- Consider two servers on a 3-node graph, where $d(A, C) < d(B, C)$.
- The input sequence is $(A, B, C, A, C, A, C, A, \ldots)$.



The Offline Problem

Known Results

- Dynamic programming
- Reduce to a minimum cost/max flow problem $(O(kn^2))$.

A Lower Bound

Theorem 1. Let $\mathcal{M}$ be any metric space with at least $k+1$ points. Then k is a lower bound on the competitive ratio of any online algorithm.

Proof

- A is any online algorithm. Fix $k+1$ points in $\mathcal{M}$, and let the initial configuration of A be $\{1, 2, \ldots, k\}$.
- $\sigma = (r_1, r_2, \ldots, r_n)$ requests at each step the unique point unoccupied by A. $A(\sigma) = \sum_{i=0}^{n-1} d(r_i', r_{i+1})$, where r_i' is the location of the server moving to r_{i+1}..

A Lower Bound

Proof

- $\sigma = (r_1, r_2, \ldots, r_n)$ requests at each step the unique point unoccupied by A. $A(\sigma) = \sum_{i=0}^{n-1} d(r'_i, r_{i+1})$, where r'_i is the location of the server moving to r_{i+1}.
- Maintain k (offline) algorithms, together with the (online) algorithm A, each holding a distinct configuration.
- For the request r_{i+1}, only the algorithm A leaves r_{i+1} unoccupied. While A moves a server from some point p to r_{i+1}, the algorithm leaving p unoccupied moves the server at r_{i+1} in the opposite direction to p.
- For each request, the total distance moved by the k algorithms is the same as A.

A Lower Bound

Proof

- $\sigma = (r_1, r_2, \ldots, r_n)$ requests at each step the unique point unoccupied by A. $A(\sigma) = \sum_{i=0}^{n-1} d(r'_i, r_{i+1})$, where r'_i is the location of the server moving to r_{i+1}.
- Maintain k (offline) algorithms, together with the (online) algorithm A, each holding a distinct configuration.
- For the request r_{i+1}, only the algorithm A leaves r_{i+1} unoccupied. While A moves a server from some point p to r_{i+1}, the algorithm leaving p unoccupied moves the server at r_{i+1} in the opposite direction to p.
- For each request, the total distance moved by the k algorithms is the same as A.

A Lower Bound

Proof

- $\sigma = (r_1, r_2, \ldots, r_n)$ requests at each step the unique point unoccupied by A. $A(\sigma) = \sum_{i=0}^{n-1} d(r_i', r_{i+1})$, where r_i' is the location of the server moving to r_{i+1}.
- Maintain k (offline) algorithms, together with the (online) algorithm A, each holding a distinct configuration.
- For the request r_{i+1}, only the algorithm A leaves r_{i+1} unoccupied. While A moves a server from some point p to r_{i+1}, the algorithm leaving p unoccupied moves the server at r_{i+1} in the opposite direction to p.
- For each request, the total distance moved by the k algorithms is the same as A.

A Lower Bound

Proof

- $\sigma = (r_1, r_2, \ldots, r_n)$ requests at each step the unique point unoccupied by A. $A(\sigma) = \sum_{i=0}^{n-1} d(r_i', r_{i+1})$, where r_i' is the location of the server moving to r_{i+1}.
- Maintain k (offline) algorithms, together with the (online) algorithm A, each holding a distinct configuration.
- For the request r_{i+1}, only the algorithm A leaves r_{i+1} unoccupied. While A moves a server from some point p to r_{i+1}, the algorithm leaving p unoccupied moves the server at r_{i+1} in the opposite direction to p.
- For each request, the total distance moved by the k algorithms is the same as A.

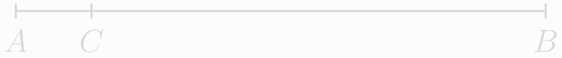
k-Server on a Line

Algorithm DC

As a request is coming, do:

- if the request falls outside the convex hull of the servers, serve it with the nearest server;
- if the request is in between two adjacent servers, move both towards the request at the same speed until one reaches it (if two servers occupy the same point, choose one of them) .

Revisit the Example for Greedy



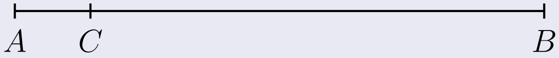
k-Server on a Line

Algorithm DC

As a request is coming, do:

- if the request falls outside the convex hull of the servers, serve it with the nearest server;
- if the request is in between two adjacent servers, move both towards the request at the same speed until one reaches it (if two servers occupy the same point, choose one of them) .

Revisit the Example for Greedy



Analysis of DC

Theorem 2. DC is k-competitive.

Interleaving Moves

- OPT and DC play a game by moving their own servers in turn.
- Define a potential function $\Phi \geq 0$. Φ_0 is the initial value before the game starts, while Φ_i is the value after the i-th event.
- If OPT moves a distance d, Φ is increased by at most kd.
- If DC moves a distance d, Φ is decreased by at least d.

Amortized Analysis

- The sum of increments of Φ is $\Phi_n - \Phi_0$, which is at most $k \cdot OPT(\sigma) - DC(\sigma)$, giving the desired bound.

Analysis of DC

Theorem 2. DC is k-competitive.

Interleaving Moves

- OPT and DC play a game by moving their own servers in turn.
- Define a potential function $\Phi \geq 0$. Φ_0 is the initial value before the game starts, while Φ_i is the value after the i-th event.
- If OPT moves a distance d, Φ is increased by at most kd.
- If DC moves a distance d, Φ is decreased by at least d.

Amortized Analysis

- The sum of increments of Φ is $\Phi_n - \Phi_0$, which is at most $k \cdot OPT(\sigma) - DC(\sigma)$, giving the desired bound.

Analysis of DC

Theorem 2. DC is k-competitive.

Interleaving Moves

- OPT and DC play a game by moving their own servers in turn.
- Define a potential function $\Phi \geq 0$. Φ_0 is the initial value before the game starts, while Φ_i is the value after the i-th event.
- If OPT moves a distance d, Φ is increased by at most kd.
- If DC moves a distance d, Φ is decreased by at least d.

Amortized Analysis

- The sum of increments of Φ is $\Phi_n - \Phi_0$, which is at most $k \cdot OPT(\sigma) - DC(\sigma)$, giving the desired bound.

Analysis of DC

Theorem 2. DC is k-competitive.

Interleaving Moves

- OPT and DC play a game by moving their own servers in turn.
- Define a potential function $\Phi \geq 0$. Φ_0 is the initial value before the game starts, while Φ_i is the value after the i-th event.
- If OPT moves a distance d, Φ is increased by at most kd.
- If DC moves a distance d, Φ is decreased by at least d.

Amortized Analysis

- The sum of increments of Φ is $\Phi_n - \Phi_0$, which is at most $k \cdot OPT(\sigma) - DC(\sigma)$, giving the desired bound.

Analysis of DC

Theorem 2. DC is k-competitive.

Interleaving Moves

- OPT and DC play a game by moving their own servers in turn.
- Define a potential function $\Phi \geq 0$. Φ_0 is the initial value before the game starts, while Φ_i is the value after the i-th event.
- If OPT moves a distance d, Φ is increased by at most kd.
- If DC moves a distance d, Φ is decreased by at least d.

Amortized Analysis

- The sum of increments of Φ is $\Phi_n - \Phi_0$, which is at most $k \cdot OPT(\sigma) - DC(\sigma)$, giving the desired bound.

Analysis of DC

Potential Function

- M_{min}: the minimum cost perfect matching between OPT's and DC's servers
- s_i: DC's servers, for $i = 1, 2, \ldots, k$
- $\Sigma = \sum_{i<j} d(s_i, s_j)$
- Potential function $\Phi = k \cdot M_{min} + \Sigma \geq 0$

OPT Moves

- If OPT moves a distance d, Φ is increased by at most kd.
- Σ remains.
- M_{min} cannot increase by more than d.

Analysis of DC

Potential Function

- M_{min}: the minimum cost perfect matching between OPT's and DC's servers
- s_i: DC's servers, for $i = 1, 2, \ldots, k$
- $\Sigma = \sum_{i<j} d(s_i, s_j)$
- Potential function $\Phi = k \cdot M_{min} + \Sigma \geq 0$

OPT Moves

- If OPT moves a distance d, Φ is increased by at most kd.
- Σ remains.
- M_{min} cannot increase by more than d.

Analysis of DC

If DC moves a distance d, Φ is decreased by at least d.

DC: Single Server Moves

- The moving server is an extreme point of the convex hull over all the servers. It moves away from the others.
- $\Sigma = \sum_{i<j} d(s_i, s_j)$ increases by $(k-1)d$.
- There exists a minimum cost matching in which the moving server matched the request. M_{min} decreases by d.

DC: Two Servers Move

- Both move a distance d. Σ decreases by $2d$.
- The server matched the request point decreases M_{min} by at least d, while the other moving server may move away from its match. M_{min} does not increase at all.

Analysis of DC

If DC moves a distance d, Φ is decreased by at least d.

DC: Single Server Moves

- The moving server is an extreme point of the convex hull over all the servers. It moves away from the others.
- $\Sigma = \sum_{i<j} d(s_i, s_j)$ increases by $(k-1)d$.
- There exists a minimum cost matching in which the moving server matched the request. M_{min} decreases by d.

DC: Two Servers Move

- Both move a distance d. Σ decreases by $2d$.
- The server matched the request point decreases M_{min} by at least d, while the other moving server may move away from its match. M_{min} does not increase at all.

k-Server on a Tree

Algorithm DC-Tree

Assume a request is coming. A server is called neighbor of the request if there is no other server on the simple path between the server and the request. We only name one the neighbor if at least two servers are located at the same spot.

- At each time, all the servers neighboring the request are moving in a constant speed towards the request.
- DC-Tree is k-competitive.

k-Server on a Tree

Algorithm DC-Tree

Assume a request is coming. A server is called neighbor of the request if there is no other server on the simple path between the server and the request. We only name one the neighbor if at least two servers are located at the same spot.

- At each time, all the servers neighboring the request are moving in a constant speed towards the request.
- DC-Tree is k-competitive.

k-Server on a General Network

For any metric space $\mathcal{M}$, a configuration C of an algorithm is a set of k points where the servers are located.

Work-Function Algorithm (WFA)

- σ_i: the request sequence so far
- C: configuration of WFA after serving σ_i
- Work Function $W(C)$ $(w_{\sigma_i}(C))$ is the optimal offline cost of sequentially serving σ_i starting from the initial configuration C_0 and ending at C.
- As the next request $r = r_{i+1}$ arrives, WFA moves a server $s \in C$ to r, where

$$s = \arg\min_{x \in C}\{w(C - x + r) + d(x, r)\}.$$

k-Server on a General Network

For any metric space $\mathcal{M}$, a configuration C of an algorithm is a set of k points where the servers are located.

Work-Function Algorithm (WFA)

- σ_i: the request sequence so far
- C: configuration of WFA after serving σ_i
- Work Function $W(C)$ $(w_{\sigma_i}(C))$ is the optimal offline cost of sequentially serving σ_i starting from the initial configuration C_0 and ending at C.
- As the next request $r = r_{i+1}$ arrives, WFA moves a server $s \in C$ to r, where

$$s = \arg\min_{x \in C} \{w(C - x + r) + d(x, r)\}.$$

k-Server on a General Network

For any metric space $\mathcal{M}$, a configuration C of an algorithm is a set of k points where the servers are located.

Work-Function Algorithm (WFA)

- σ_i: the request sequence so far
- C: configuration of WFA after serving σ_i
- Work Function $W(C)$ $(w_{\sigma_i}(C))$ is the optimal offline cost of sequentially serving σ_i starting from the initial configuration C_0 and ending at C.
- As the next request $r = r_{i+1}$ arrives, WFA moves a server $s \in C$ to r, where

$$s = \arg\min_{x \in C}\{w(C - x + r) + d(x, r)\}.$$

k-Server on a General Network

Analysis of WFA

Algorithm WFA is $(2k-1)$-competitive.

k-Server Conjecture

- k-server problem adnits a k-competitive online algorithm.
- Algorithm WFA is k-competitive.

Reference: Allan Borodin and Ran El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press 1998.

k-Server on a General Network

Analysis of WFA

Algorithm WFA is $(2k-1)$-competitive.

k-Server Conjecture

- k-server problem adnits a k-competitive online algorithm.
- Algorithm WFA is k-competitive.

Reference: Allan Borodin and Ran El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press 1998.

k-Server on a General Network

Analysis of WFA

Algorithm WFA is $(2k-1)$-competitive.

k-Server Conjecture

- k-server problem adnits a k-competitive online algorithm.
- Algorithm WFA is k-competitive.

Reference: Allan Borodin and Ran El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press 1998.

Homework

In-Class-Work

If one bit advice is allowed, how much you can achieve for those in Problems I (Page 7)?

- Search on a line
- Online bidding
- Ski rental

After-Class-Work

Consider the 2-server problem on a 3-node cycle. Try to design a 2-competitive algorithm.