COMP 6651

Lecture on Online Algorithms and Competitive Analysis, Part 1

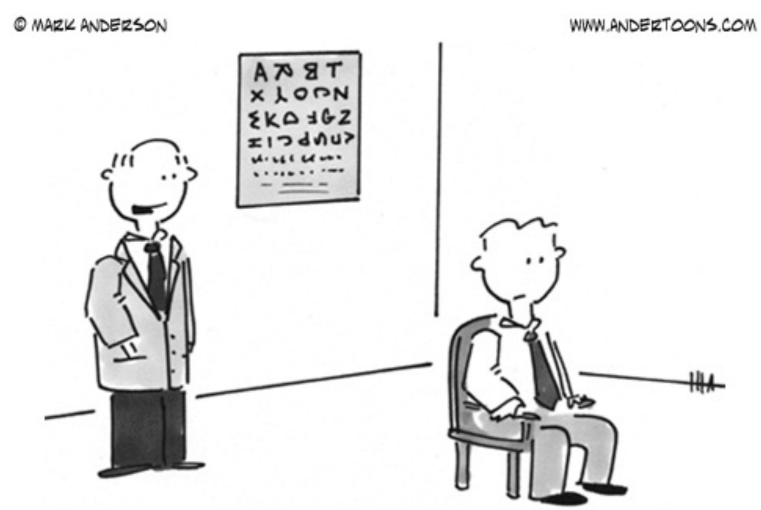
Denis Pankratov

Introduction

What is this topic about?

Hindsight

“understanding of a situation or event only after it has happened or developed.”



“OK, let’s check your hindsight.”

Offline problems

Given input, examine it in **its entirety**, and produce some output

Examples:

given a graph, find a minimum spanning tree

given a course calendar, schedule exams with no conflicts

given a Boolean formula, find a satisfying assignment

given a flow network, find a maximum flow

etc…

Online problems

Require decisions to be done in **real time without seeing future input**

Examples:

patients arrive at a clinic, assign them to be seen by doctors

jobs arrive at a supercomputer, assign them to computing units

packets arrive at a switch port, forward them to an outgoing port

user clicks on a website, decide which ad to display, etc…

In all cases, decisions are either fully or at least partially ***irrevocable***

Measure of quality of an online solution

“online” = “irrevocable decisions”

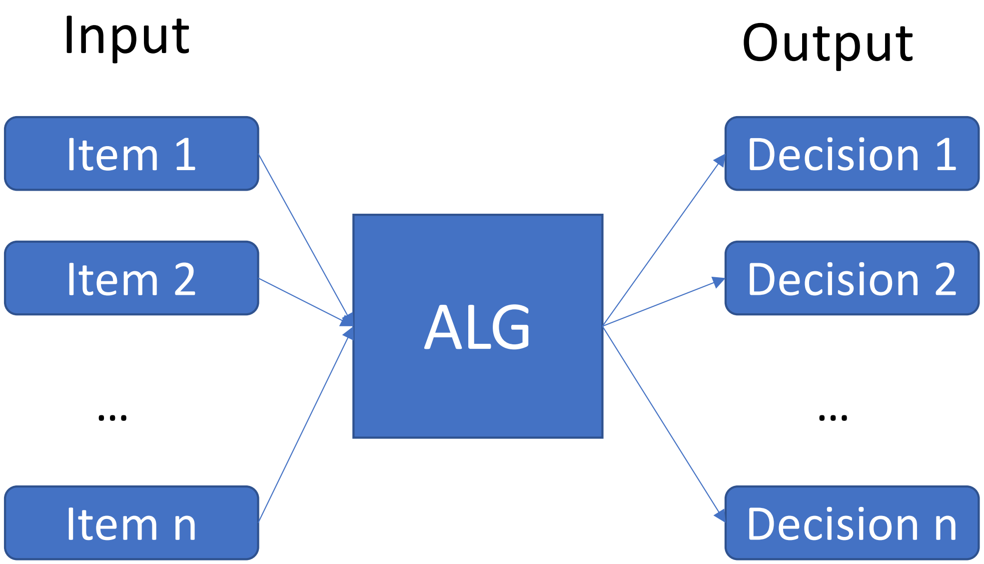
“online” ≠ “internet”

Main question of interest:

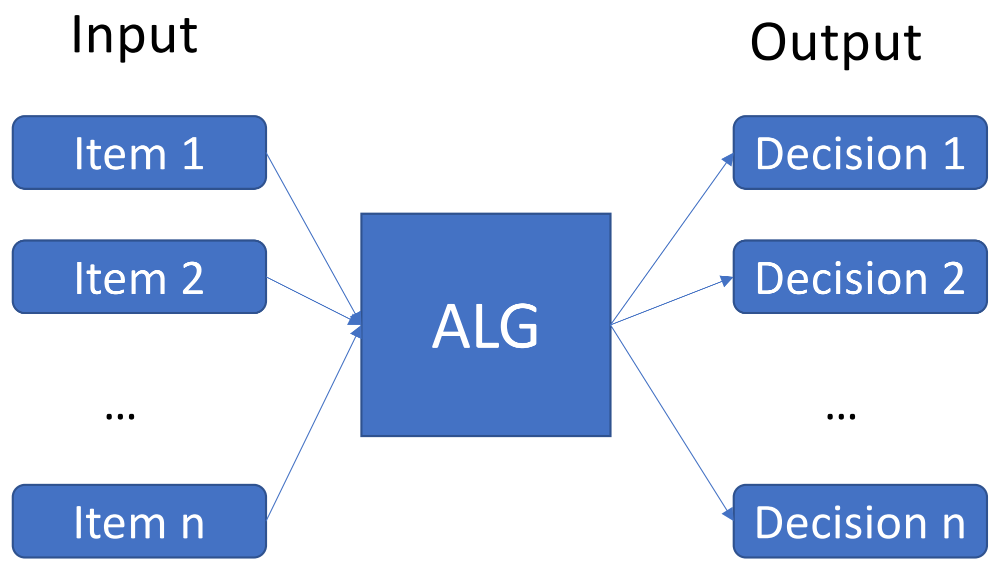
How well can we solve an ***online task*** as compared to an ***offline optimum*** that sees the entire input?

I.e., how powerful can hindsight be?

Offline



Online



Online problem template

Input: sequence of items $I = i_1, i_2, i_3, \ldots, i_n$

Output: sequence of decisions $D = d_1, d_2, d_3, \ldots, d_n$

Goal: optimize objective function $f(I, D)$

Online restrictions:

i_k presented one at a time

d_k is in response to i_k

decisions are *irrevocable*

Ski Rental

Ski rental

Can rent skis for **10$** or buy skis for **100$**

Day	1	2	3	4	5
Weather					
Decision	rent	rent	buy	N/A	N/A
Accumulated Cost	10$	20$	120$	120$	120$

Online problem example: ski rental

Weather 

<u>For this instance (rent 10$, buy 100$)</u>

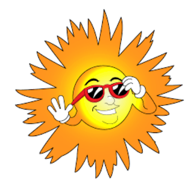
Online solution: 120$ (how much we paid)

Offline optimal solution: 40$ (we would have rented, if we knew the whole input)

Overpaid by a factor 120/40 = 3

Online problem example: ski rental

Can rent skis for **10$** or buy skis for **30$**

Day	1	2	3	4	5
Weather					
Decision	rent	rent	buy	N/A	N/A
Accumulated Cost	10$	20$	50$	50$	50$

Online problem example: ski rental

Weather

For the new instance (rent 10$, buy 30$)

Online solution: 50$ (how much we paid)

Offline optimal solution: 30$ (we would have bought on the first day if we knew the whole input)

Overpaid by a factor 50/30 = 5/3

Ski rental formally

Input: sequence $i_1, i_2, \dots, i_n$

where $i_j = 1$ if weather on day j is good, and

$i_j = 0$ if weather on day j is bad

Output: sequence $d_1, d_2, \dots$

where $d_j = 1$ if you decide to rent on day j, and

$d_j = 0$ if you decide to buy on day j

Goal: minimize total accumulated cost

Side information: b – cost to buy, r – cost to rent

Online algorithm formally

Online algo: produces decisions based on past, but not future, inputs

$$d_k = d_k(i_1, i_2, \dots, i_k)$$

Offline optimum: produces decisions based on entire input

$$\widehat{d_k} = \widehat{d_k}(i_1, i_2, \dots, i_n)$$

Competitive ratio for minimization problem, informally

Accumulated cost of our online algorithm

Competitive ratio

$$\rho = \max_{\{n, i_1, i_2, \ldots, i_n\}} \frac{ALG(i_1, i_2, \ldots, i_n)}{OPT(i_1, i_2, \ldots, i_n)}$$

”How close can we get to hindsight?”

Accumulated cost of an optimal offline algorithm

Competitive ratio: $\rho = \max_{\{n, i_1, i_2, \dots, i_n\}} \frac{ALG(i_1, i_2, \dots, i_n)}{OPT(i_1, i_2, \dots, i_n)}$

Always, $\rho \geq 1$ (Why?)

Worst-case measure:

have to guarantee good performance on ***all*** inputs

Maximum (over instances) might not exist:

replace by supremum

Asymptotic measure:

need to guarantee good performance on ***all large*** inputs

replace max by limit supremum

Back to ski rental

Can guarantee to spend no more than **twice** an offline optimum

The **break-even** online algorithm:

rent until accumulated cost is about to reach cost of buying

buy skis the following day

Formally:

rent for i days where i is smallest such that $(i + 1)r \geq b$

buy skis on day $i + 1$

if weather spoils at any day before then, stop and leave resort

Break-even algorithm: example

Can rent skis for **10$** or buy skis for **30$**

Day	1	2	3	4	5
Weather					
Decision	rent	rent	buy	N/A	N/A
Accumulated Cost	10$	20$	50$	50$	50$

Analysis of break-even algorithm

By scaling, assume cost of renting $r = 1$

For simplicity, assume cost of buying $b \in \mathbb{Z},\ b > 1$

Break-even: rent for $b - 1$ days, buy on day b

Case: weather spoils on day $i < b$

we rent for i days

OPT also rents for i days

Thus, competitive ratio is 1 in this case

Analysis of break-even algorithm

Case: weather spoils on day $i \geq b$

we rent for $b - 1$ days and buy on day b => total cost $2b - 1$

OPT is to buy skis on day 1 => total cost b

In this case, competitive ratio is $\frac{2b-1}{b} = 2 - \frac{1}{b}$

Overall, $\rho = \max\left(1, 2 - \frac{1}{b}\right) = 2 - \frac{1}{b} \to 2$ as $b \to \infty$

Can we do better?

If we use deterministic algorithm, then **NO**

Adversary argument: view execution of an online algorithm as a game between **algorithm (ALG)** and **adversary (ADV)**

Game proceeds in rounds. In each round:

<u>ADV</u>: constructs new input item based on past history and ALG

<u>ALG</u>: responds to the new input item

Adversary argument: ski rental

ADV: tries to maximize the competitive ratio

ALG: tries to minimize the competitive ratio

In deterministic case, ADV knows ***everything*** about ALG

Ski Rental: ADV knows that ALG buys skis on day i

We may assume that $i < \infty$, otherwise ADV can force infinite competitive ratio

Adversary argument: ski rental

ADV knows that ALG buys skis on day i

Strategy for ADV:

declare weather to be bad on day $i + 1$ for the first time

Thus, ADV forces ALG to incur cost $i - 1 + b$

Analysis of adversary strategy

If $i \leq b - 1$

OPT rents for i days having cost i

Competitive ratio: $\frac{i-1+b}{i} = 1 + \frac{b-1}{i} \geq 2$

If $i \geq b$

OPT buys skis on day 1 incurring cost b

Competitive ratio: $\frac{i-1+b}{b} \geq \frac{2b-1}{b} = 2 - \frac{1}{b}$

Analysis of adversary strategy

Since ALG tries to minimize competitive ratio, and

ADV has to work for all possible ALG

The competitive ratio that ADV can force is

$$\min\left(2, 2 - \frac{1}{b}\right) = 2 - \frac{1}{b}$$

Putting it together

(1) We found a particular algorithm: Break-Even with competitive ratio

$$\leq 2 - \frac{1}{b}$$

(2) We showed that any algorithm for Ski Rental has competitive ratio

$$\geq 2 - \frac{1}{b}$$

Therefore

Break-Even is an optimal *deterministic* online algorithm for Ski Rental

More generally

Results of type (1), i.e., find a good algorithm, are called

Upper bounds or

Positive results

Results of type (2), i.e., prove no algorithm can do well, are called

Lower bounds or

Negative results

Upper/lower bound terminology becomes ambiguous when dealing with maximization problems, so prefer to use positive/negative

Notes on positive/negative results

When the two results coincide, we get a **tight bound**

Positive/negative results can refer to algorithms AND problems

	Positive result	Negative result
Algorithm	Prove that **this particular algorithm** works well on **all instances**	Find **an instance** on which **this particular algorithm** works badly
Problem	Find **some algorithm** that works well **on all instances** of the problem	Prove that **all algorithms** work badly on **some instance** of this problem

Notes on positive/negative results

Negative result for a problem implies the same negative result for each algorithm

Positive result for an algorithm implies the same positive results for the corresponding problem

We proved the tight result of $2 - \frac{1}{b}$ for the Ski Rental **problem**

We also proved the tight result of $2 - \frac{1}{b}$ for Break-Even **algorithm**

We will later see problems for which positive and negative results are not tight

Why care about online algorithms?

Sometimes irrevocable decisions are forced upon us:

- scheduling – scheduling jobs at a data center
- resource allocation – matching doctors to patients
- packing – fulfilling warehouse orders
- caching – evicting memory pages from a cache
- online advertising – displaying ad banner to a user
- online learning – each new data sample is processed and incorporated into a learning algorithm
- big data – processing large volumes of data (streams)

Why care about online algorithms?

Online algorithms are **also** useful in applications where irrevocable decisions are not forced

Online algorithms can be interpreted as offline algorithms, often with properties:

- efficient
- conceptually simple
- achieving non-trivial approximation ratios for NP-hard problems and problems in P
- can be used to model greedy algorithms

Typical process of studying online problems

Step 1: define the problem precisely

define input items

define decisions

define an objective

Step 2: prove a tight bound on deterministic algorithms (worst-case)

Step 3: prove a tight bound on randomized algorithms (worst-case)

Step 4: prove a tight advice-competitive ratio tradeoff (worst-case)

Step 5: redo previous steps under other models: stochastic, streaming, etc.

Short history of online algorithms

1966 – Ron Graham gave analysis of an online greedy alg for Makespan

1973 – Johnson's PhD thesis on online algorithms for bin packing

1985 – Sleator and Tarjan analyze online algorithms for paging

this paper argued in favor of worst-case analysis

1988 – Karlin et al. introduced the term competitive analysis

… - A lot of work on k-server, paging, makespan, etc.

1998 – El-Yaniv and Borodin book on online algorithms

2009 – Feldman et al. reintroduce stochastic input model for bipartite matching

Back to ski rental: Can we do better than 2-1/b?

We have proved:

NO (if we use a deterministic algorithm)

Can we do better if we allow randomized decisions?

Solutions constructed by such algorithms are random variables

How should we measure performance of such algorithms?

What's a competitive ratio for randomized algorithms?

Ski rental revisited: randomized alg

Competitive ratio for randomized algorithm:

expected cost of randomized algorithm / OPT

Cost to rent: 1\$

Cost to buy: b\$

We know that to break even, we need to buy on day b

To fool the adversary, buy earlier on day i with probability p_i

How to set the p_i to minimize expected cost?

Randomized Ski Rental alg cost

If the adversary spoils weather on day $g + 1$ where $g < b$, then the expected cost of our solution:

$$\sum_{i=0}^{g-1} (i + b)p_i + \sum_{i=g}^{b-1} gp_i$$

If the adversary spoils weather on day $g + 1$ where $g \geq b$, then the expected cost of our solution:

$$\sum_{i=0}^{b-1} ip_i + b$$

Randomized Ski Rental OPT cost

If the adversary spoils weather on day $g+1$ where $g < b$, then OPT:

$$g$$

If the adversary spoils weather on day $g+1$ where $g \geq b$, then OPT:

$$b$$

Randomized Ski Rental analysis

Let c denote the competitive ratio of our algorithm

$$
\begin{array}{lll}
\text{minimize} & c & \\
\text{subject to} & \displaystyle\sum_{i=0}^{g-1}(i+b)p_i + \sum_{i=g}^{b-1} g p_i \leq cg & \text{for } g \in [b-1] \\
& \displaystyle\sum_{i=0}^{b-1} i p_i + b \leq cb & \\
& p_0 + p_1 + \cdots + p_{b-1} = 1 &
\end{array}
$$

Randomized Ski Rental analysis

Solving the linear program we get

$$p_i = \frac{c}{b}\left(1 - \frac{1}{b}\right)^{b-i-1}$$

Euler's constant

and

$$c = \frac{1}{1 - \left(1 - \frac{1}{b}\right)^{b}} \to \frac{e}{e-1} \approx 1.5819 \ldots$$

Ski Rental wrap-up

Deterministic case: competitive ratio ≈ 2

Randomized case: competitive ratio $\approx 1.5819 \ldots$

Randomness definitely helps for Ski Rental

Randomness does not always help

Line Search

Line Search Problem

Also known as **Cow Path Problem** or **Robot Exploration in 1D**

Setting:

A robot starts at the origin of the x-axis

The robot can move at unit speed in either direction and change direction of travel instantaneously

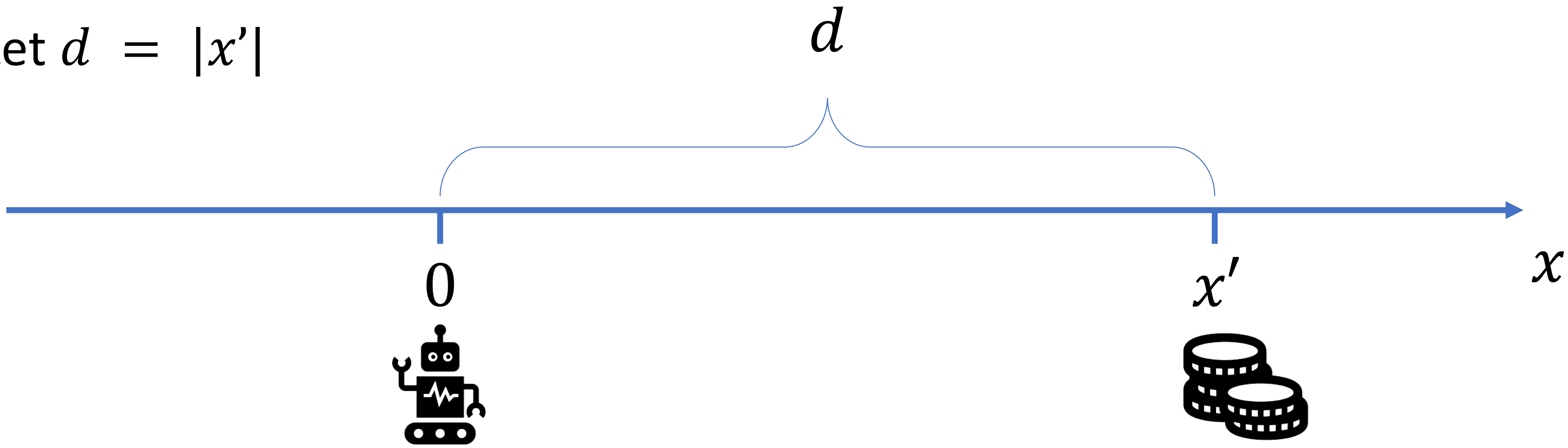
Treasure is located somewhere on the axis at x'

The goal is to find the treasure as soon as possible

The robot can learn if there is treasure at location x' by visiting x'

Line Search Problem

Let $d = |x'|$

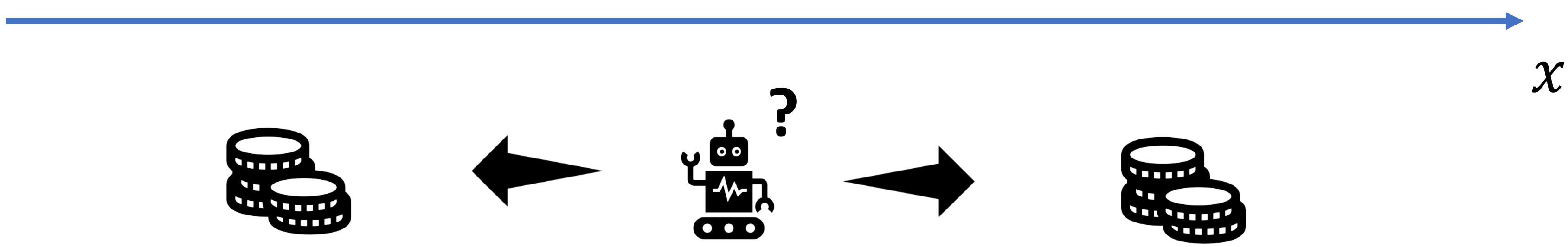


If robot knew $\mathrm{sgn}(x')$, it could find treasure by travelling to it directly

Therefore $OPT = d$

Line Search Problem

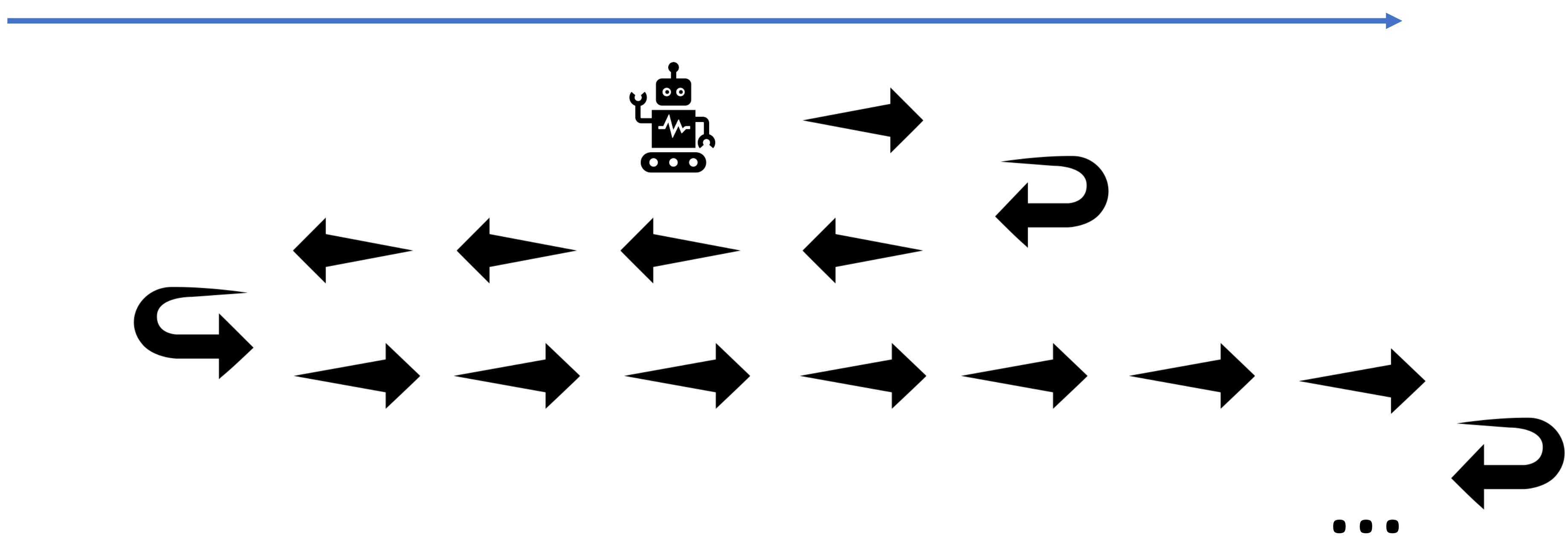
However, the robot does not know $\mathrm{sgn}(x')$



It needs a strategy to potentially explore the **entire** x-axis

Line Search Problem

Natural strategy: zig-zag!



Line Search Problem

Zig-zag doubling strategy:

Pick a direction +1 and distance 1

Repeat until treasure is found:

Travel in the chosen direction for the given distance

Return to the origin

Flip the direction and **double** the distance

Note: if you've seen vector data structure implementation, then doubling strategy will look familiar

Line Search Problem

Analysis of the doubling strategy

Recall, $OPT = d$

How much more does the robot travel?

Phase i: robot travels in direction $(-1)^i$ distance 2^i, then returns to the origin

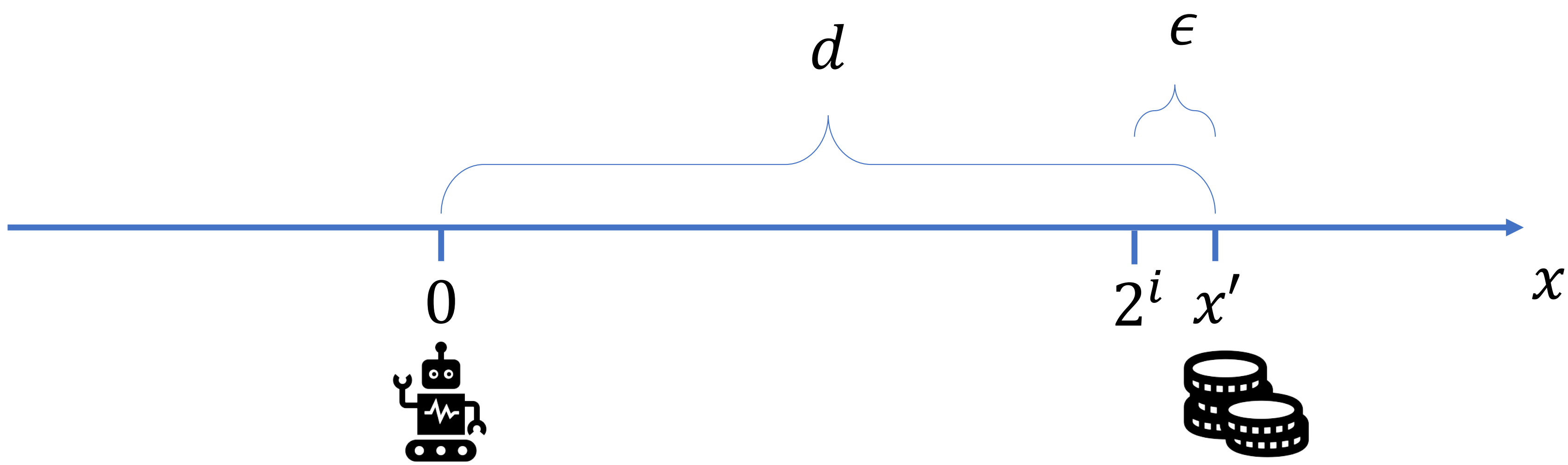
If treasure is not found in phase i, total distance travelled in phase i is

$$2 \cdot 2^i = 2^{i+1}$$

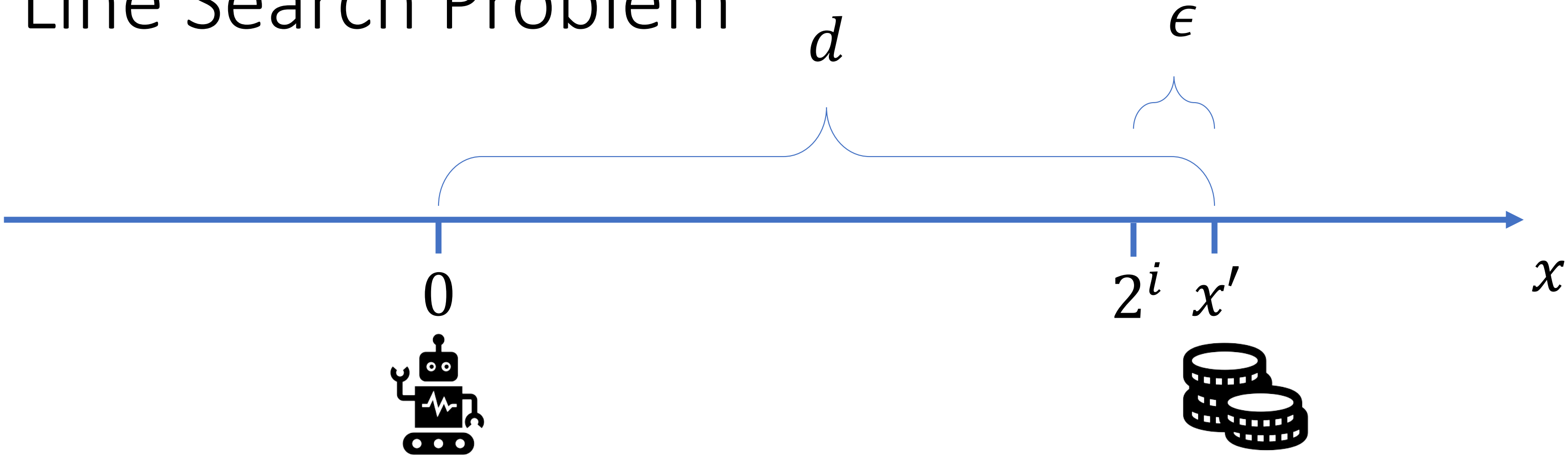
Line Search Problem

Worst case happens when treasure is just outside of the radius covered in some phase

We have $d = 2^i + \epsilon$ and $\mathrm{sgn}(x') = (-1)^i$



Line Search Problem



Then the robot returns back to the origin

Doubles the distance, and travels in the wrong direction

Returns to the origin and discovers treasure by travelling in the right direction for distance d

Line Search Problem

The total distance travelled by the robot is

$$2\Big(\underbrace{1 + 2 + 4 + \cdots 2^i}_{\substack{\text{all work until} \\ \text{barely missing treasure}}} + \underbrace{2^{i+1}}_{\substack{\text{wrong} \\ \text{direction} \\ \text{one last time}}}\Big) \underbrace{+ d}_{\substack{\text{Finding} \\ \text{treasure}}} < 2 \cdot 2^{i+2} + d < 8d + d = 9d$$

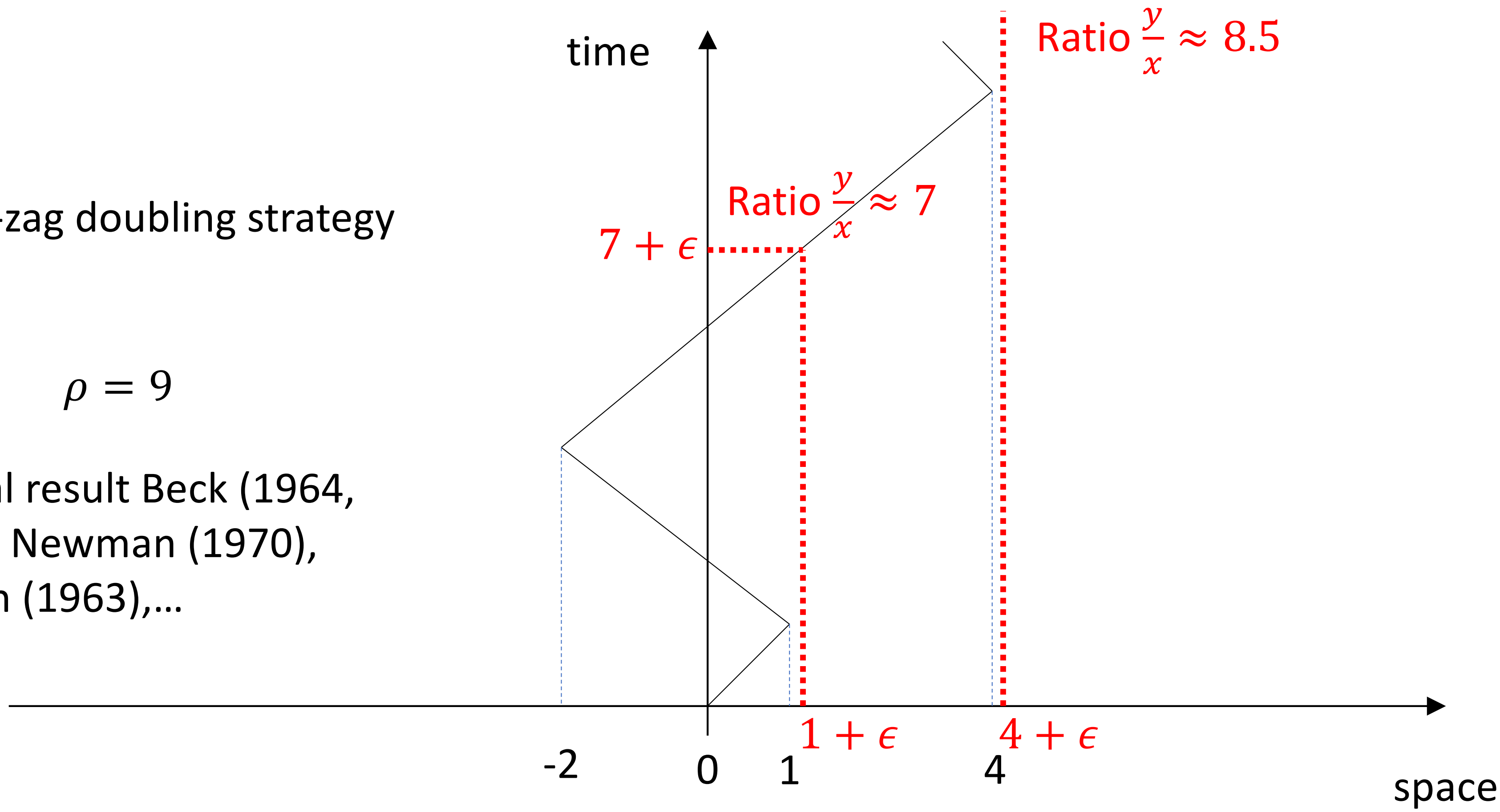
The last inequality is since $d = 2^i + \epsilon > 2^i$

Useful Tool: Time-space Diagrams

Zig-zag doubling strategy

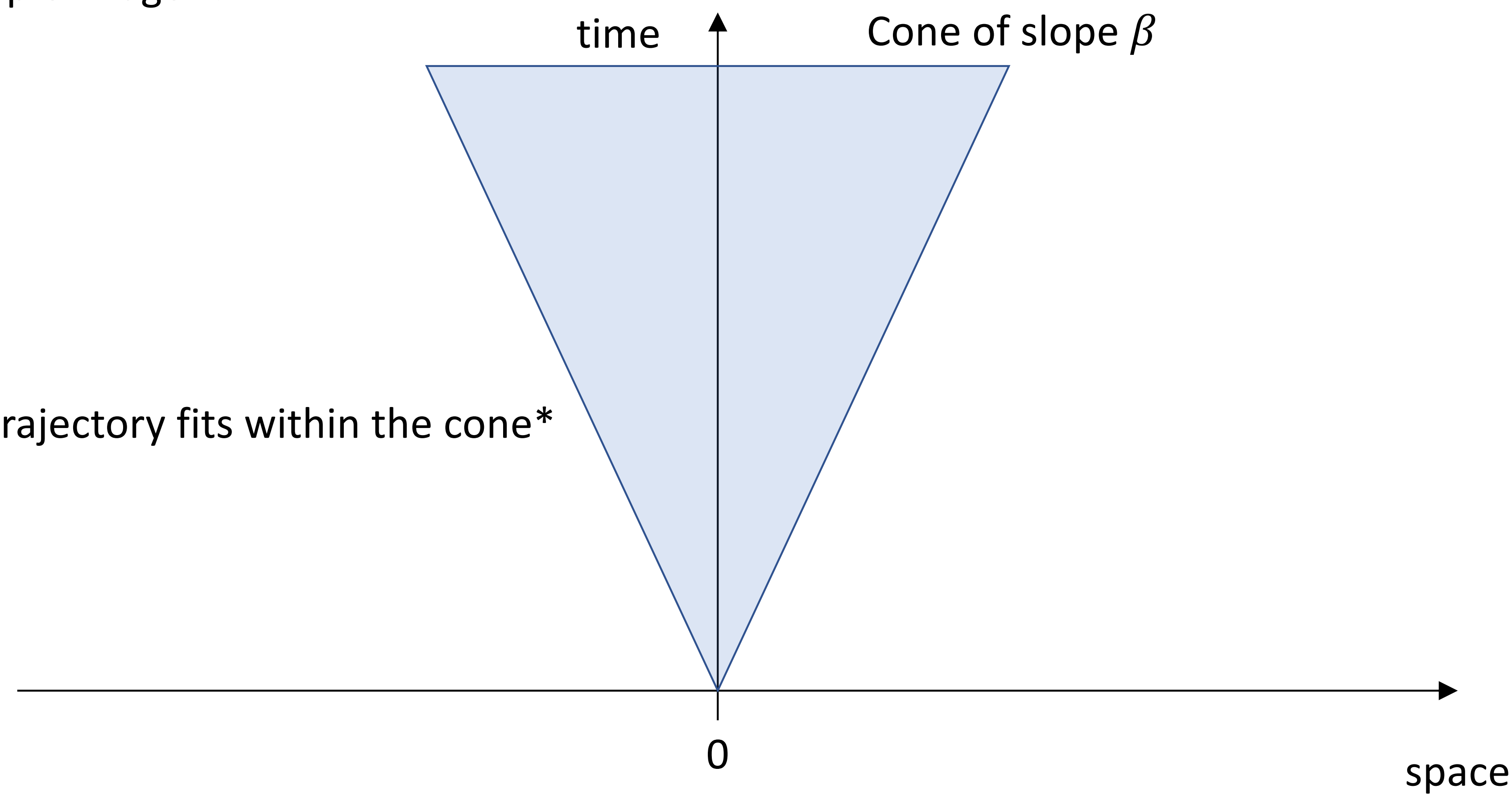
$\rho = 9$

Classical result Beck (1964, 1965) + Newman (1970), Bellman (1963),…



Lower Bound Technique: Cones in Time-Space Diagrams

Example: 1 agent



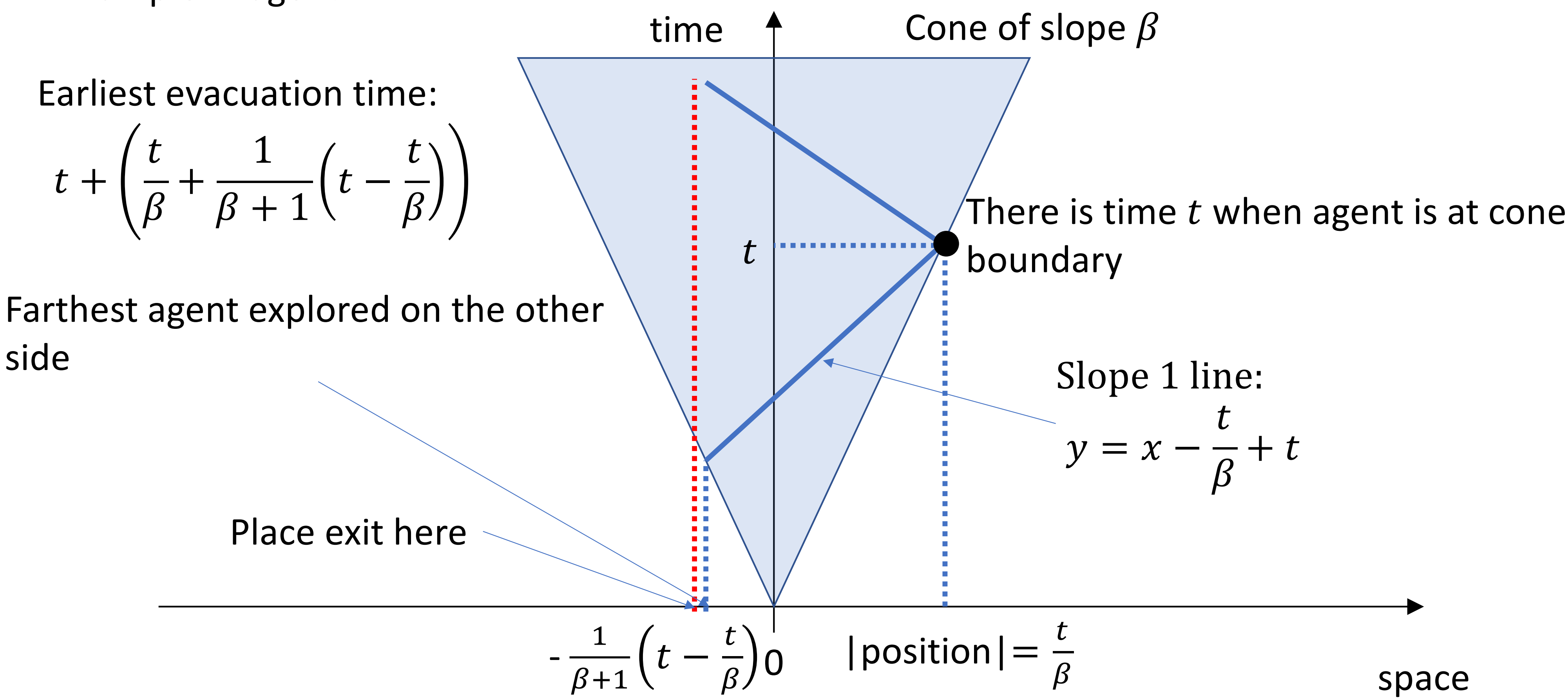
ALG trajectory fits within the cone*

Lower Bound Technique: Cones in Time-Space Diagrams

Example: 1 agent

Earliest evacuation time:

$$t + \left(\frac{t}{\beta} + \frac{1}{\beta+1}\left(t - \frac{t}{\beta}\right)\right)$$



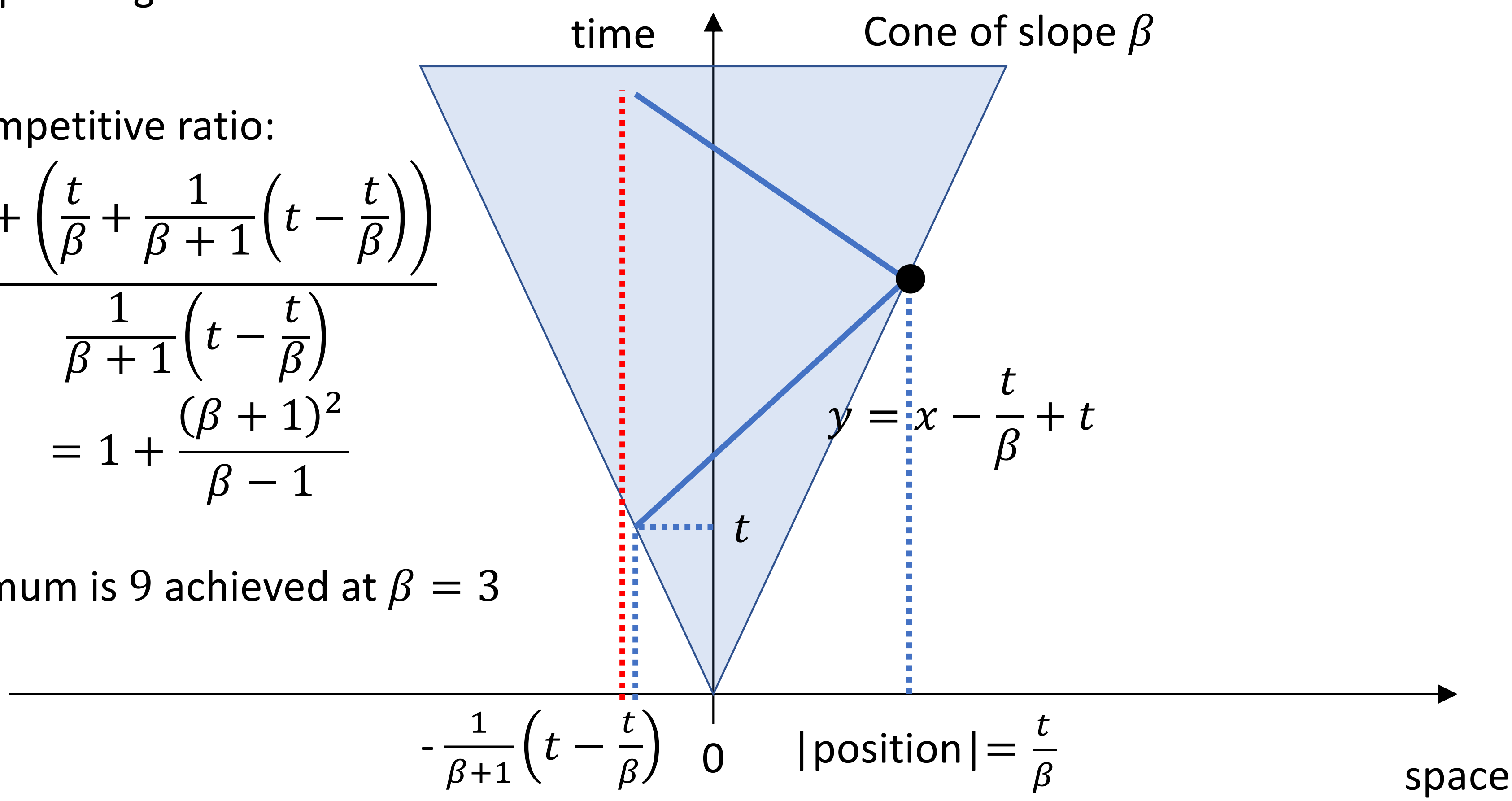
Lower Bound Technique: Cones in Time-Space Diagrams

Example: 1 agent

Competitive ratio:

$$\frac{t + \left(\frac{t}{\beta} + \frac{1}{\beta+1}\left(t - \frac{t}{\beta}\right)\right)}{\frac{1}{\beta+1}\left(t - \frac{t}{\beta}\right)} = 1 + \frac{(\beta+1)^2}{\beta - 1}$$

Minimum is 9 achieved at $\beta = 3$



Line Search Problem

Conclusion:

Zig-zag doubling strategy is 9-competitive

Important note:

Typical online problems have well-defined input presented in "online" fashion.

In line search problem, input is presented in response to algorithm queries: "is there a treasure at location x that I am visiting?"