The k-server problem

- Problem definition
- Examples
- An offline algorithm
- A lower bound and the k-server conjecture
- Several online algorithms

Problem definition

- $k > 1$ servers
- M is a metric space with metric d
- Servers are located at points of M
- Request sequence σ consists of points of M
- A request is served by moving a server there
- Cost is total distance traveled by servers
- Goal: minimize the total cost

Examples (1)

- Paging
  - Uniform space (all distances are 1)
  - Servers are slots in the cache
  - Fault (moving a server) costs 1
- Weighted paging
  - As above, but cost of moving a page into the cache depends on the page
  - E.g. a distributed file system
  - Asymmetric k-server problem
  - This space is not metric!

Examples (2)

- k-headed disk
  - A disk with multiple read/write heads
  - Each head can access all locations on the disk
  - Which head should be moved for a particular request?
  - Possible performance measure: total distance moved by all heads

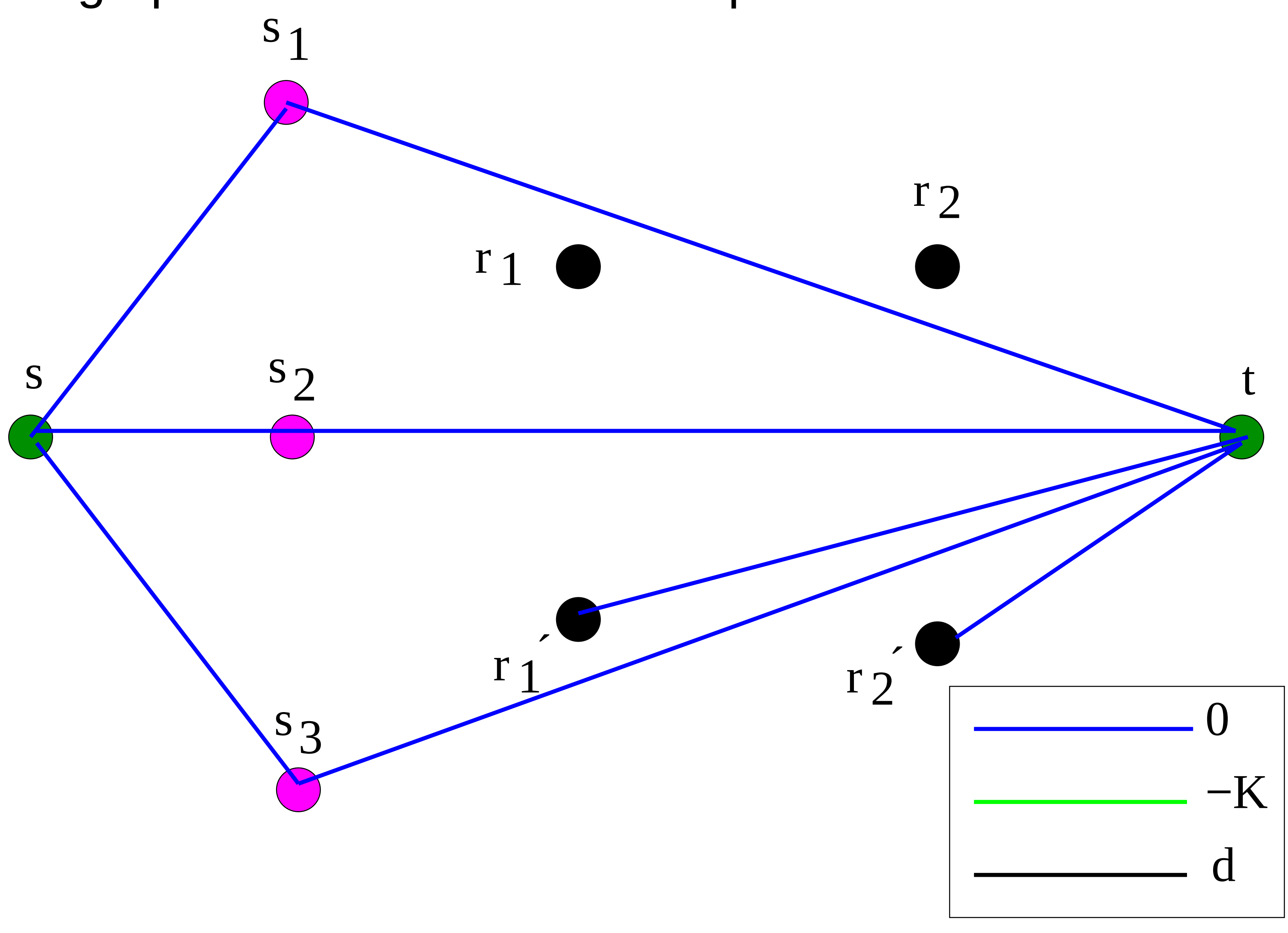
The offline problem

- Can be solved using dynamic programming
- This is not the most efficient solution
- Better: reduce to mincost / maxflow problem
- We will construct a graph with maximum flow k
- Minimum cost for this flow will correspond to k-server solution

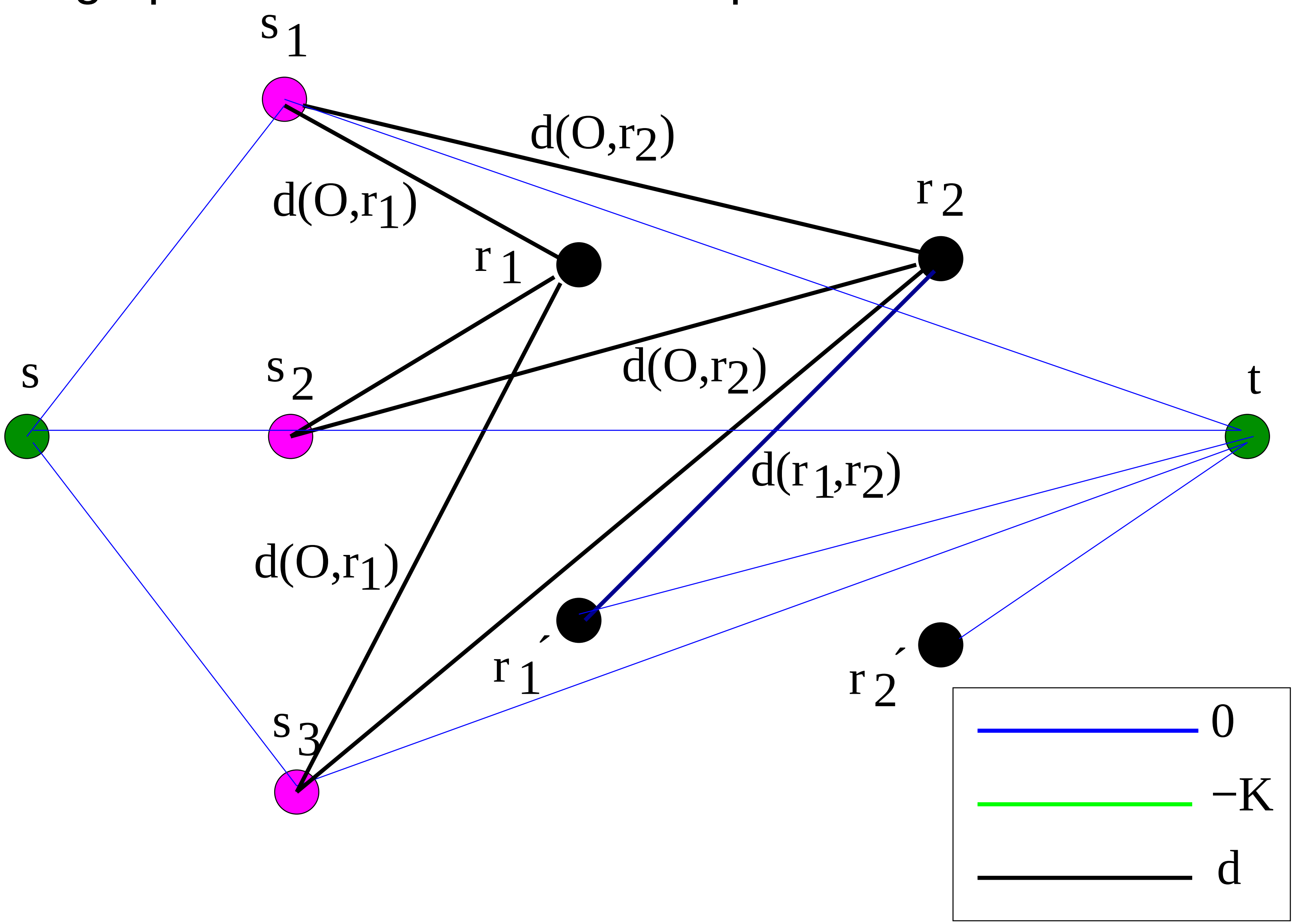
Construction of the graph

- Servers are $s_1, \ldots, s_k$
- Request sequence is $r_1, \ldots, r_n$
- Nodes are $s, t, s_1, \ldots, s_k, r_1, r'_1, \ldots, r_n, r'_n$
- All arcs have capacity 1
- Costs depend on arcs
- We assume all servers start in the same point, the origin O

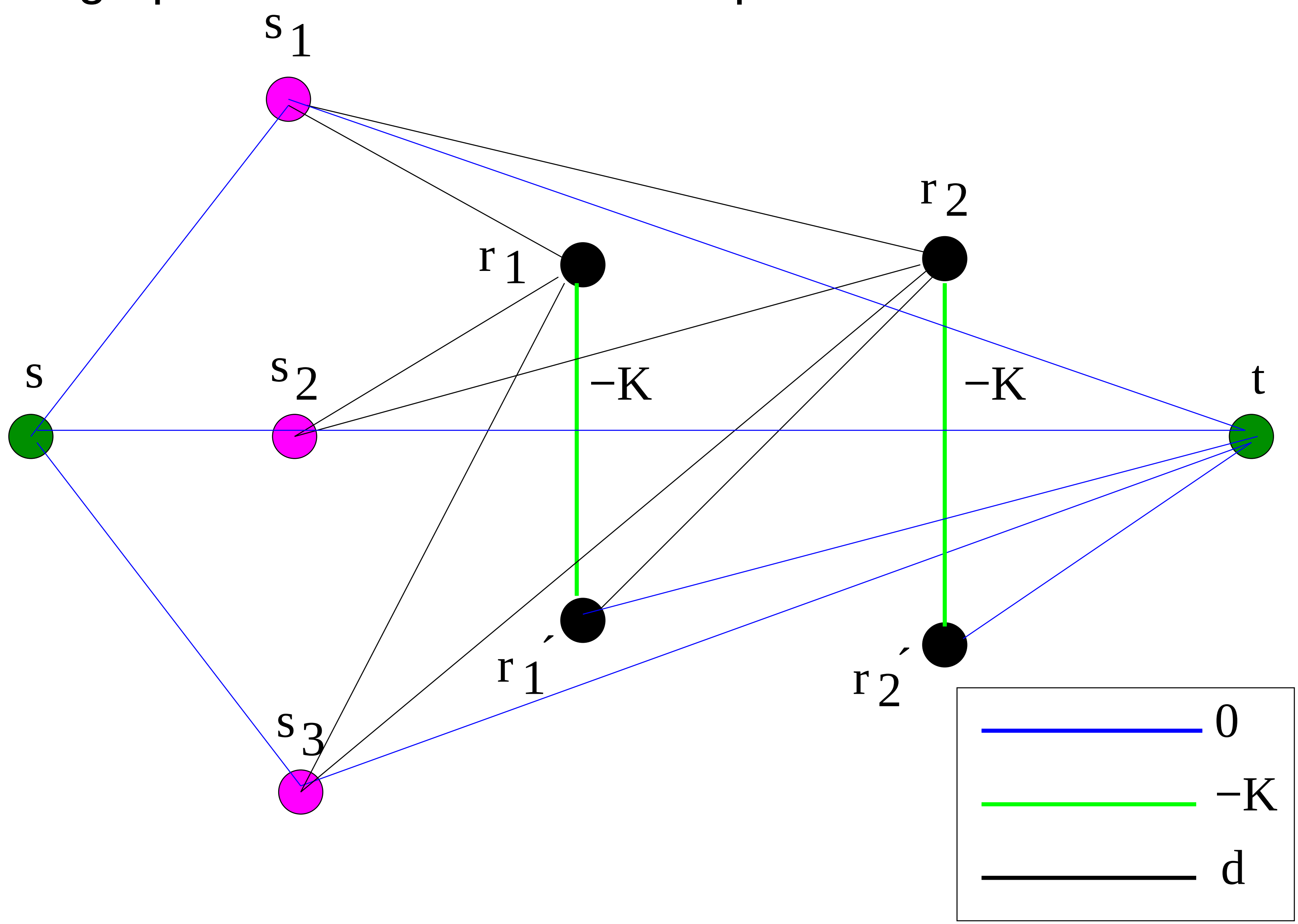
The graph for 3 servers and 2 requests



The graph for 3 servers and 2 requests



The graph for 3 servers and 2 requests



The maximum flow

- Since all capacities are one, maxflow = k (consider the servers)
- Since all capacities are integer, we can find an integral min-cost flow of value k in time $O(kn^2)$
- This flow basically consists of k disjoint paths
- All edges (r_i, r_i') will be used in a min-cost solution
- Each path corresponds to a server visiting the requests on its path
- This gives an optimal schedule for the servers

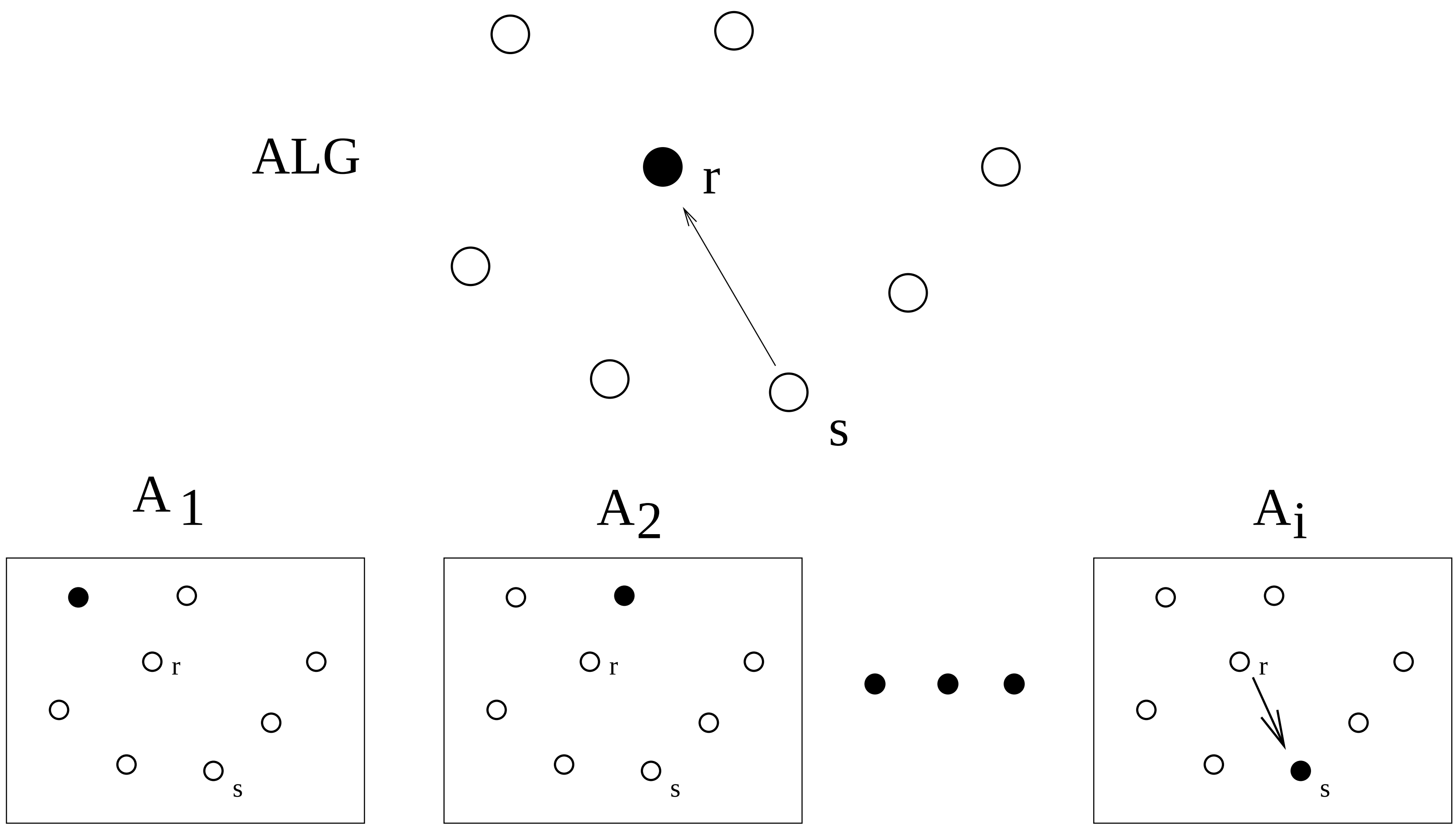
Lower bound

- We show a lower bound of k for an arbitrary online algorithm ALG
- We use an arbitrary space with $k+1$ points
- We compare to k different other algorithms $A_1, \ldots, A_k$ that the adversary controls
- An algorithm is determined by the uncovered point (hole)
- Invariant: holes of ALG and k other algorithms cover the space

Cruel request sequence

- Before first request, each A_i moves one server to hole of ALG to ensure invariant holds
- At each step, we request the hole of ALG
- Denote request by r, then ALG moves to r from, say, s
- Other algorithms: all have a server at r, exactly one (say A_i) has no server at s
- Now, A_i moves from s to r
- The other $k-1$ algorithms do nothing

Cruel request sequence



Relative costs

- In each step j, ALG pays some cost c_j
- Only one of the other algorithms $A_1, \ldots, A_k$ pays c_j
- Summing over all algorithms and the entire sequence, we get

$$\sum_{i=1}^{k} A_i(\sigma) = \text{ALG}(\sigma) + \sum_{i=1}^{k} d(x_i, x_0)$$

- There must be one algorithm which has a cost of at most ALG$(\sigma)/k$ (plus an additive constant)
- This proves the lower bound

The k-server conjecture

Any metric space allows for a deterministic, k-competitive algorithm.

- The work function algorithm is $(2k-1)$-competitive in any metric space
- For certain metric spaces, k-competitive algorithms are known

Fundamental open question in online algorithms

The k-server conjecture

Any metric space allows for a deterministic, k-competitive algorithm.

Note: other generalizations of paging results fail!

- There is no $k/(k-h+1)$-competitive k-server algorithm for the (h,k)-server problem
- Not every metric space allows a randomized H_k-competitive algorithm

The greedy algorithm

Definition: serve each request by the closest server

The greedy algorithm

Definition: serve each request by the closest server

This algorithm is not competitive

Request sequence: $c, b, a, b, a, b, a, \ldots$

Greedy leaves one server at c forever

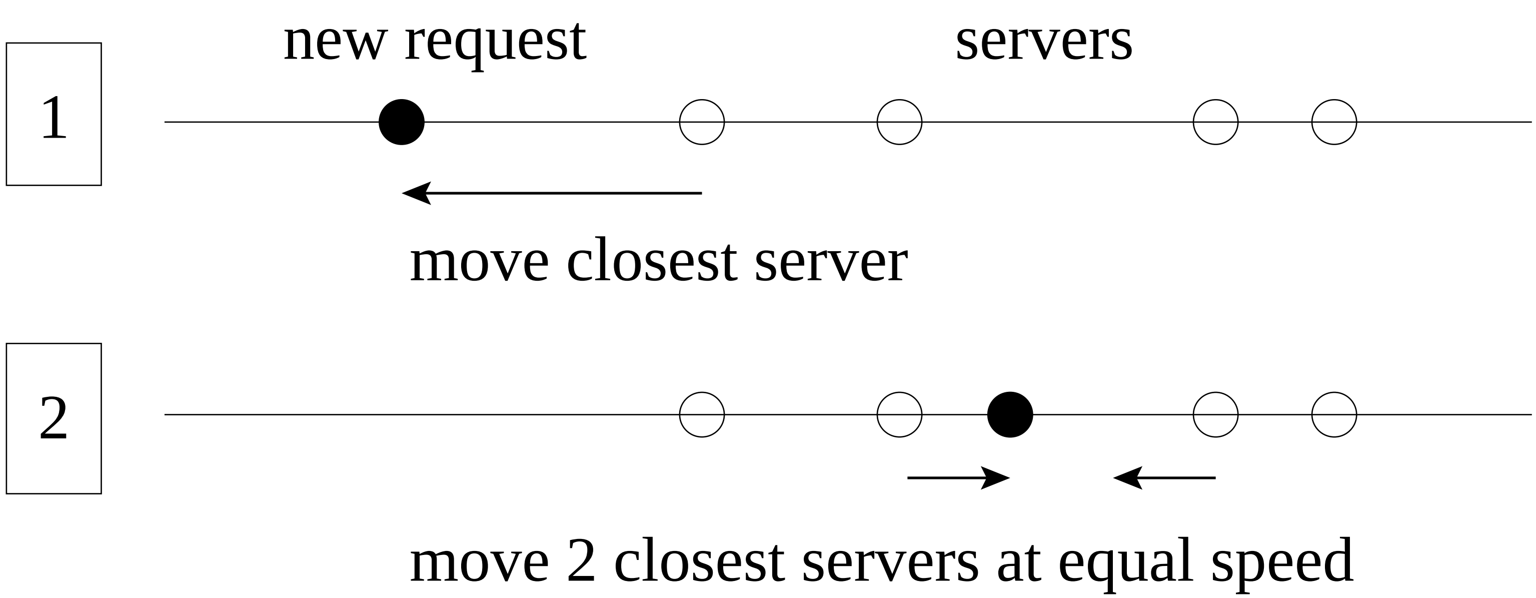
The other one moves between a and b

OPT moves servers to a and b and has constant cost

k servers on the line

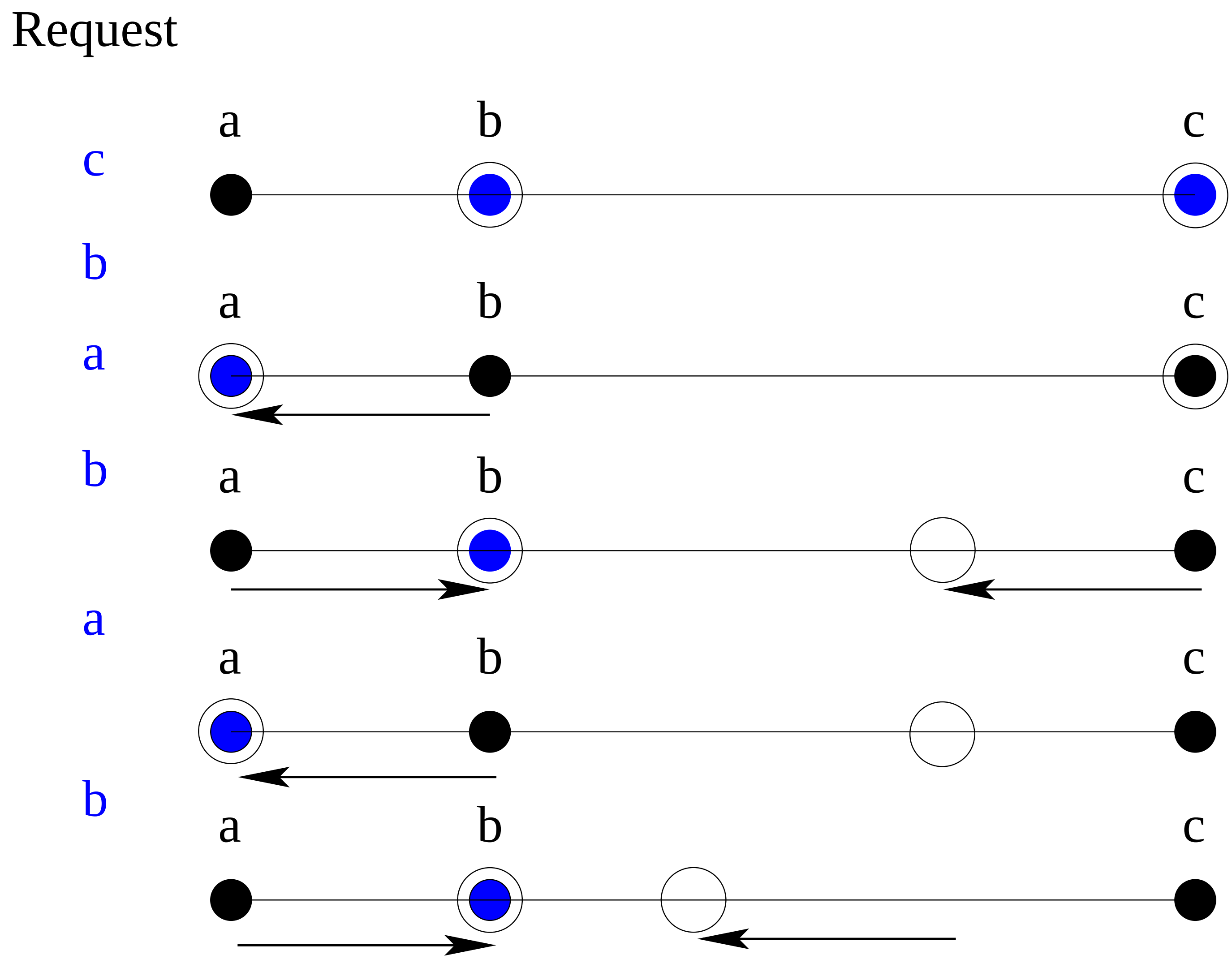
Algorithm Double Cover

Two cases: request is between two servers, or at one side



If two servers are at same point, choose one to move

Double Cover on first example



Eventually, servers are at a and b and stop moving

Analysis of Double Cover

- We show that DC is k-competitive
- We use a potential function as in the List Update problem
- Let min be the cost of the minimum cost matching between the servers of DC and OPT
- Let s_i be the ith server of DC
- Define $sum = \sum_{i<j} d(s_i, s_j)$
- Potential function:

$$\Phi = k \cdot min + sum$$

The potential function

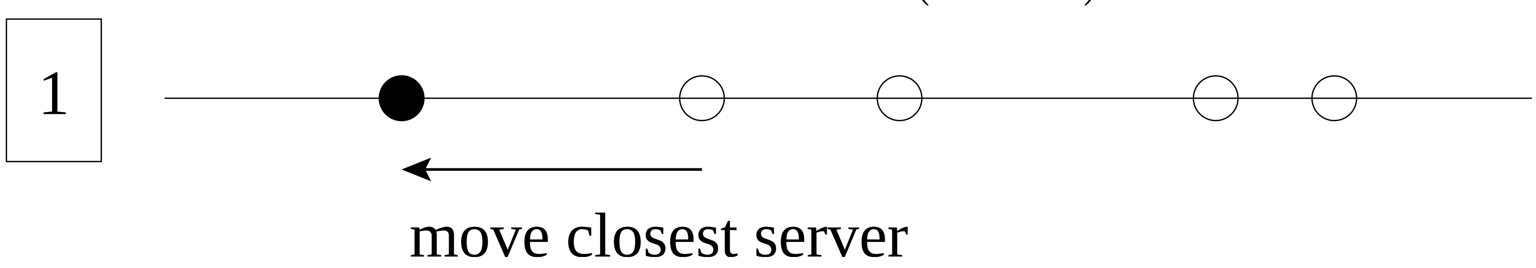
- $\Phi = k \cdot min + sum$, so it is bounded from below
- We show:
  1. If OPT moves a distance d, Φ increases by at most kd
  2. If DC moves a distance d, Φ decreases by at least d
- Since $\Phi \geq 0$ at all times, this shows DC is k-competitive
- Property 1 holds since
  - sum is unchanged by move of adversary
  - min cannot increase by more than d

Change of Φ when DC moves

DC moves only 1 server over a distance d:

- it moves away from all other servers
- sum increases by $(k-1)d$
- there exists a minimum cost matching where this server is matched to this request (one OPT server is there)
- Therefore, min decreases by at least d

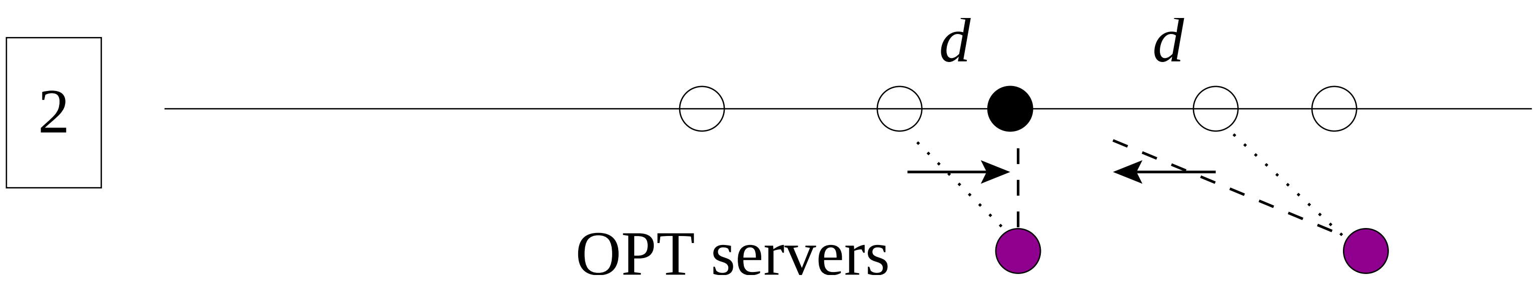
Overall decrease of Φ is at least $k \cdot d - (k-1)d = d$



Change of Φ when DC moves

DC moves two servers, s_1 and s_2, by a distance d:

- one of them is matched to the request in some minimum cost matching
- min is decreased by at least d by this move
- other server moves at most d away from its match
- min does not increase overall



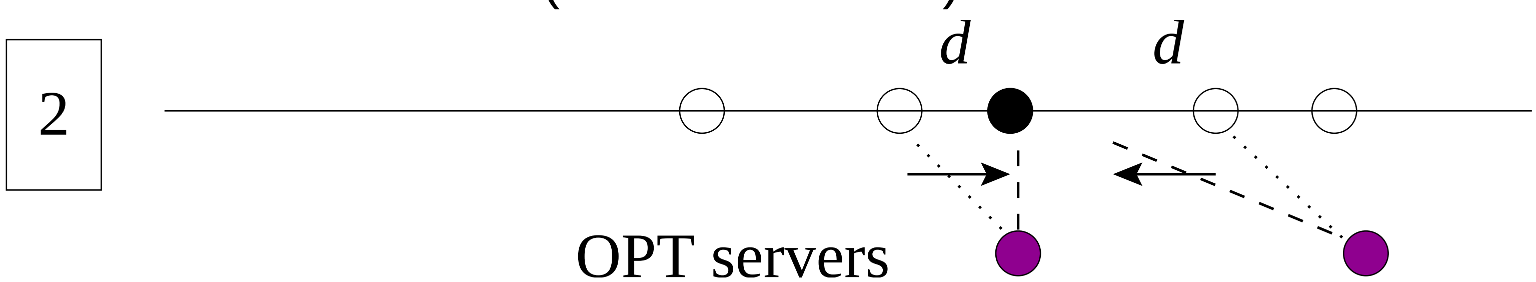
Change of Φ when DC moves

DC moves two servers, s_1 and s_2, by a distance d:

what is the change of sum?

- total distance from s_1 and s_2 to any other online server is unchanged
- distance between s_1 and s_2 decreases by $2d$

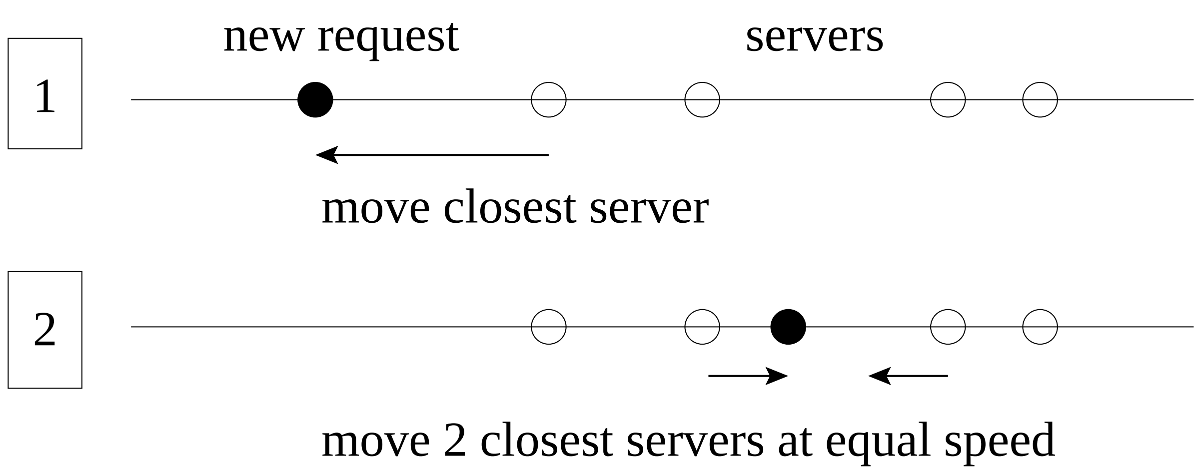
Overall decrease of sum (and therefore Φ) is at least $2d$



k servers on trees

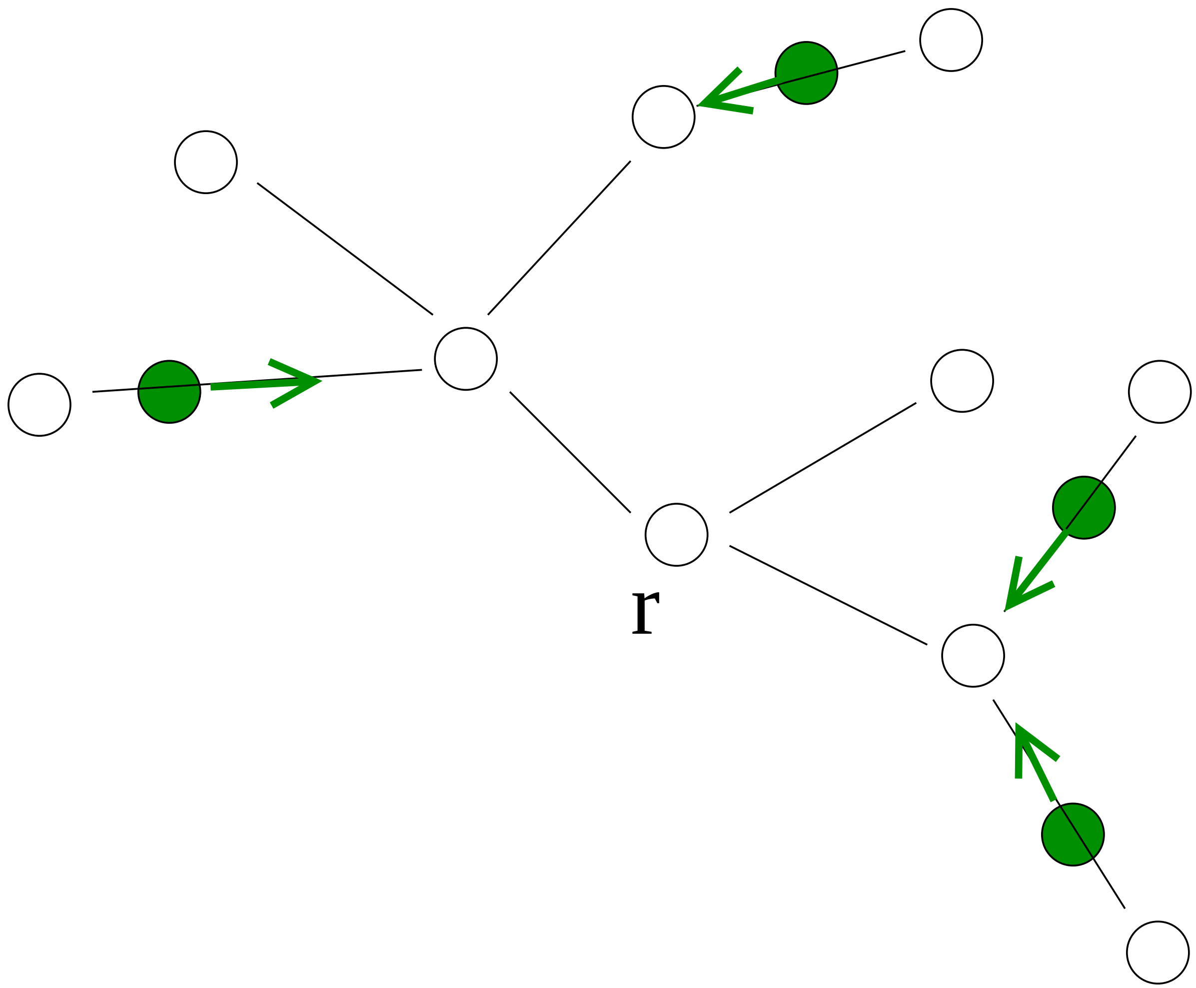
- Algorithm Double Cover can be extended for trees
- It still has a competitive ratio of k
- Definition of DC-TREE:

  At all times, all the servers neighboring the request are moving in a constant speed towards the request
- On a line, DC-TREE is identical to DC



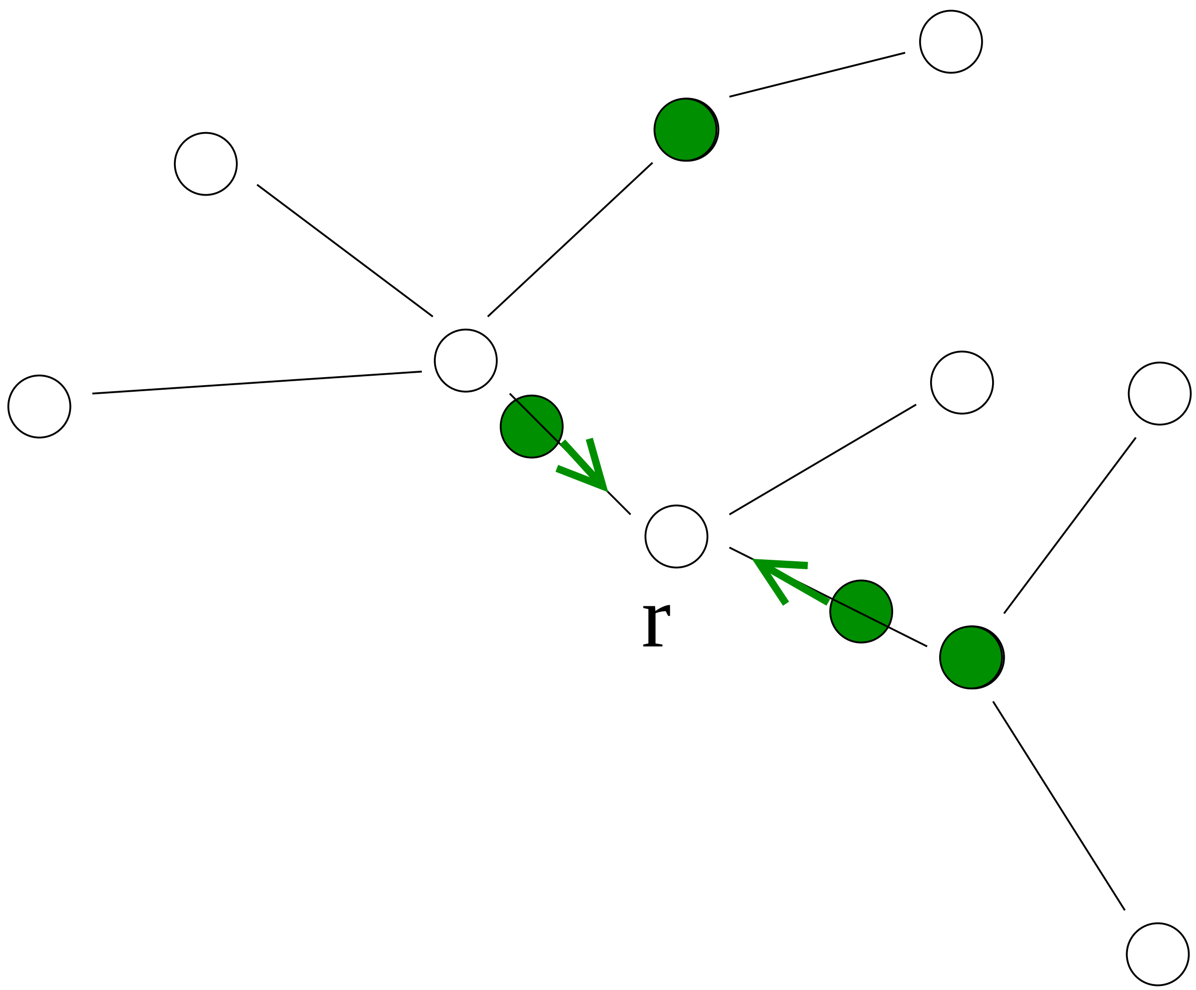
DC-TREE

- DC-TREE may move all k servers simultaneously



DC-TREE

- While moving towards a request, some servers may get “cut off” and stop moving



Upper bound for DC-TREE

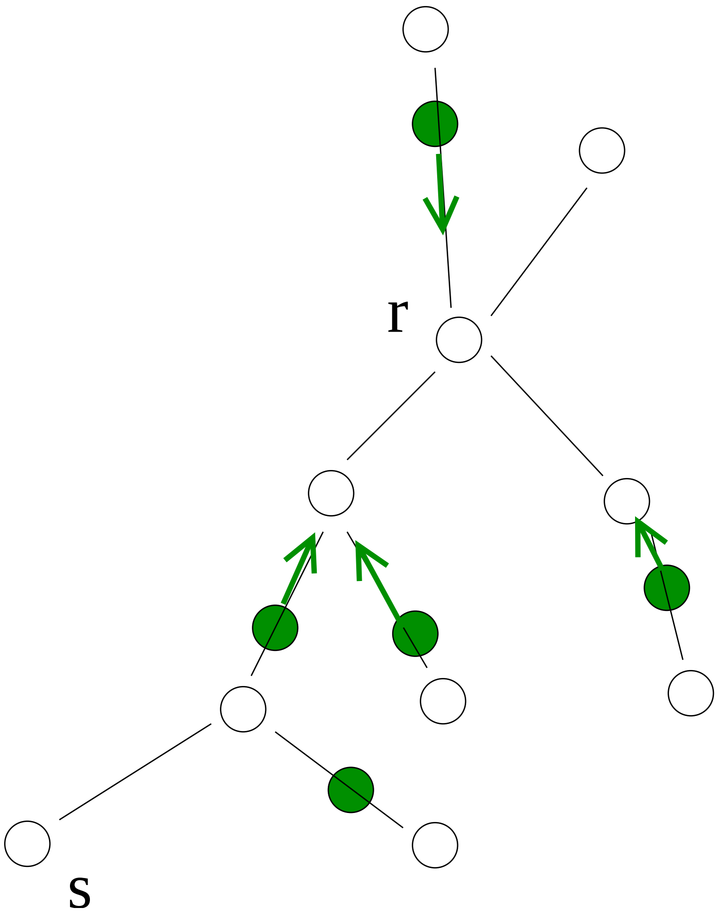
- We use the same potential function $\Phi = k \cdot min + sum$
- A move by OPT still increases Φ by at most kd
- We break the action of DC-TREE to serve a single request into phases
- In each phase, the subset of servers that moves is fixed
- Need to show: Φ decreases at least by total distance traveled by DC-TREE
- We consider separately the change of min and sum in a phase

The change of min

- Denote the number of neighbours in a phase by m
- One of these is matched to the request in a minimum cost matching
- Moving that server by d decreases min by d
- Moving the $m-1$ other servers by d increases min by at most $(m-1)d$
- min increases by at most $(m-2)d$

The change of sum: non-moving servers

- Consider a server s which is not moving (no neighbour of the request)
- Exactly one server is moving away from s, $m-1$ others are moving towards s
- Change in sum caused by this server is $(m-2)d$



The change of sum: non-moving servers

- We need to sum over the $k-m$ non-moving servers
- sum decreases by

$$(k-m)(m-2)d$$

The change of sum: moving servers

- Each pair of moving servers gets closer together by $2d$
- Summing over $m(m-1)/2$ pairs, this gives a decrease in sum of

$$dm(m-1)$$

The change of Φ

- min increases by at most $(m-2)d$
- Due to non-moving servers, sum decreases by

$$(k-m)(m-2)d$$

- Due to moving servers, sum decreases by

$$dm(m-1)$$

- In total, $\Phi = k \cdot min + sum$ decreases by at least

$$(k-m)(m-2)d + dm(m-1) - k(m-2)d = dm$$

- This is exactly the total distance that DC-TREE moves

Application: arbitrary graph G

- take a spanning tree T, apply DC-TREE on it
- Let n be the number of nodes of G
- An edge of length d in G has a detour on T of length at most $(n-1)d$
- Thus

$$\textsc{opt-tree}(\sigma) \leq (n-1)\text{OPT}(\sigma)$$

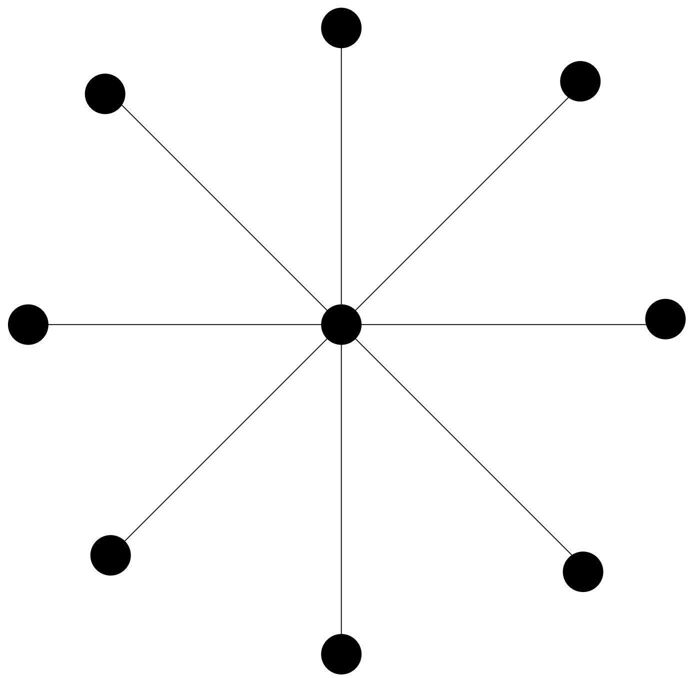
- Since DC-TREE is k-competitive on trees, we have

$$\textsc{dc-tree}(\sigma) \leq k \cdot \textsc{opt-tree}(\sigma)$$

- We have a $(n-1)k$-competitive algorithm

Application: paging

- Suppose there are N slow memory pages
- Create a star graph with N edges of length $1/2$
- The central node is labeled v
- The other nodes are the “page nodes”



DC-TREE for paging (1)

- Servers start on k page nodes
- On first request, all servers move to v
- **One** server continues to requested page
- On subsequent requests, other servers move away from v
- Once all servers have left, next request causes all servers to return to v
- One server continues to request, etc.

DC-TREE for paging (2)

- This algorithm is equivalent to FLUSH-WHEN-FULL
- Moving to v is equivalent to clearing the cache
- This gives an alternative proof that FWF is k-competitive

Euclidean spaces

- DC is k-competitive for the line
- This is the one-dimensional Euclidean space
- Can we extend this to the higher dimensions?
- Even for the plane, no efficient algorithm with good competitive ratio is known
- Efficient = computational cost per request does not depend on length of input sequence

The Work Function Algorithm

- Tries to mimic OPT
- Keeps track of optimal offline cost so far
- Tries to have a configuration similar to OPT
- Is $(2k-1)$-competitive for any metric space

Attempt 1

- For each request, calculate an optimal way to serve the entire input seen so far
- Move all the servers so that they are at the locations of the optimal servers

Problems with this approach:

- Optimal configuration may change a lot from one step to the next
- Very expensive to keep chasing OPT with all the servers

Better idea: move just one server, try to get close to optimal configuration, don't travel too far

Work functions

- Configuration = set of locations of servers
- This is a multiset (two servers may be at same location)
- For a configuration C and input sequence σ, the work function $w_\sigma(C)$ is the minimum cost to reach C while serving σ (from the starting configuration)
- Suppose sequence so far is σ, new request is r
- How do we compute $w_{\sigma r}(C)$, given $w_\sigma(C)$?

Calculation of work function

- If $r \in C$, then $w_{\sigma r}(C) = w_\sigma(C)$
- Otherwise, we need to move **one** server from some other configuration B
- The difference between B and C is one point (server)
- We need to minimize the cost to get to B while serving σ, and then move to r
- Thus,

$$w_{\sigma r}(C) = \min_{x \in C}(w_\sigma(C - x + r) + d(x, r))$$

Definition of WFA

- Let C be the current configuration
- Let r be the new request
- We serve r with server $s \in C$ which satisfies

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

Notes:

- Minimizing only $d(x, r)$ is what the greedy algorithm does
- Minimizing $w(C - x + r)$ mimics OPT so far (retrospective greedy)

Idea behind WFA

- From C, we can move to k different configurations to serve r (we can move any of k servers)
- We move to the “best” one that is not too far away
- In effect, the algorithm is trying to find the optimal servers, without paying too much
- We do not use any properties of the metric space
- To apply this algorithm, we need to store the work function for all relevant configurations (with points where requests already occurred or where the servers started)$\rightarrow$very inefficient

Performance of WFA

- WFA is $(2k-1)$-competitive
- The proof uses a (complicated) potential function
- For some special metric spaces, WFA is known to be k-competitive
- E.g., the line, any metric space with at most $k+2$ points
- The popular conjecture is that WFA is k-competitive in any metric space