Knowledge State Algorithms and the 2-Server Problem in an Interesting Class of Metric Spaces

Lawrence L. Larmore
School of Computer Science
University of Nevada, Las Vegas

with Wolfgang Bein, Rüdiger Reischuk,
and Edward A. Larmore(Programming Consultant)

Grant support: NSF CCR-0132093

These slides are a (slightly updated) version of the talk I gave the Workshop on On-Line Algorithms on July 10, 2004:

`http://ola.imada.sdu.dk/`

I also presented a more extensive version at the *Zentrum für Angewandte Informatik Köln* (ZAIK) on June 25, 2004:

`http://www.zaik.uni-koeln.de/AFS/teachings/seminars/oberseminar.html`

Programming support was provided (in his spare time) by my son Edward, who works for Lockheed Martin. Bernhard Fuchs helped me refine the presentation at ZAIK. Some help with drawing the figures was provided by my 10-year old daughter Ruth.

The k-Server Problem

Schedule the movement of k servers in a metric space M in response to a sequence ϱ of requests

requests $\varrho = r^1 r^2 ... r^n$, points in M

after r^i: move one server to r^i

online: decision must be made before r^{i+1} is revealed

Goal: minimize total movement cost

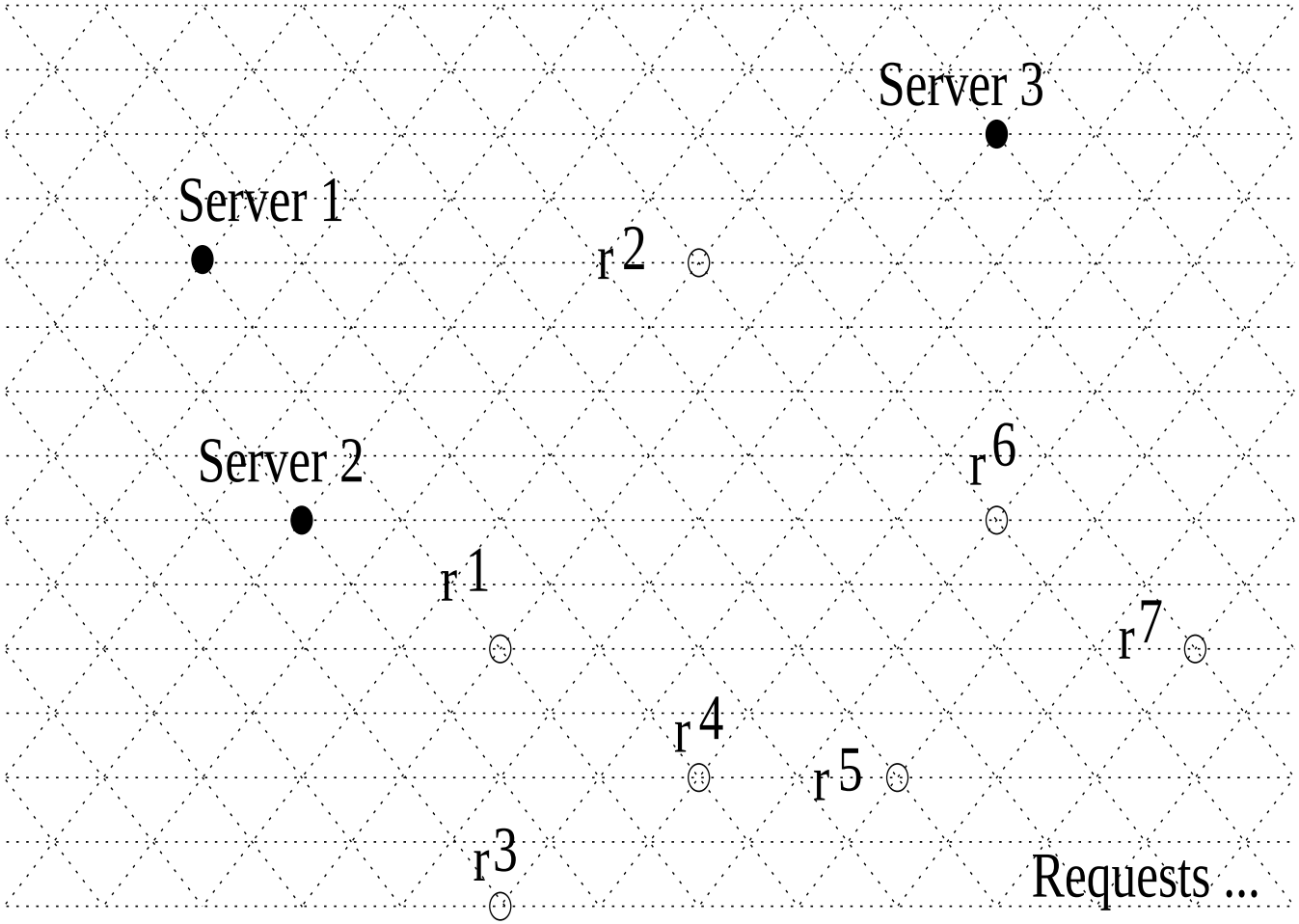
Narrative for Slide 2:

The classic k-server problem is defined as follows. There are k identical mobile *servers* in a metric space M. Initially, and after each *step* each server is located at some point in M.

There is a sequence of *requests* $r^1, \ldots r^n$, which are points in M. At step t, we must *serve* the request, *i.e.*, move one server to r^t. The *cost* is defined to be the distance the server is moved. The goal is to minimize the total cost over all steps.

It is a straightforward matter to compute the optimal schedule for servicing the request sequence if it is known in advance. An *online algorithm* for the k-server problem must service each request without knowing future requests.

Example with Three Servers



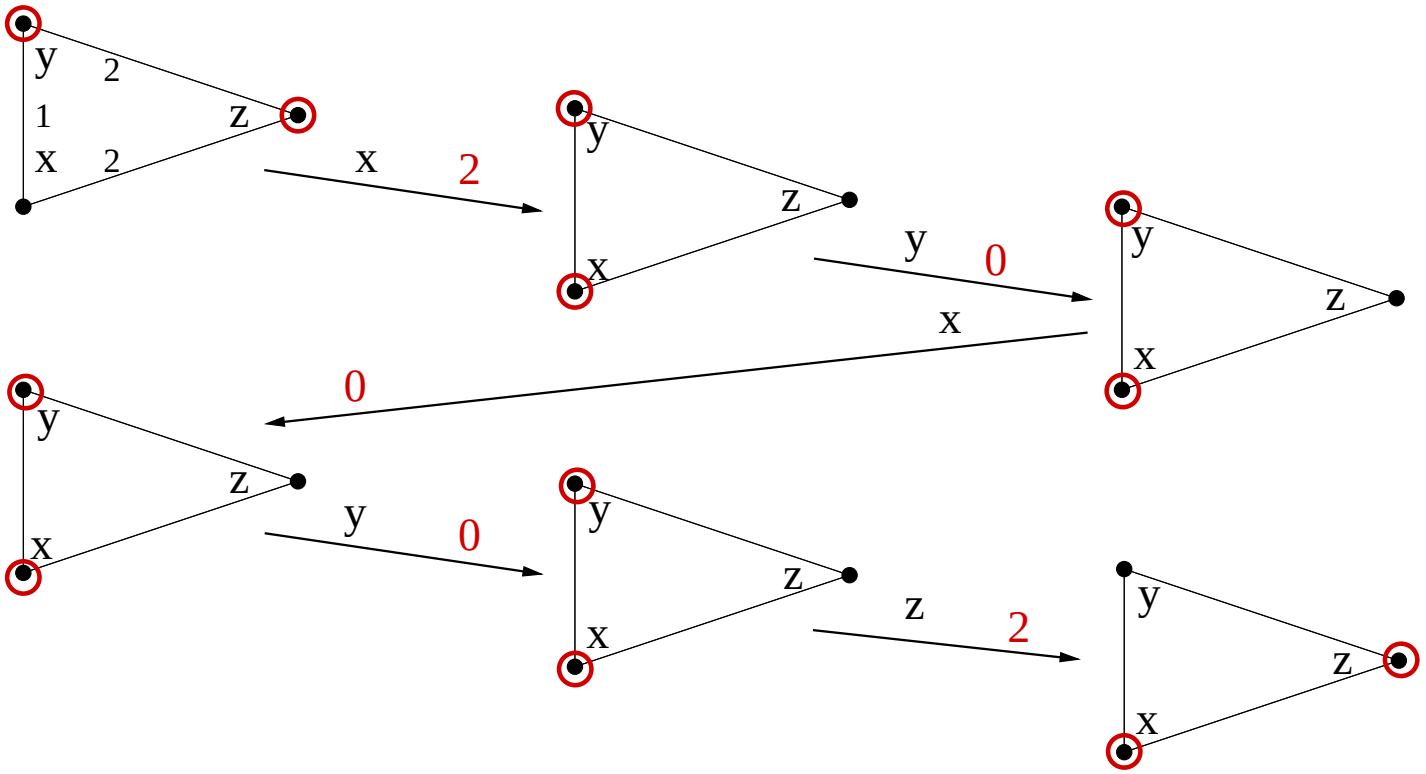
Narrative for Slide 3:

We show an instance of the 3-server problem in a metric space indicated by a graph. (The distance between points is the path length, for example, the distance between r^1 and r^4 is 3.)

Suppose that Server 2 serves r^1. When r^2 is revealed, it is not obvious to an online algorithm which server should serve that request, given that it does not know future requests.

Example: $k = 2$, $xy = 1$, $xz = yz = 2$, and $\varrho = xyxyz$.

Optimal schedule: Cost = 4



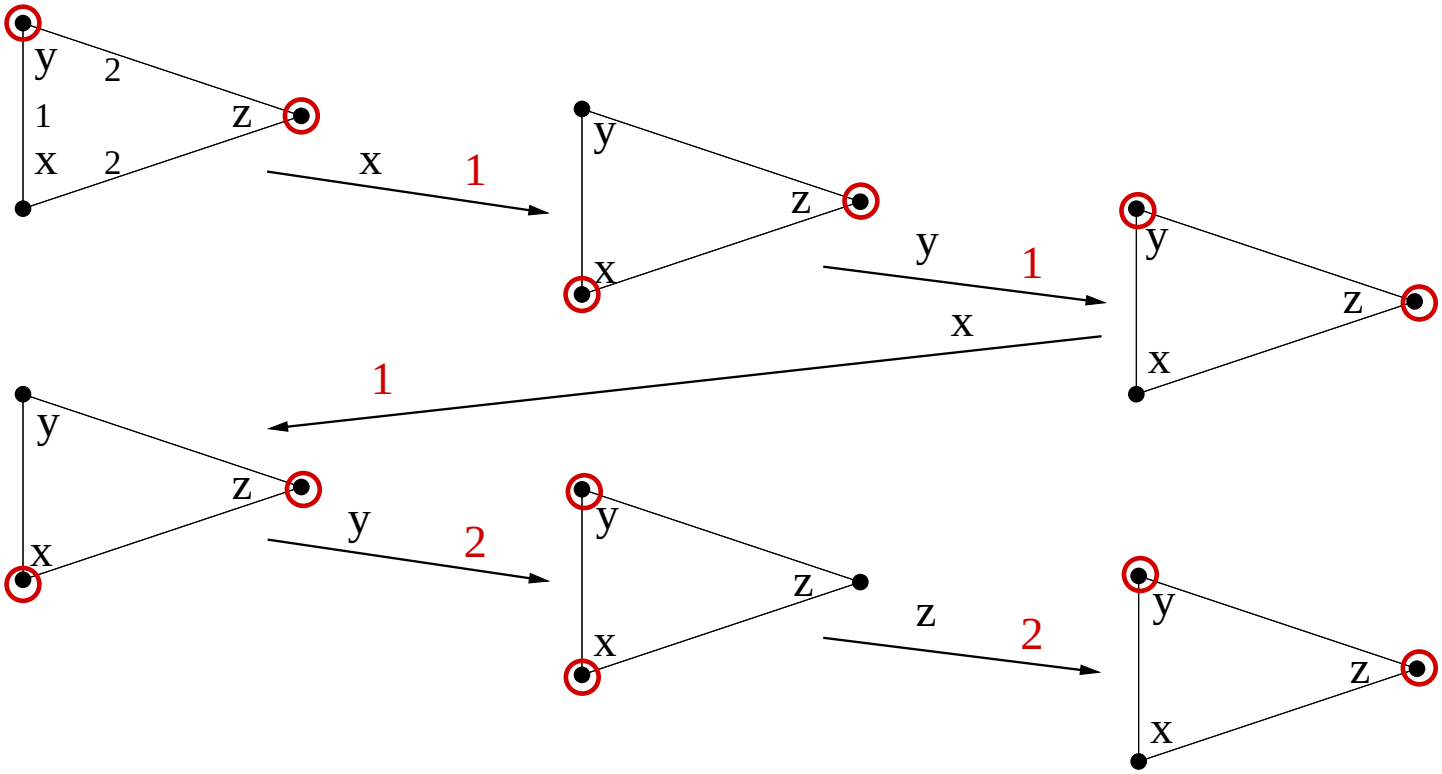
Narrative for Slide 4:

Consider the 2-server problem in a metric space consisting of three points, x, y, and z. The distance from x to y is 1, and other distances are 2. The initial configuration is $\{y, z\}$. Servers are represented by red circles.

The request sequence is $xyxyz$. The optimal schedule (*i.e.*, service) for this request sequence has cost 4, as shown.

It is "counter-intuitive" to serve the first request at x by moving the server from z. The only way you could know that is the optimal move is by knowing future requests.

Another schedule, Cost = 7



"Competitive ratio" = $\frac{7}{4}$

Narrative for Slide 5:

A different service for the same request sequence is shown. This one has cost 7. The ratio of the cost of this service to the optimal cost is $\frac{7}{4}$.

In general, an online algorithm (not knowing future requests) cannot achieve the optimal cost, but instead tries to minimize this "competitive ratio."

Competitiveness

For request sequence $\varrho = r^1, r^2, \ldots$ consider

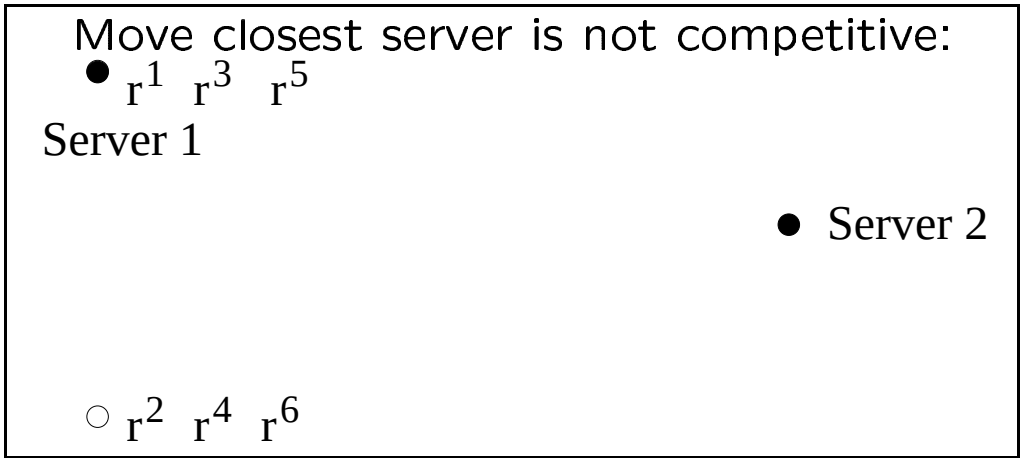
$cost_{\mathcal{A}}(\varrho)$: the cost on ϱ achieved by the servers of our online algorithm $\mathcal{A}$

$cost_{Opt}(\varrho)$: the cost on ϱ achieved by the server of an optimal offline algorithm

We say that a [randomized] algorithm is C-*competitive* if for each sequence ϱ we have

$$E\ cost_{\mathcal{A}}(\varrho) \leq C \cdot cost_{Opt}(\varrho) + \lambda$$

where λ depends only on the initial configuration



Narrative for Slide 6:

We give a formal definition of *competitiveness*, a traditionally accepted complexity measure for online algorithms. (The smaller the competitiveness, the better.) We genally say that an online algorithm is *optimal* if its competitiveness is the smallest possible for any online algorithm.

For randomized online algorithms, competitiveness is defined using the expected value of cost for a given request sequence. For many problems, the optimal competitiveness of a randomized online algorithm is smaller than the optimal competitiveness of a deterministic online algorithm. It is never larger.

Known about the 2-server problem:

- For any metric space M, there is a 2-competitive online algorithm for the 2-server problem.
- If a metric space M has at least 3 points, there is no deterministic algorithm for the 2-server problem on M whose competitiveness is less than 2.
- For many classes of metric spaces, there are randomized online algorithms with competitiveness less than 2.
- There is no known randomized online algorithm with competitiveness less than 2 for all metric spaces.
  [15? years]

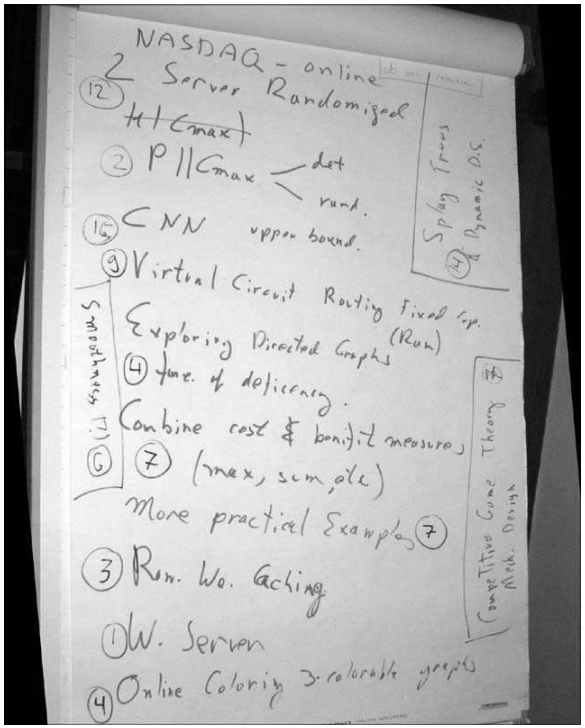
Narrative for Slide 7:

It was conjectured by Manasse, McGeogh, and Sleator (1991) that the optimal competitiveness of the k-server problem is k for general metric spaces. (This is the "notorious" k-server conjecture.) They also proved that this is a lower bound for all metric spaces with at least $k + 1$ points. But the problem remains open except for some special classes of metric spaces, and for the case $k = 2$ for all metric spaces.

Randomization has not been shown to decrease the competitiveness of the 2-server problem in general, although there are randomized online algorithms for the 2-server problem that are less than 2-competitive for some classes of metric spaces, such as the line, and all three point spaces. The conjecture that such a general algorithm exists has been open for years.

The True Quest!

Find a randomized algorithm for the 2-server problem that is C-competitive on all metric spaces, for some $C < 2$.



Narrative for Slide 8:

At the On-Line Workshop in 2002 at Dagstuhl, a number of open problems were presented, and participants voted on the importance of each. Three problems received a large number of votes, and one of them was the conjecture that there is a C-competitive randomized online algorithm for the 2-server problem for some $C < 2$.

The photo shows the poster on which votes were counted.

WFA, *etc.*.	all	$C = 2$	deterministic	McGeoch, Sle
Harmonic	all	$C = 3$	memoryless	Chrobak *et*
Random Slack	all	$C = 2$	memoryless	Coppersmith *e*
Six Thirds	line	$C = \frac{155}{78}$	barely random	Bartal *et a*
Unnamed	line	$C = \frac{71}{36}$	order 2 knowledge state (computer)	Bein *et al*
Partition	uniform	$C = \frac{3}{2}$	work function distribution	McGeoch *et*
Equitable	uniform	$C = \frac{3}{2}$	forgiveness finite memory	Achlioptas *et*
K_2	uniform	$C = \frac{3}{2}$	order 2 k.s. remember 1 point	Bein *et al*
It depends	3 points	$C = \frac{e}{e-1}$	work function distribution	Reingold *et*

Narrative for Slide 9:

Some known upper bounds for the competitiveness of 2-server problem are listed.

Several deterministic algorithms (including WFA, explained below) are 2-competitive for all metric spaces.

HARMONIC is a memoryless randomized algorithm which moves a server with probability inversely proportional to its distance to the request point. It has been shown that HARMONIC is 3-competitive for the 2-server problem, and this is a known lower bound for HARMONIC.

The randomized competitiveness for each individual 3-point metric space was worked out by Lund and Reingold. $C < \frac{e}{e-1}$ is all cases.

Narrative for Slide 9 (continued):

A memoryless randomized algorithm which we call RANDOM_SLACK was introduced by XXXX, and is 2-competitive for all metric spaces.

The *paging problem* is equivalent to the server problem for uniform metric spaces. The optimal determinisic competitiveness is k, and is achieved by the well-known algorithm LRU. The optimal randomized competitiveness is known to be the *harmonic number* $H_k = \sum_{i=1}^{k} \frac{1}{i}$. EQUITABLE achieves that competitiveness, at the cost of unlimited memory. PARTITION achieves the same competitiveness with memory which is finite, but which grows rapidly with k.

We have introduced a randomized algorithm, K_2, for the case $k = 2$, which uses very small memory and only one bookmark.

Narrative for Slide 9 (continued):

For the line, a randomized $\frac{155}{78}$-competitive algorithm was published with Bartal and Chrobak. In this algorithm, there are six "third-servers." To service a request, three of them move deterministically to the request point. Somewhat later, we constructed, using a computer program, a $\frac{71}{36}$-competitive randomized algorithm for the line. No hand proof has yet been published. The value $\frac{71}{36}$ was chosen for descriptive convenience, and is not the best that can be achieved by this method.

Some Lower Bounds

any	deterministic	$C \geq 2$	McGeoch *et al.*
any	memoryless	$C \geq 2$	McGeoch *et al.*
line		$C \geq 1+e^{-\frac{1}{2}}$ ≈ 1.6065	Chrobak *et al.*
general	trackless deterministic	$C \geq \frac{23}{11}$	Bein *et al.*
general	trackless	$C \geq 1+\frac{\sqrt{2}}{2}$	Bein *et al.*
uniform		$C \geq \frac{3}{2}$	Fiat, Karp, *et al.*
uniform	trackless	$C \geq \frac{8453}{5458} \approx 1.548735$	Bein *et al.*

Narrative for Slide 10:

No deterministic online algorithm for the 2-server problem can achieve comnpetitiveness less than 2.

No memoryless randomized online algorithm for the 2-server problem can achieve competitiveness less than 2.

No randomized online algorithm for general spaces can achieve competitiveness less than $1 + e^{\frac{1}{2}}$. The space for which the lower bound is proved consists of three equally spaced points on a line, together with a fourth point "infinitesimally" close to the middle point. Thus, the lower bound holds for the line as well.

Narrative for Slide 10 (continued):

An online server algorithm is said to be *trackless* if it does not keep track of points where it does not have servers. (A trackless algorithm may have memory, however.) For the paging problem, this means that the algorithm may not keep any bookmarks.

BALANCE_SLACK is a 4-competitive deterministic trackless algorithm for the 2-server problem in general metric spaces. RANDOM_SLACK is a 2-competitive randomized trackless algorithm for the 2-server problem in general metric spaces. There is known lower bound of $\frac{23}{11}$ for the competitiveness of any deterministic trackless algorithm for the 2-server problem for general spaces, and a lower bound of $1 + \sqrt{2}/2$ for the randomized case.

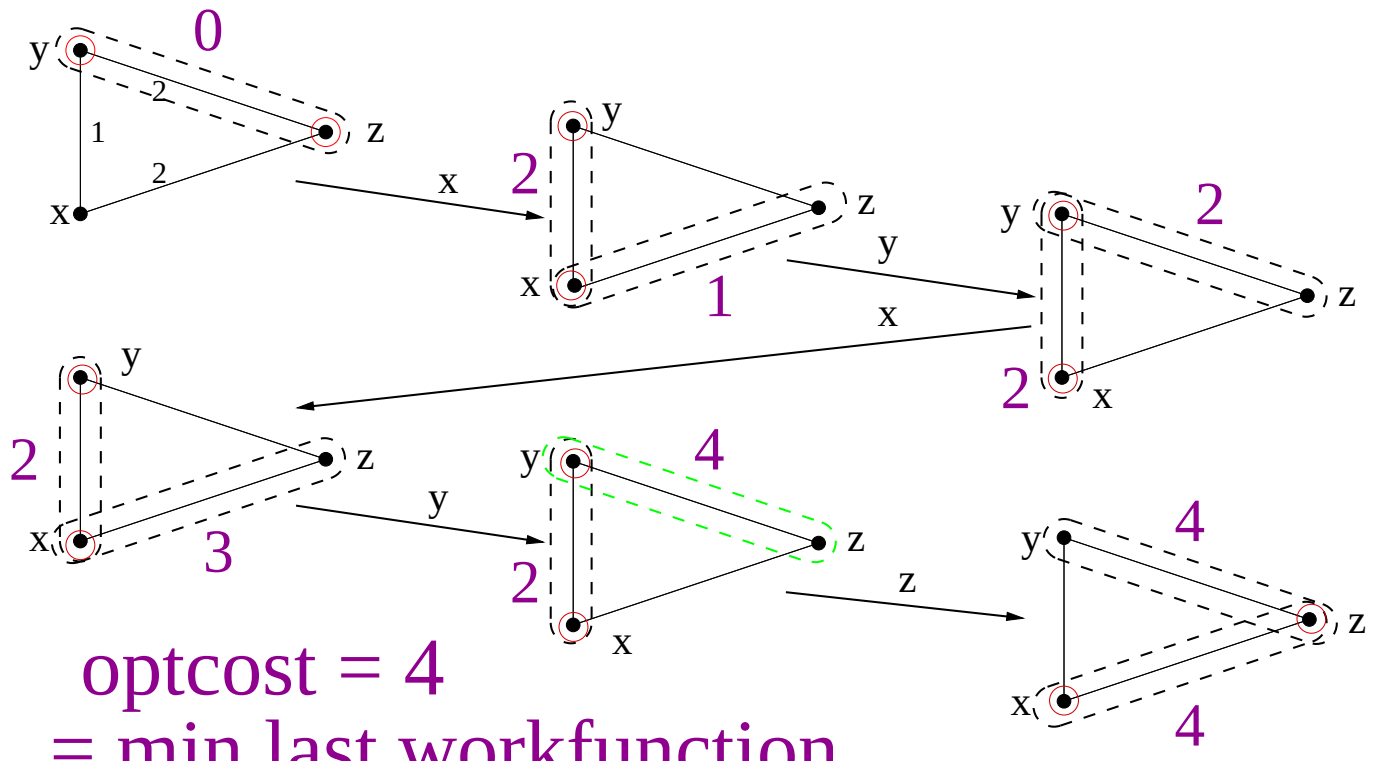
Narrative for Slide 10 (continued):

For the trackless uniform case (*i.e.*, paging) there is a lower bound of $\frac{8453}{5458}$ obtained by computer for $k = 2$. (A hand proof exists for a more modest value, which is still greater than $\frac{3}{2}$.) An upper bound of $\frac{3+\sqrt{13}}{4} \approx 1.6514$ was obtained by Chrobak, Koutsoupias, and Noga.

Work Functions

Function on configurations: Dynamic programming

The optimal cost of being **there then**



Narrative for Slide 11:

For an online problem, the domain of ω^t, the *work function at step* t, is the set of all configurations that are possible after servicing the first t requests. The value of ω^t at a configuration X is the minimum cost of servicing the first t requests and ending at X. For example, $w^3(x, z) = 3$, the minimum cost of servicing the request sequence xyx, while starting in configuration $\{y, z\}$ and ending in configuration $\{x, z\}$.

The values of ω^t are computed from ω^{t-1} by dynamic programming. The minimum value of ω^n is the minimum cost of servicing the entire request sequence $r^1 \ldots r^n$.

The red circles represent optimal server positions. Note that, given its knowledge of the future, the optimal off-line algorithm does not always move to the configuration which minimizes the work function.

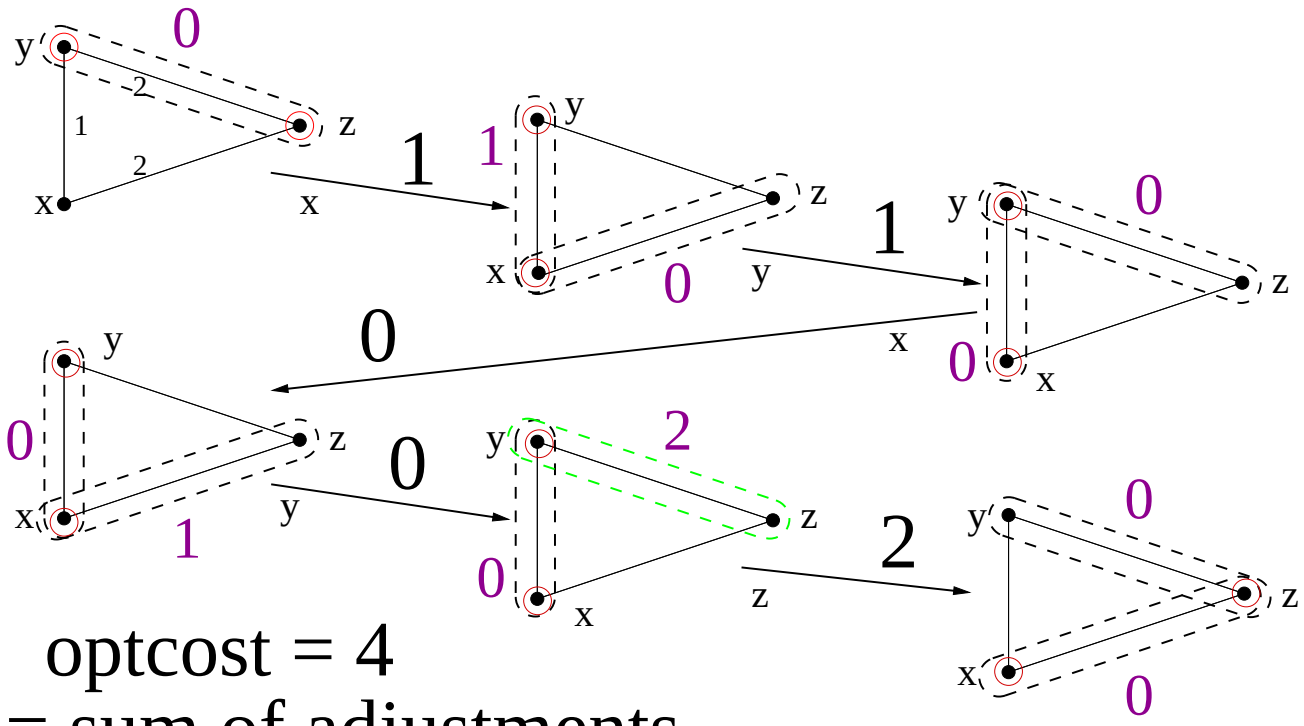
Narrative for Slide 11: (continued)

If ω is the work function, and if Y is a configuration, we say that the value of ω on Y is *supported* by the value of ω on X if $\omega(Y) = \omega(X) + d(X, Y)$, where $d(X, Y)$ denotes the cost of moving from the configuration X to the configuration Y. We define the *support* of the work function to be the minimal set of configurations such that the values of ω on that set support the values on all configurations.

Note that the configuration $\{y, z\}$ is not in the support of the work function ω^4, since $d(\{x, y\}, \{y, z\}) = 2$, $\omega^4(y, z) = 4$, and $\omega^4(x, y) = 2$. In the figure and in subsequent figures in this talk, we indicate the fact that a a configuration is not in the support by either coloring the dotted oval enclosing the pair light green, or omitting it entirely. Without loss of generality, the optimal off-line algorithm is always at some configuration which is in the support.

Amortize the Optimal Cost!

Subtract adjustment at each step, making minimum zero



optcost = 4
= sum of adjustments

Narrative for Slide 12:

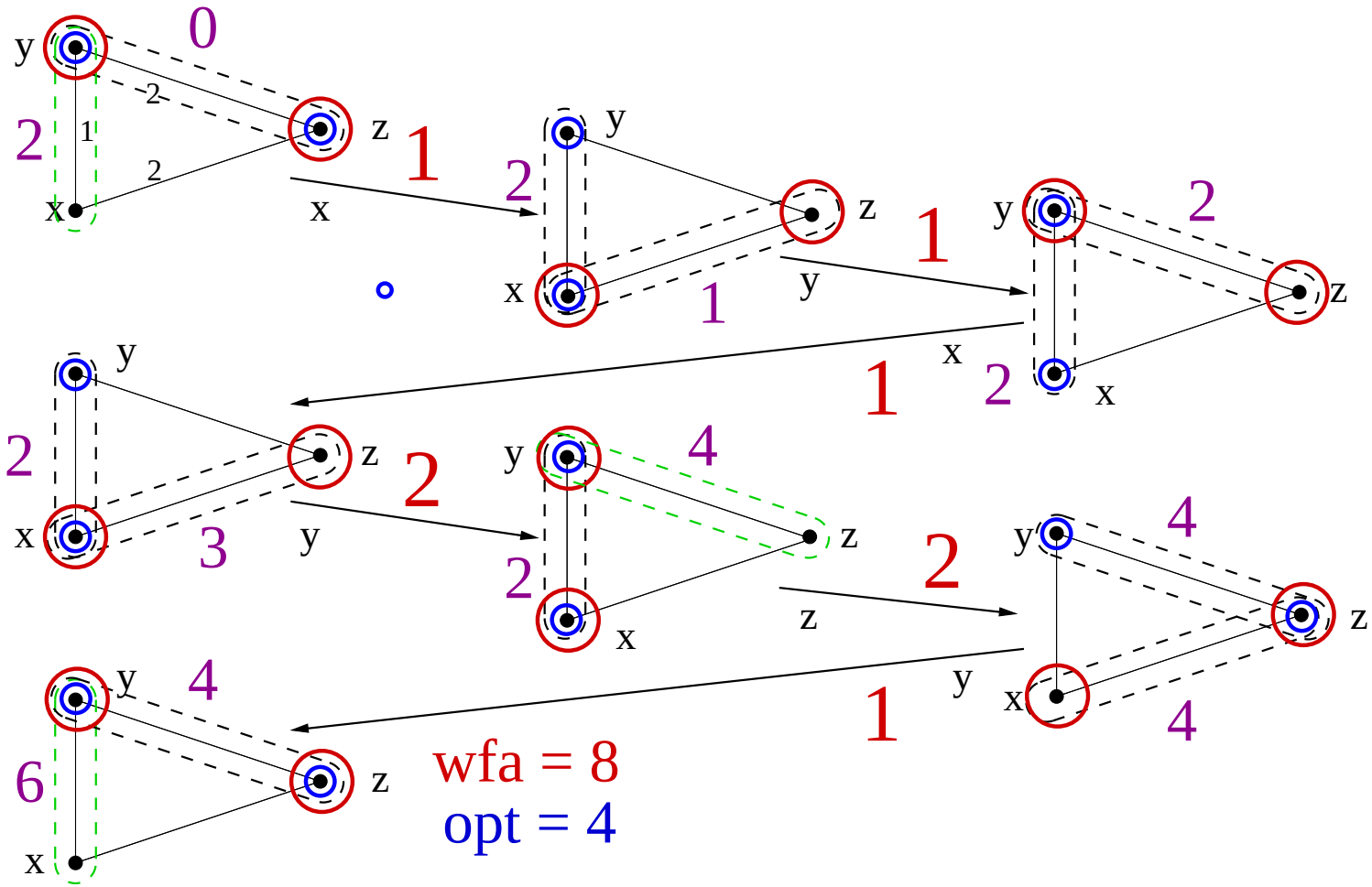
It is sometimes convenient to subtract a constant from the work function at each step. We call this the *amortized optimal cost*, or *offset*, or *adjustment*. For example, after one step, the online algorithm knows that the offline algorithm has paid either 1 or 2, depending on its configuration. It then "demands" a payment of 1, since it knows that the offline algorithm has paid at least that much. The resulting function (which is sometimes called the *offset function*, and which we later call the *estimator*) shows the remaining "debt" of the offline algorithm as a function of its current configuration.

After all requests are made, the estimator has a minimum of zero, and the total of the adjustments up to that point is equal to the optimal cost. It is important to remember that this is just a bookkeeping device, and does not differ in substance from the work function approach.

The Work Function Algorithm (WFA)

2-Competitive. Choose configuration to minimize

cost + work function



Narrative for Slide 13:

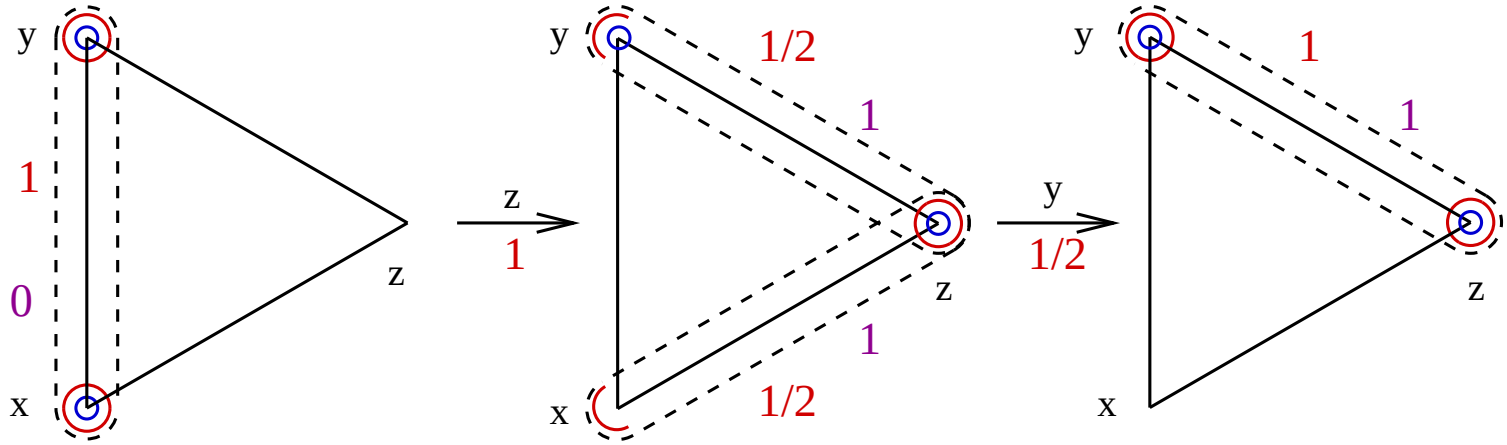
The *work function algorithm* is the deterministic online algorithm (defined for a wide variety of online problems) which, at each step, makes that move which minimizes the sum of the cost of that move and the value of the work function at the new configuration. (Trivially, the offset function can be substituted for the work function in this definition.) In case of ties, we assume the adversary makes the choice.

In the example shown, with the same space as before, with the request sequence $xyxyzy$, the optimal cost is 4 and WFA pays 8. The optimal servers are shown as small blue circles, while WFA's servers are shown as larger red circles. To achieve the limiting ratio of 2, the adversary can repeat the the same request sequence as many times as necessary.

Does Randomization Help?

PARTITION is $\frac{3}{2}$-Competitive for Uniform Spaces

Distribution is Uniform on the support of Work Function



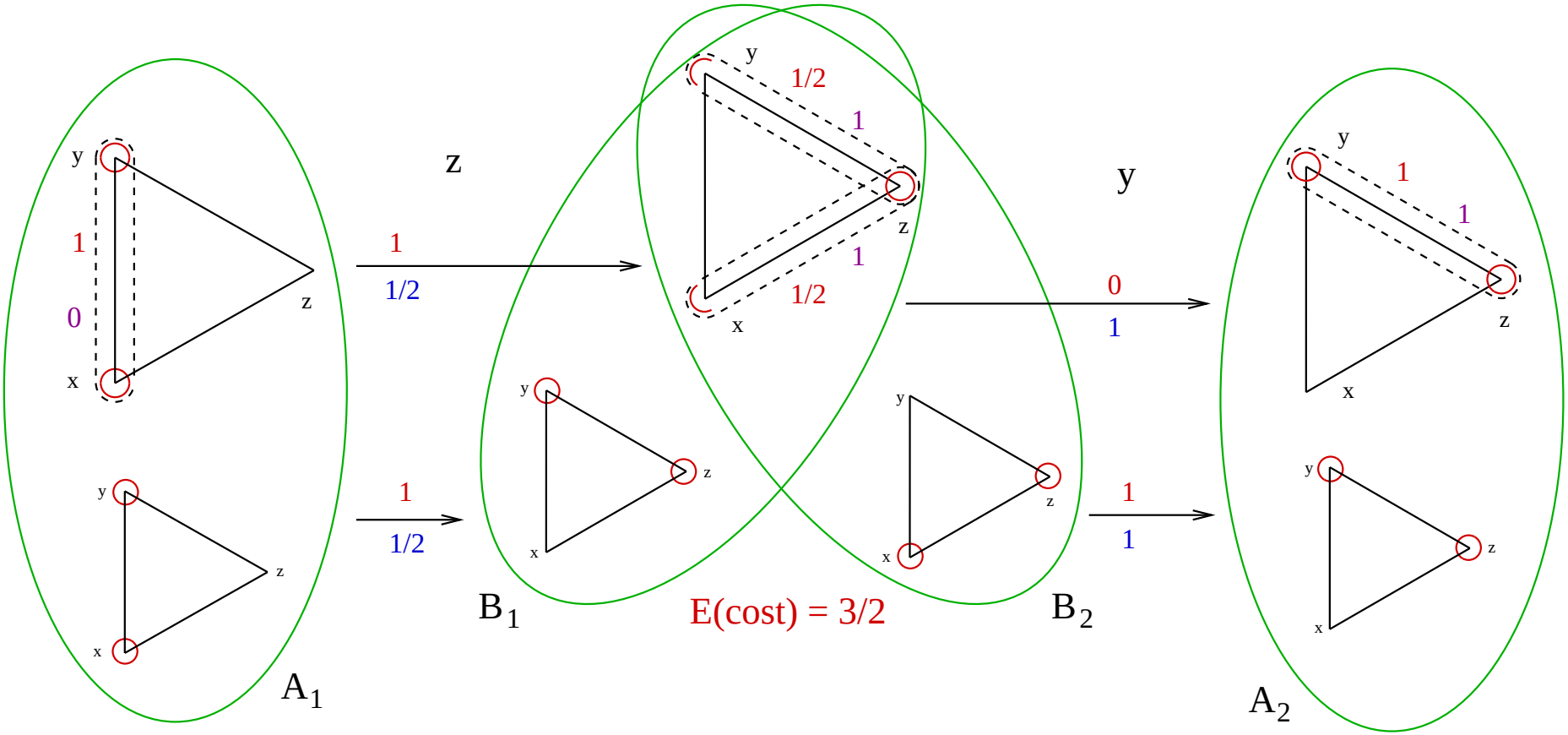
cost = 3/2
optcost = 1

Narrative for Slide 14:

PARTITION is a randomized online algorithm for the paging problem (*i.e.*, the server problem for uniform spaces) which is described using the *distributional model*, meaning that the algorithm chooses a distribution at each step. Cost in the distributional model is computed as minimum transportation distance. In the case that $k = 2$, PARTITION's distribution is uniform on the support of the current work function.

Only the values of the work function on the support are shown. The red values on those same configurations are the distributional weights for PARTITION, must sum to 1. The competitiveness is $\frac{3}{2}$, and the example shown, repeated as many times as necessary, gives the worst case.

The Underlying Behavioral Algorithm of PARTITION



This Talk

We present a $\frac{7}{4}$-competitive

Randomized Algorithm for the 2-Server Problem

In the class of spaces we call $\mathcal{M}_{24}$.

We use the knowledge state approach.

Def: $\mathcal{M}_{24}$ consists of all metric spaces such that

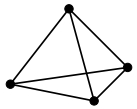
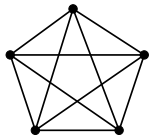
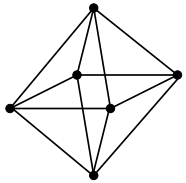
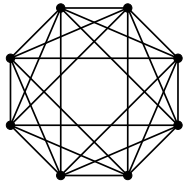
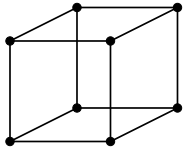
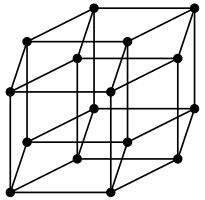
1. All distances are 1 or 2.
2. $d(x,y) + d(x,z) + d(y,z) \leq 4$

Why $\mathcal{M}_{24}$?

1. A step in the direction of the goal (better than 2-competitive randomized algorithm for the 2-server problem).
2. Allows a simple example of the knowledge state method.
3. An interesting class in its own right, derived from the cross polytopes, an infinite family of regular convex polytopes, generalizing the octahedron.

Exactly Three Infinite Families of Convex Regular Polytopes

(Ludwig Schläfli, 1852)

Infinite Family of Regular Polytopes	Graph Class	Metric Space Class	3–d	4–d
Regular Simplices	Complete Graphs	Uniform Spaces		
Cross Polytopes = Orthoplices	Circulant Graphs $Ci_{1,\ldots,n-1}(2n)$	M_{24}		
Hypercubes	Hypercubes	Hamming Metric Spaces		

Narrative for Slide 18:

Any polytope gives rise to a graph by considering only the faces of dimensions 0 and 1. The vertices then form a metric space, using minimum path length as the metric. All distances are integers.

As first published by Schläfli, there are exactly three regular polytopes in each dimension $n \geq 5$. (As opposed to five in dimension 3 and six in dimension 4.) There are three infinite families of regular polytopes, the familiar simplices and hypercubes, and the less familiar cross polytopes. For example, consider all vectors in $\mathbf{R}^n$ where one coordinate is 1 or -1 and all other coordinates are 0. The convex hull of that set of $2n$ vectors is a cross polytope of dimension n. The familiar octahedron is the cross polytope of dimension 3.

Narrative for Slide 18 (continued):

The class $\mathcal{M}_{24}$ is the class of all metric spaces M such that every finite subspace of M is a subspace of the metric space derived from some cross polytope.

What is a Knowledge State?

- $\mathcal{X}$ = all configurations (pairs of points), itself a metric space under the minimum matching metric.
- A knowledge state $k = (\omega, \pi)$ where:
- $\omega : \mathcal{X} \to \mathbf{R}$ is the estimator.
- π is a distribution on $\mathcal{X}$.
- If $\{x, y\} \in \mathcal{X}$, then
  - $\pi(x, y)$ is the probability we are at $\{x, y\}$.
  - $\omega(x, y)$ is the estimated unpaid cost of the adversary if it is at $\{x, y\}$.
- A knowledge state also has a potential, $\phi(k)$.

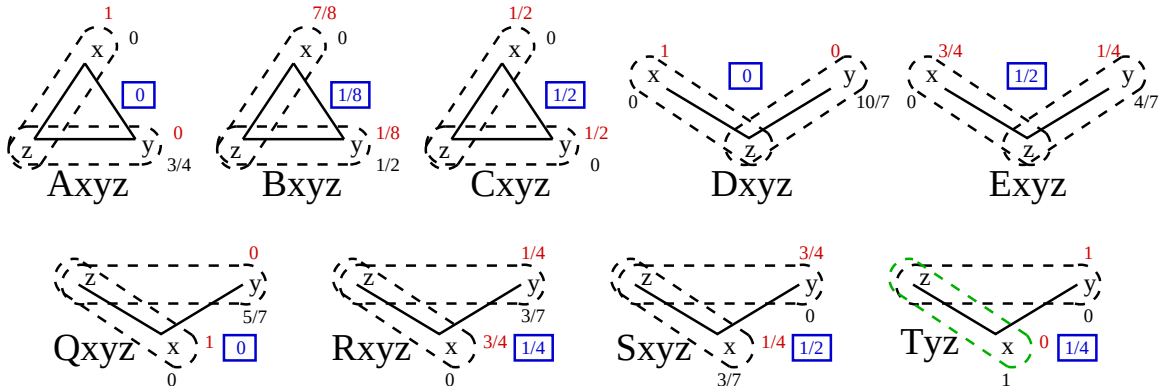
Narrative for Slide 19:

The estimator estimates how much "credit" the adversary owes us if it is at a given configuration. Unlike the classic work function, which gives a precise value, the estimator is only guaranteed to have a value which is at most the expected value of the credit owed us.

Knowledge state algorithms are a special case of a more general class, "mixed model randomized algorithms" where there is a distribution of our configurations and a "memory" state. The knowledge state functions as our memory state.

The mixed model generalizes both the behavioral model and the distribution model.

An order 2 knowledge state over (x, y, z), where z is the last request, is defined by the value of its estimator ω and distribution π on its support, the two configurations $\mathcal{S} = \{\{x, z\}, \{y, z\}\}$.

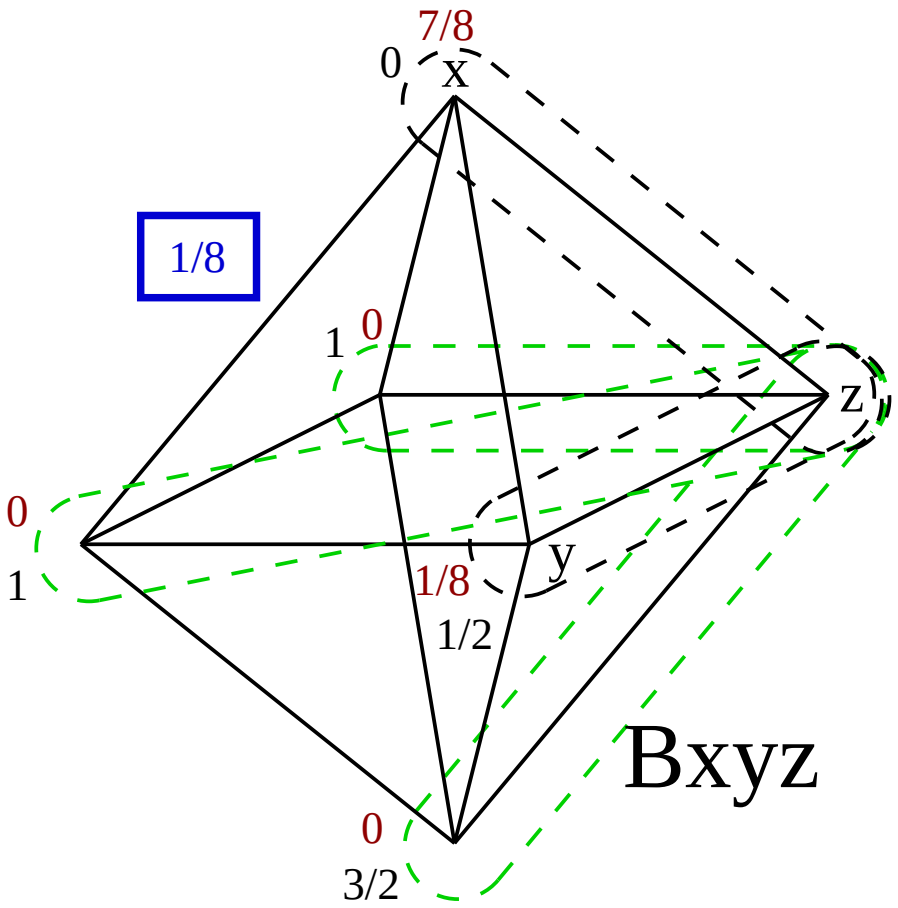


Up to symmetry, there are 9 standard knowledge states of our $\frac{7}{4}$-competitive algorithm for $\mathcal{M}_{2,4}$.

Narrative for Slide 20:

For a given metric space $M \in \mathcal{M}_{24}$, and for given points x, y, z such that $d(x,y) = d(x,z) = d(y,z) = 1$, then the knowledge states $Axyz$, $Bxyz$, and $Cxyz$ are defined. Similarly, if $x, y, z \in M$ and $d(x,y) = 2$ and $d(x,z) = d(y,z) = 1$, then the knowledge $Dxyz$, $Exyz$, $Eyxz$, and $Dyxz$ are defined.

There is one case where the knowledge state has only two parameters: if $d(y,z) = 2$, then Tyz is defined. In the figure, an extra point x is shown of distance 1 from both y and z, but the pair $\{x, z\}$ plays no role, as indicated by the green color.



Bxyz

The estimator and distribution are defined for all configurations but characterized by their values only on the

support

If $X \in \mathcal{X} - \mathcal{S}$, then

- $\pi(X) = 0$.
- $\omega(X) = \min_{Y \in \mathcal{S}} \{\omega(Y) + \|X, Y\|\}$

Narrative for Slide 21:

The distribution and estimator are actually defined for all configurations, but the algorithm need only know their values on the support configurations.

The value of the distribution is zero on any other configuration, and the value of the estimator is the maximum possible subject to the *Lipschitz* constraint, namely that $\omega(Y) - \omega(X) \leq \|X, Y\|$, where $\|X, Y\|$ is the cost of moving from configuration X to configuration Y.

In this talk, configurations in the support are shown in black, while other configurations are shown in green, if at all.

One Move of the Algorithm

- Start at a standard knowledge state over (x, y, z).
- Read a request $r \neq z$. (The case $r = z$ is trivial.)
- Update the estimator. No payment either way.
- Move the distribution. We pay.
- Adjust the estimator. The adversary pays us.
- Optionally forgive; lower the estimator selectively. No payment.
- Las Vegas. Randomly pick a subsequent according to an appropriate distribution. No payment: "fair" wager.
- We are at a standard knowledge state over (x, y, r), (y, x, r), (x, z, r), (z, x, r), (y, z, r), or (z, y, r).

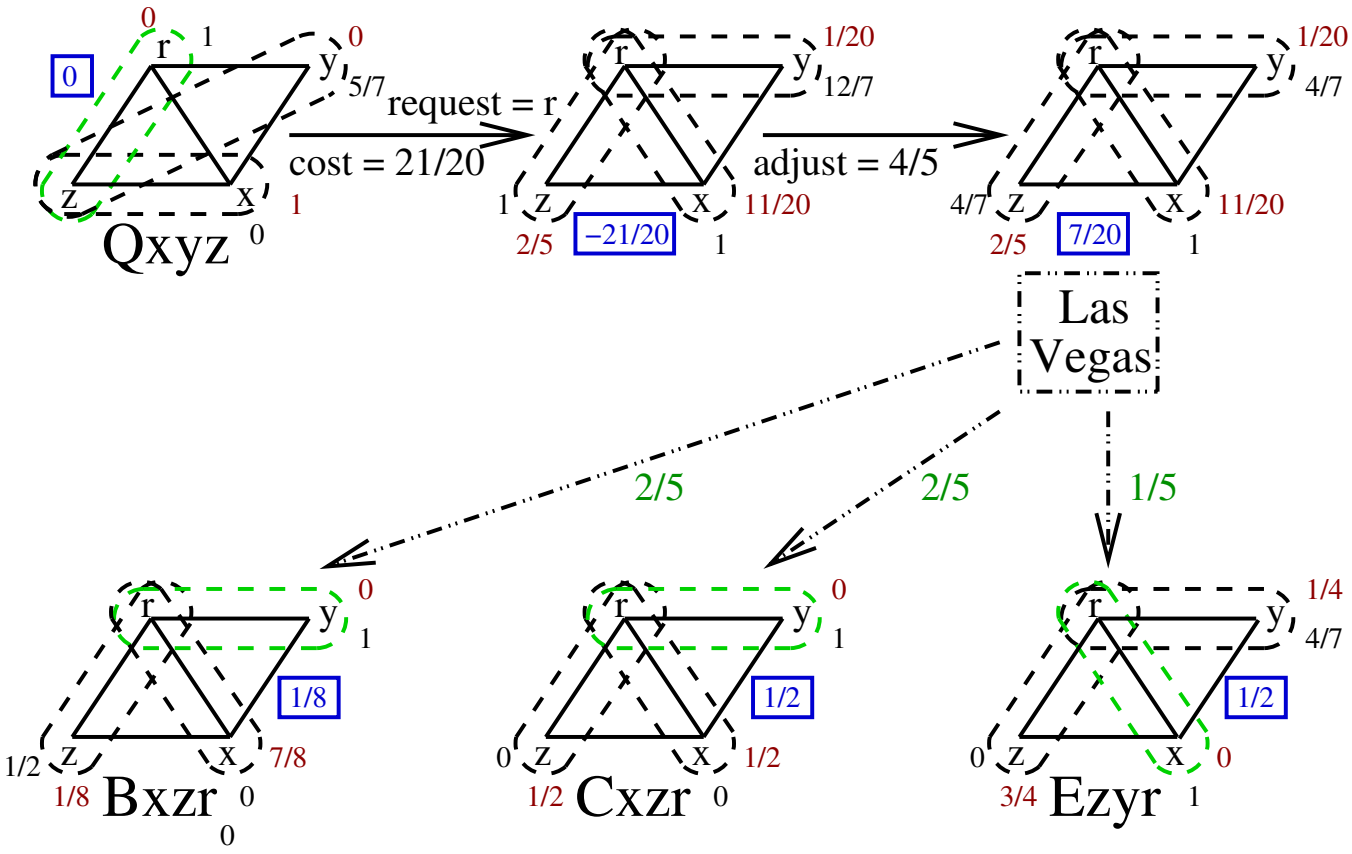
Narrative for Slide 22:

As the number of requests grows, the complexity of the classic work function increases without bound. On the other hand, if we assume that one memory unit is needed to store the name of any point, a finite order knowledge state algorithm with finitely many knowledge states can be implemented using finite memory.

In our case, the algorithm need only remember:

- Its current configuration, which consist of the last request point and the location of the other server,
- the name of one other point,
- the current knowledge state, parameterized by those three points.

There are 39 Moves. Here is One.



Narrative for Slide 23:

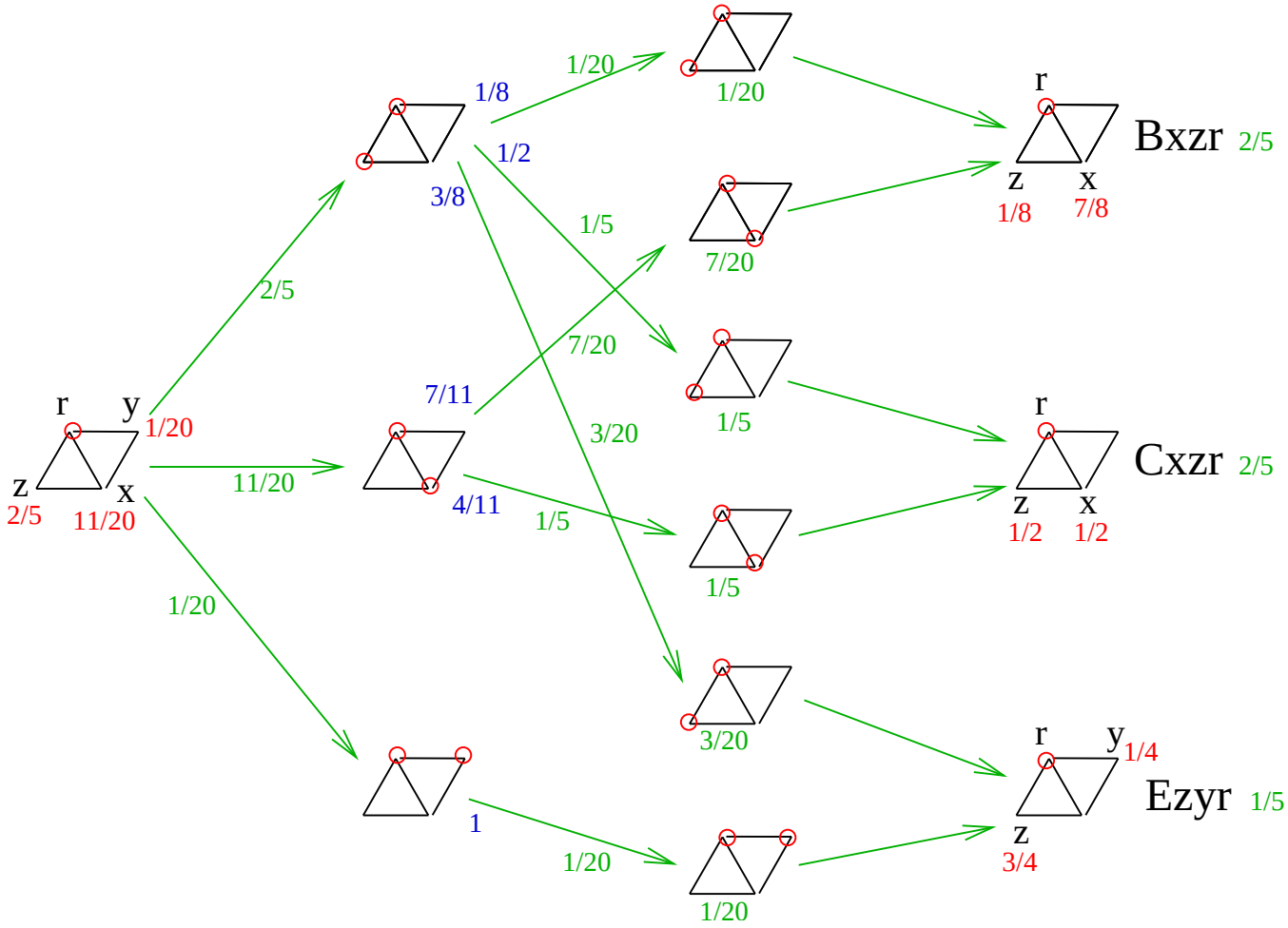
One example move is shown in this slide. Suppose the current knowledge state is $Qxyz$, which implies that $d(x,y) = d(x,z) = 1$ and $d(y,z) = 2$. Our potential is zero for this knowledge state. The probability that our actual configuration is $\{x,z\}$ is 1. We estimate that the adversary owes us for expenditure of 0 if it is at $\{x,z\}$, $\frac{5}{7}$ if it is at $\{y,z\}$, and 1 if it is at any other configuration which contains z.

Now suppose that r is requested, where $d(r,x) = d(r,y) = d(r,z) = 1$. Move our distribution so that we are at $\{x,r\}$ with probability $\frac{11}{20}$, at $\{y,r\}$ with probability $\frac{1}{20}$, and at $\{z,r\}$ with probability $\frac{2}{5}$. The expected cost of this move is $\frac{21}{20}$, which we must subtract from the potential. (The resulting negative potential is temporary.) We update the estimator, resulting in an estimator supported by three configurations.

Narrative for Slide 23 (continued):

We now demand payment of $\frac{4}{5}$ from the adversary. Because the competitiveness is $\frac{7}{4}$, we actually receive $\frac{7}{5}$ credit, making our potential $\frac{7}{5} - \frac{21}{20} = \frac{7}{20}$. We subtract $\frac{4}{5}$ from the estimator for each configuration, resulting in a second temporary state.

The final step is the Las Vegas step. With probability $\frac{2}{5}$, we move to the knowledge state $Bxzr$. With probability $\frac{2}{5}$, we move to the knowledge state $Cxzr$. With probability $\frac{1}{5}$, we move to the knowledge state $Ezyr$. Each of these subsequents has a distribution and an estimator. Note that the weighted sum of those distributions is equal to the distribution of the second temporary state. Similarly, the weighted sum of the estimators of the subsequents is equal to the estimator of the second temporary state.

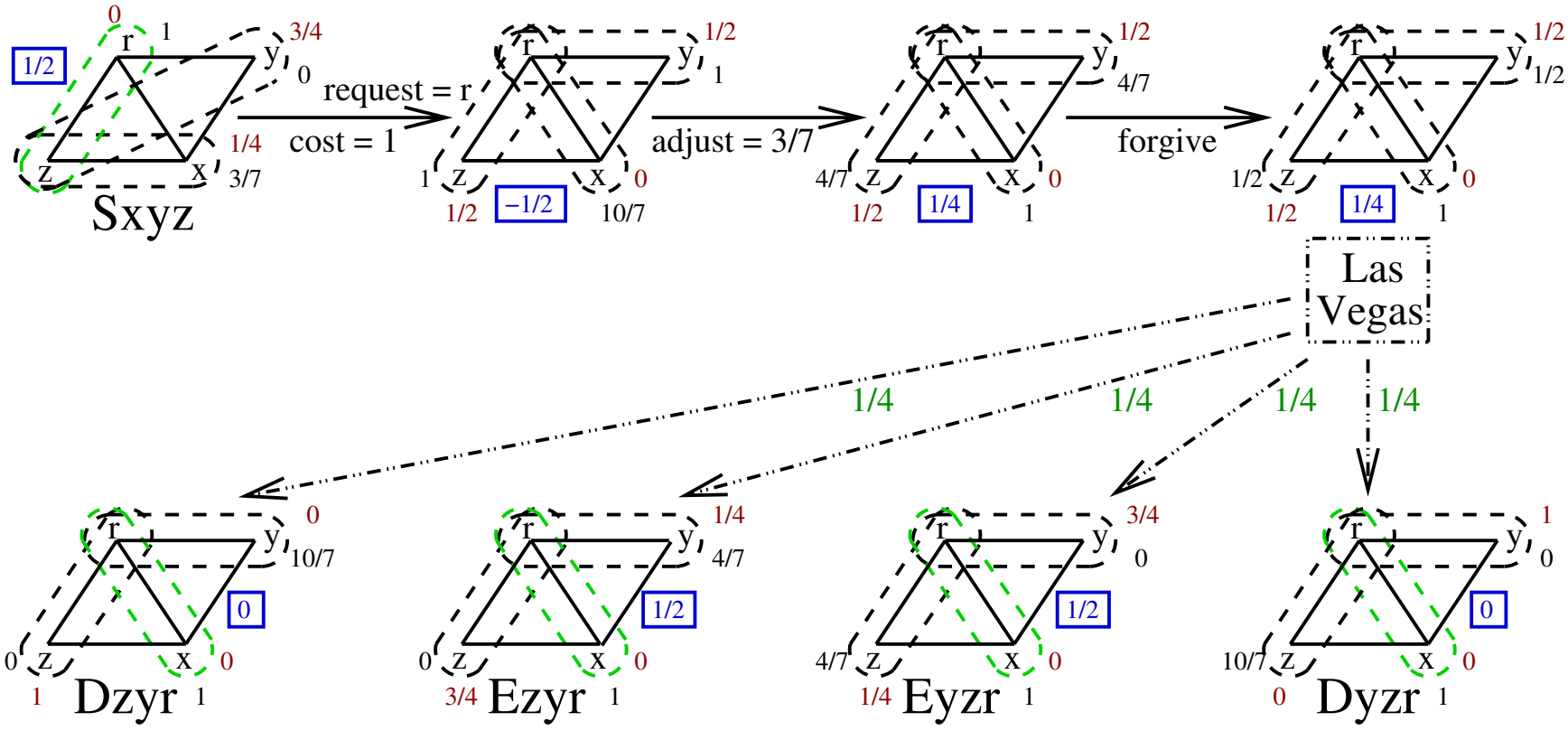

r
y
1/20
z
2/5
x
11/20
2/5
11/20
1/20
1/8
1/2
3/8
7/11
4/11
1
1/20
1/20
7/20
1/5
3/20
1/5
1/5
1/20
1/20
7/20
1/5
3/20
r
z
1/8
x
7/8
Bxzr 2/5
r
z
1/2
x
1/2
Cxzr 2/5
r
y
1/4
z
3/4
Ezyr 1/5

Narrative for Slide 24:

In this slide, we illustrate the implementation of the Las Vegas step.

In the second temporary state, we are actually at one of the three configurations, as shown in the second column of the figure. In the Las Vegas step, we do not actually move any server, so our cost is zero. Instead, we merely update our memory, using randomization. If our servers are at $\{x, r\}$, we change our memory to $Bxzr$ with probability $\frac{7}{11}$, and to $Cxzr$ with probability $\frac{4}{11}$. If our servers are at $\{y, r\}$, we change our memory to $Ezyr$. If our servers are at $\{z, r\}$, we change our memory to $Bxzr$ with probability $\frac{3}{8}$, to $Cxzr$ with probability $\frac{1}{2}$, and to $Ezyr$ with probability $\frac{3}{8}$.

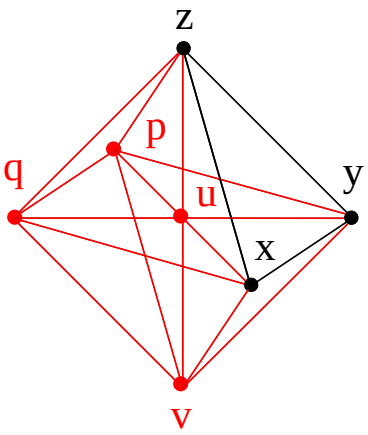
Another Example

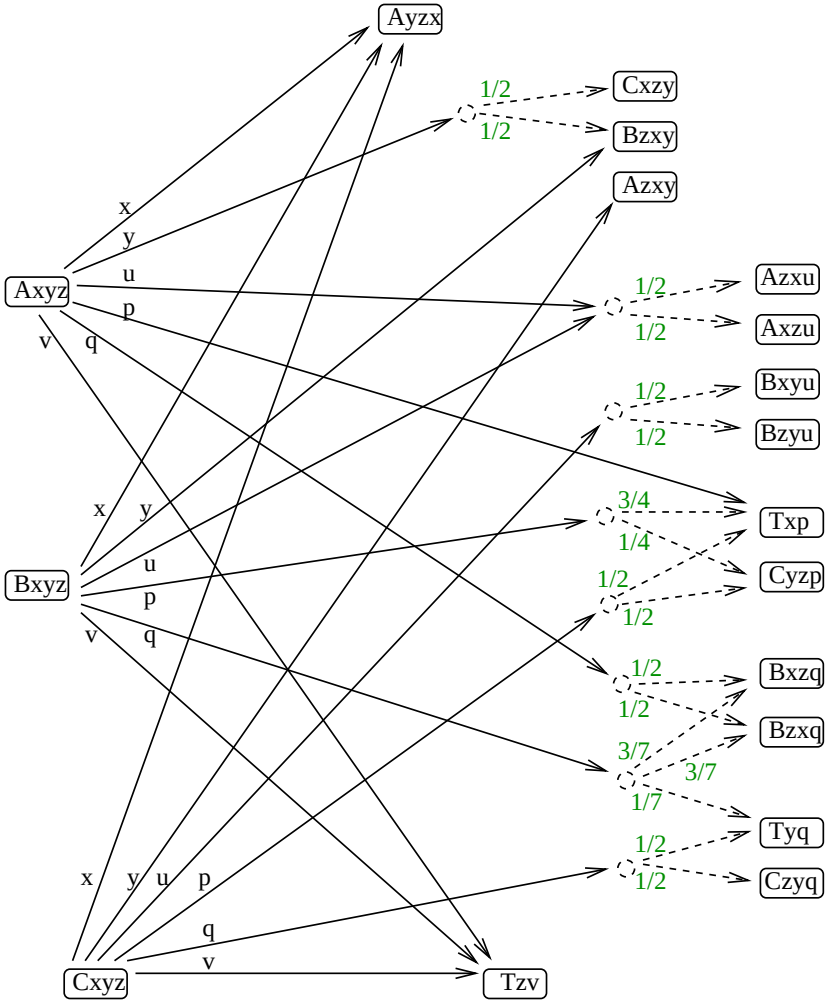


Narrative for Slide 25:

In this slide, we illustrate another move. For this move, we need one more temporary state. The move from the second temporary state to the third we call *forgiveness*. In this move, we generously forgive part of the adversary's debt to us, provided it is at configuration y, r or z, r, but we do not collect any credits.

All Moves from Knowledge States A, B, C

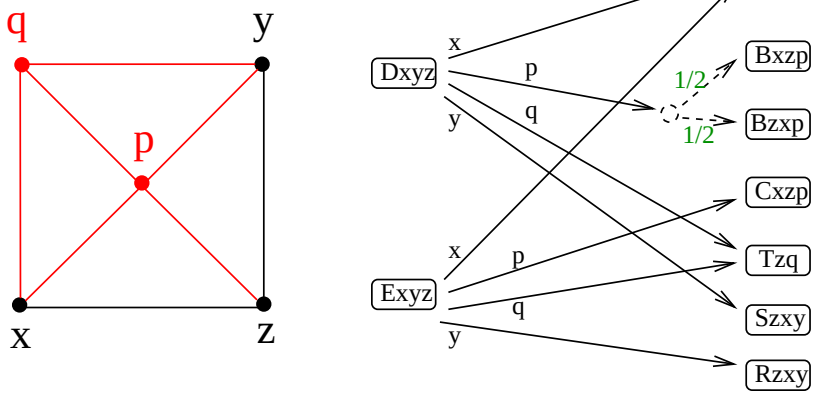




Narrative for Slide 26:

In this slide, we give the probabilities of choosing each possible subsequent for all moves starting from knowledge states A, B, and C. We can assume that x, y, z lie in the 7-point space illustrated on the left (where distance is measured by path length), and that $r \in \{x, y, p, q, u, v\}$.

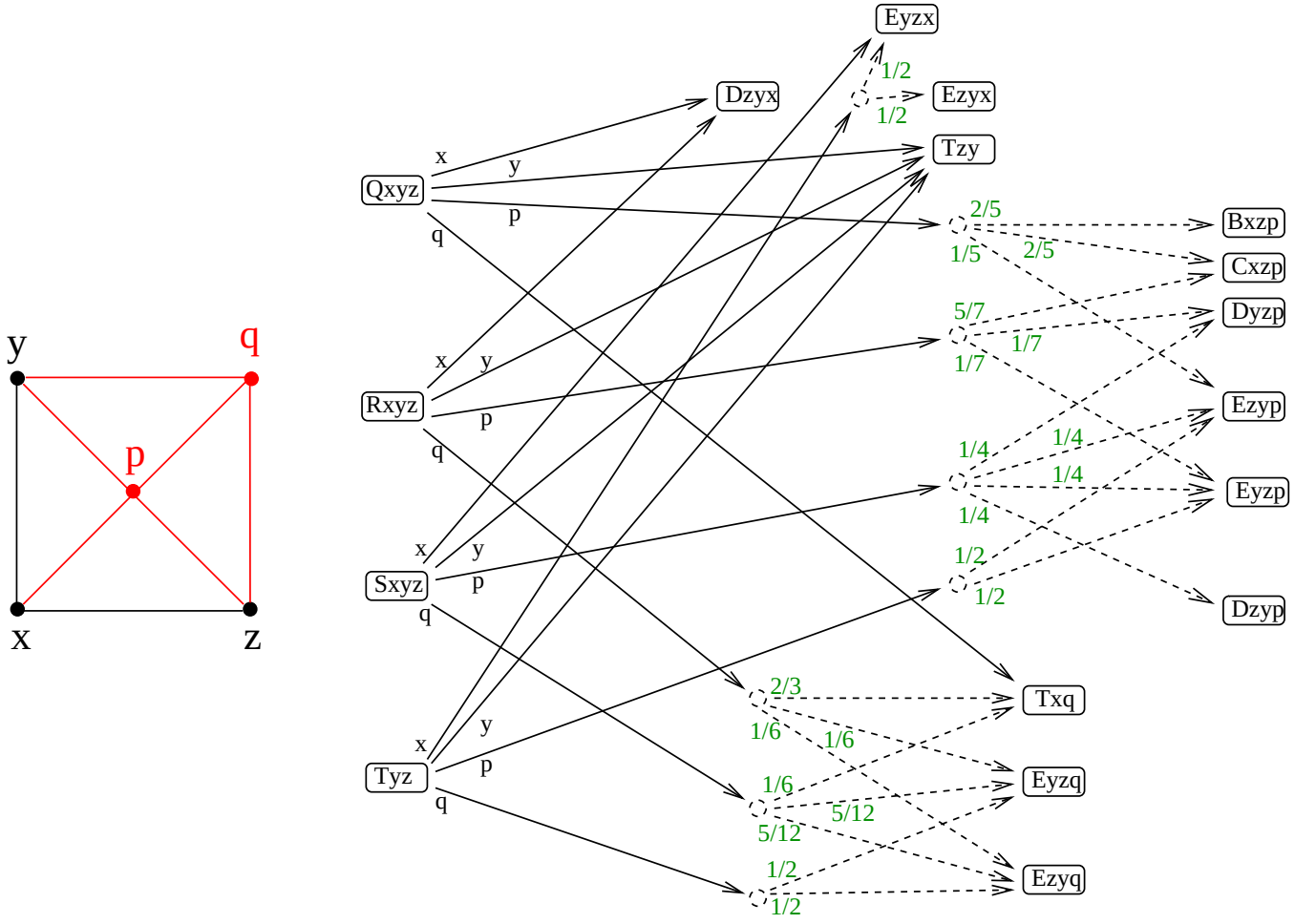
All Moves from Knowledge States D, E



Narrative for Slide 27:

In this slide, similar to the previous one, we give the probabilities of choosing each possible subsequent for all moves starting from knowledge states D and E.

All Moves from Knowledge States Q, R, S, T



Narrative for Slide 28:

In this slide, similar to the previous one, we give the probabilities of choosing each possible subsequent for all moves starting from knowledge states Q, R, S, and T.

Where do we go from Here?

One possible attack:

- Recall: Lund and Reingold found the optimal competitiveness for each 3-point metric space. (Always $C < \frac{e}{e-1}$.)
- Try to find $C < 2$ such that there is a C-competitive order 2 knowledge state algorithm for every 4-point metric space. Why? Because for an order 2 knowledge state algorithm, every move involves at most 4 points.
- Try to fit them together.