Explicit Expanding Expanders as Datacenter Topologies

Michael Dinitz
Johns Hopkins University

Based on joint work with Michael Schapira, Gal Shahaf, and Asaf Valadarsky
(Hebrew University of Jerusalem)

Outline

- Question: how should we wire datacenters?
- Expanders!

1. Background on expanders as networks
2. Can we build expanders with additional properties to ease adoption (incremental expansion)?
3. Can other approaches (degree-diameter graphs) be viewed as just other expanders?

Expander Graphs as Network Topologies

Datacenter Topologies

- What is the "**right**" topology? Many competing proposals!
- Surprising result [Jellyfish: Singla et al, NSDI '12]:
  - Random graphs outperform all of them, on almost every metric!
  - And have other nice properties (**incremental expansion**)
  - Practical?
- Can we get the benefits of random graphs without randomness?
  - Why are random graphs good? They're expanders!

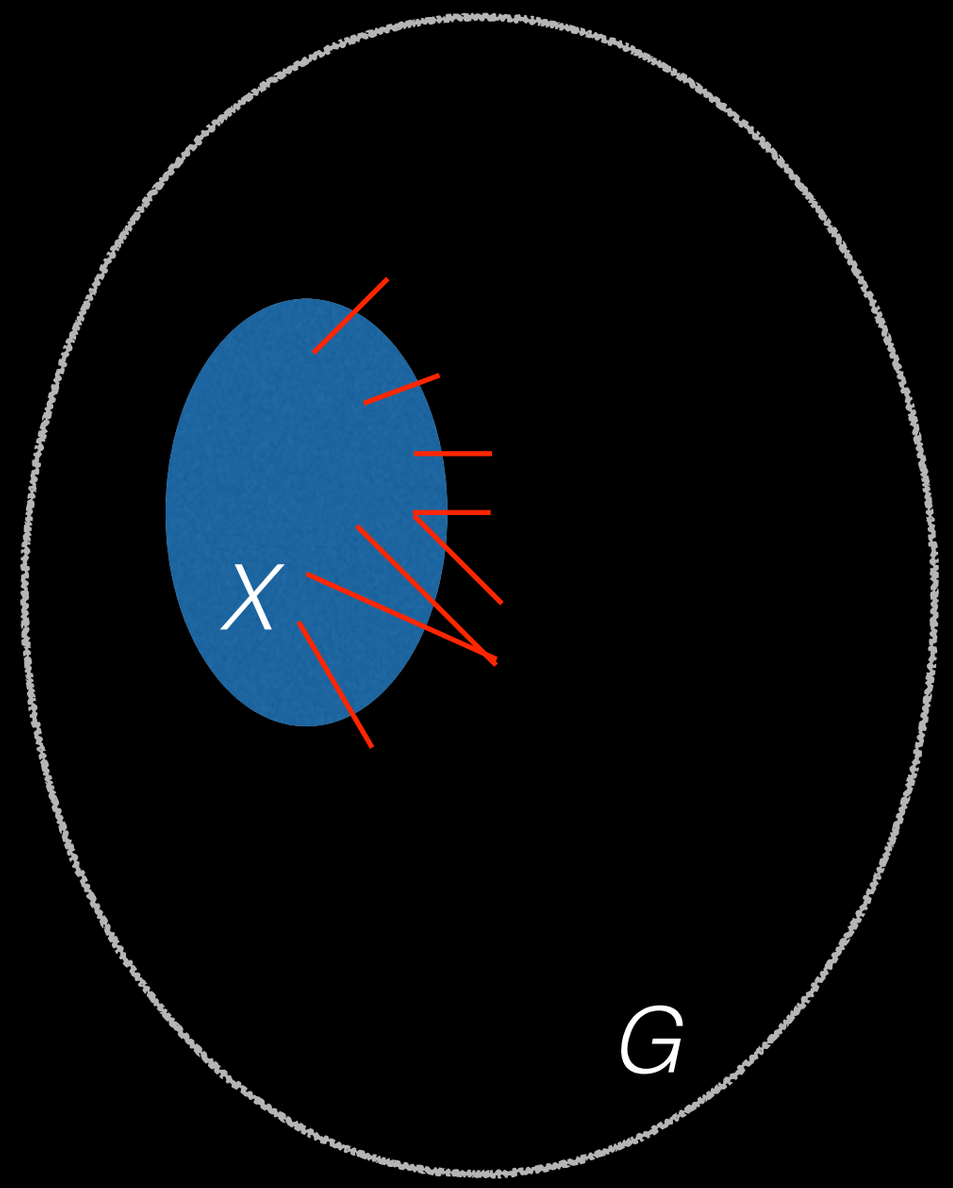
Expander Graphs

Expander Graphs

- Expanders: never get “trapped” in a subset of vertices

Expander Graphs

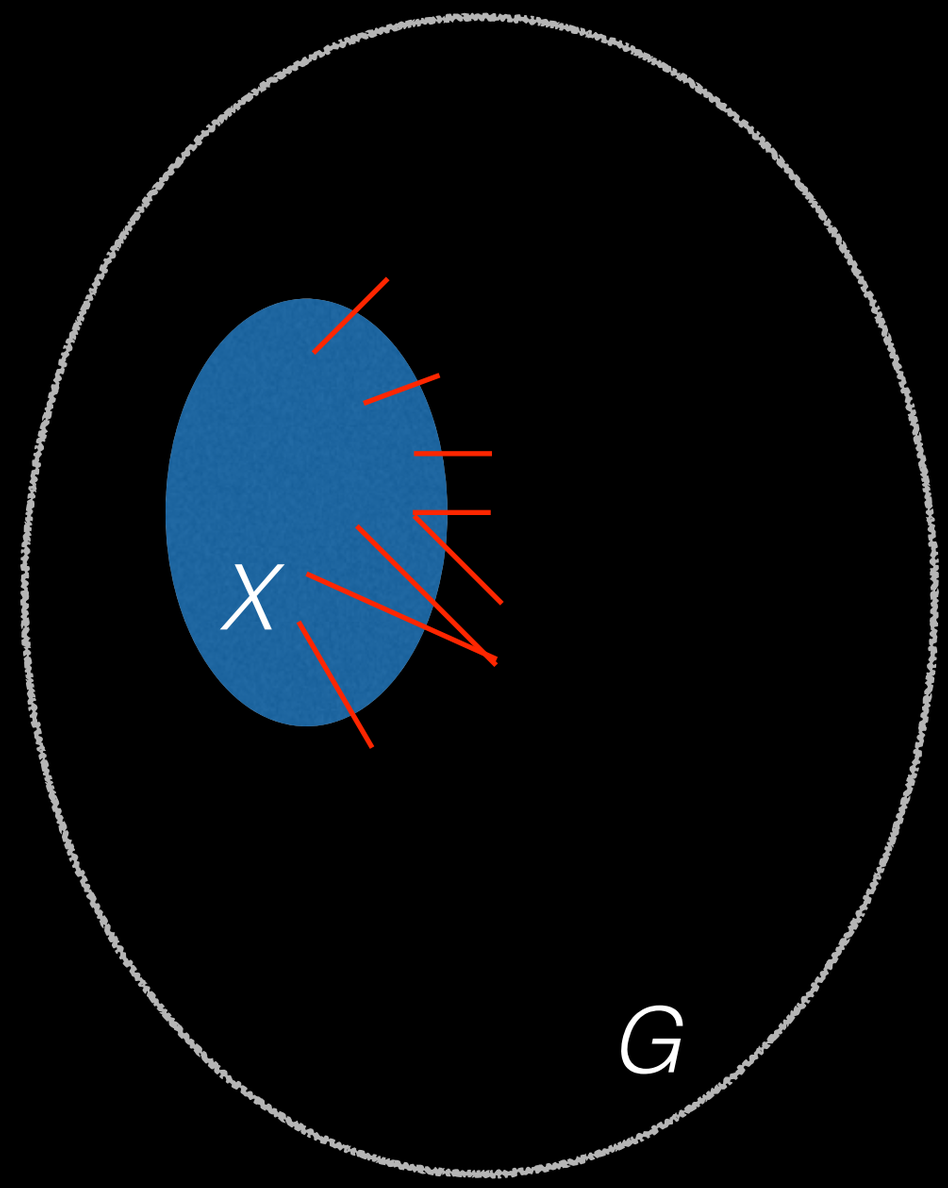
- Expanders: never get “trapped” in a subset of vertices



Expander Graphs

- Expanders: never get “trapped” in a subset of vertices
- Edge expansion:

$$h(G) = \min_{X \subset V : |X| \leq n/2} \frac{|E(X, V \setminus X)|}{|X|}$$

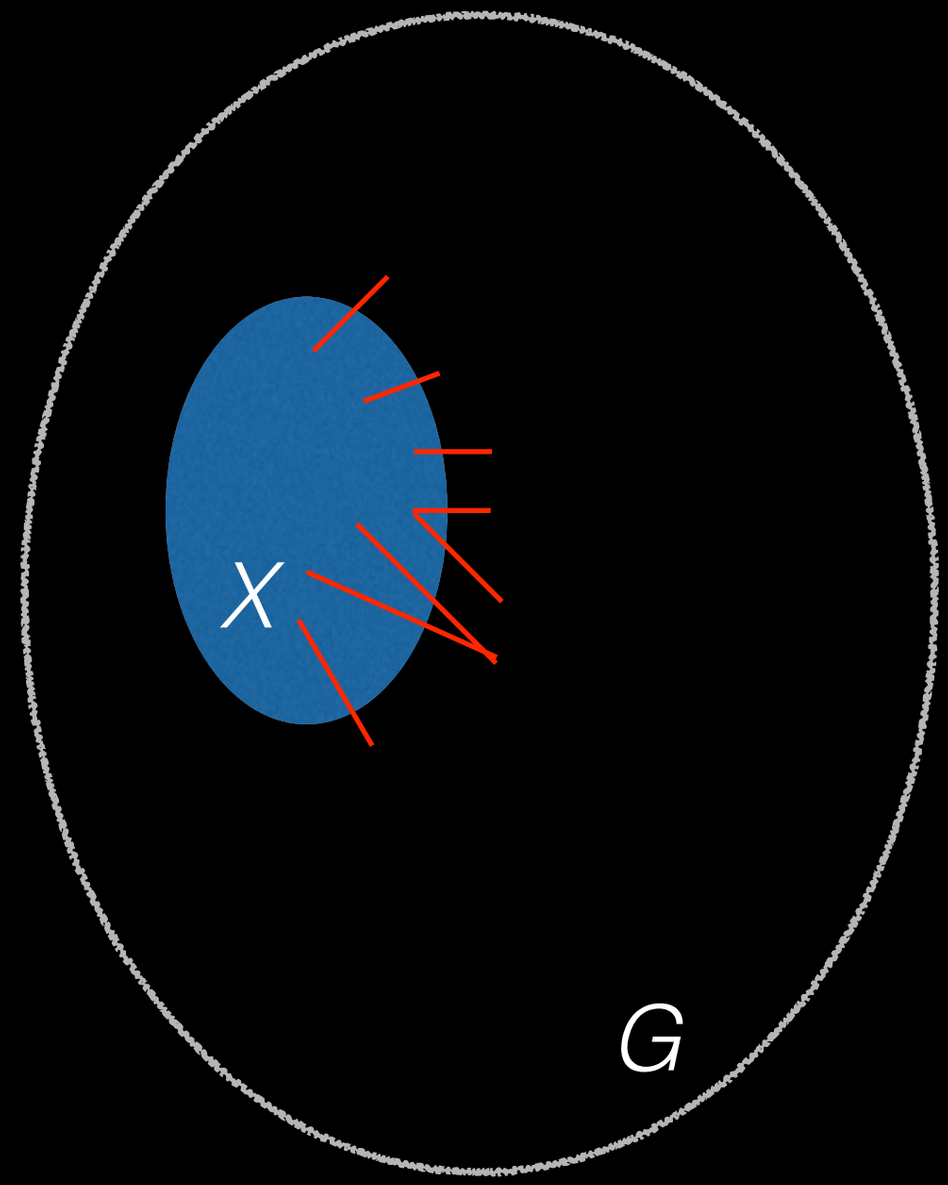


Expander Graphs

- Expanders: never get “trapped” in a subset of vertices
- Edge expansion:

$$h(G) = \min_{X \subset V : |X| \leq n/2} \frac{|E(X, V \setminus X)|}{|X|}$$

- Expander: *d*-regular graph with expansion *Ω(d)*

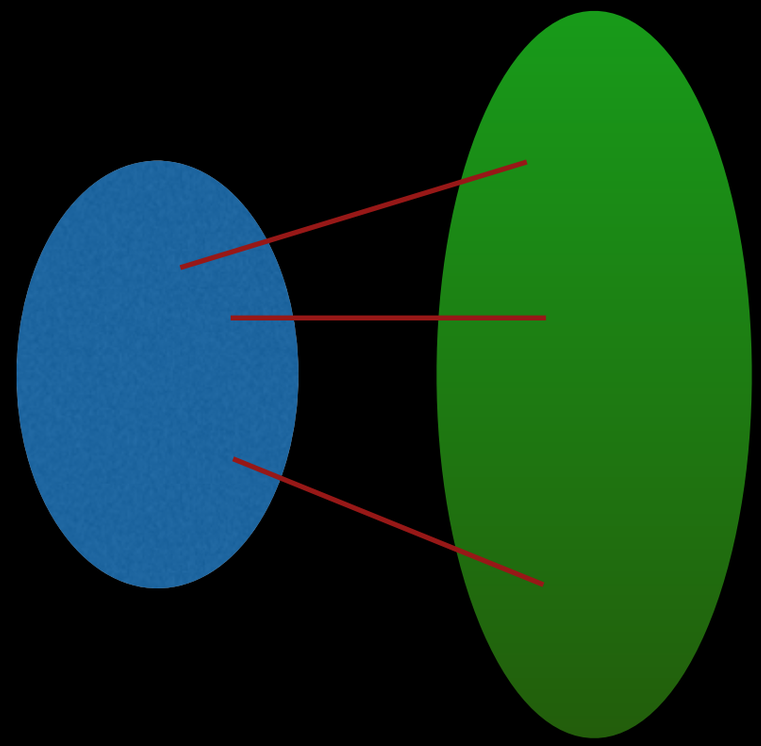


Expanders: History

- Widely studied in graph theory / theoretical CS
- Many, many applications (mostly complexity theory)
- Random graphs are (w.h.p) very good expanders
- Surprisingly difficult to construct expanders deterministically

Data Centers

- Lots of traffic between nodes
- In a bad topology, might get “stuck”
- Problem if lots of traffic from one section to the rest, not much capacity
- Lots of traffic everywhere, so traffic proportional to # vertices
- Really: want large (edge) expansion!
- Regular graph (# ports at switches)



Throughput

- Given graph *G* and traffic demand matrix *T*, throughput is amount we need to scale down all demands to make feasible
  - Max concurrent flow

- Important special case: *T* is all 1's (all-to-all traffic)

- Not the only metric for network quality, but an important one

Throughput: Theory

- Thm: If *T* is all-to-all, then ***any*** expander has throughput within *O(log d)* of the best possible *d*-regular graph.

- Thm: For any *T*, ***any*** expander has throughput within *O(log n)* of the best possible *d*-regular graph (for that *T*).

- Thm: For any *d*-regular graph *G*, there is some *T* and *d*-regular graph *G** such that *G** has throughput *Ω(log n)* more than *G*.

Throughput: Theory

- Thm: If *T* is all-to-all, then ***any*** expander has throughput within *O(log d)* of the best possible *d*-regular graph.
- Thm: For any *T*, ***any*** expander has throughput within *O(log n)* of the best possible *d*-regular graph (for that *T*).

- Thm: For any *d*-regular graph *G*, there is some *T* and *d*-regular graph *G** such that *G** has throughput *Ω(log n)* more than *G*.

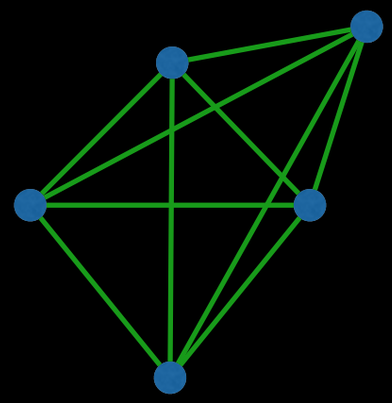
Incremental Expansion

Explicit Expanding Expanders.
Michael Dinitz, Michael Schapira, Asaf Valadarsky. ESA ‘15

Incremental Expansion

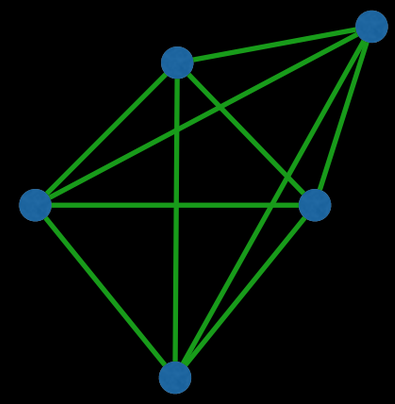
- So let's use expanders for our data centers!
- Data centers grow regularly: more servers and racks purchased and added
- Don't want to completely rewire network every time!
- Expander on *n* nodes should have approximately same edge set as expander on *n+1* nodes

Random
[Jellyfish NSDI'12]



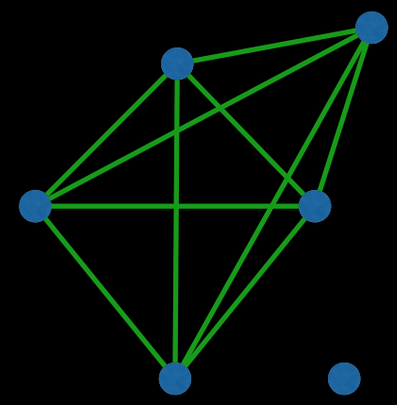
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



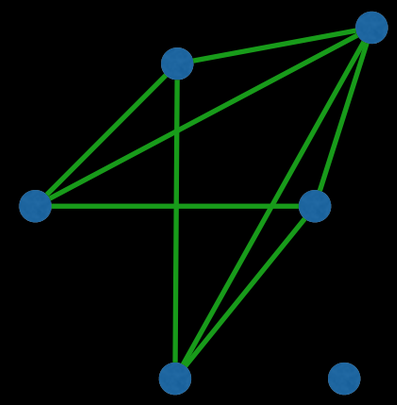
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



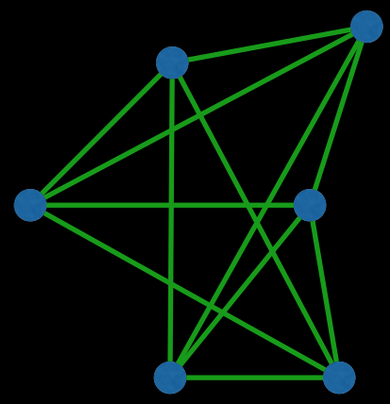
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



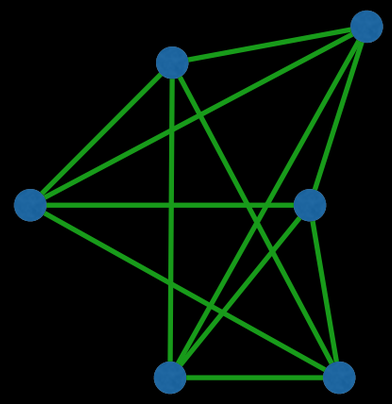
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



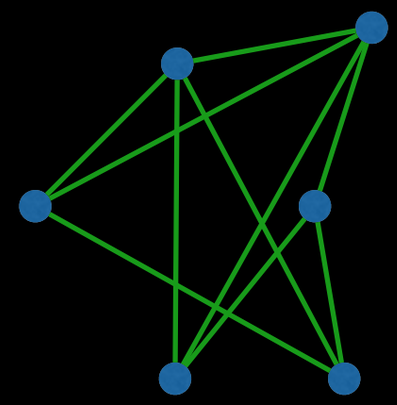
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



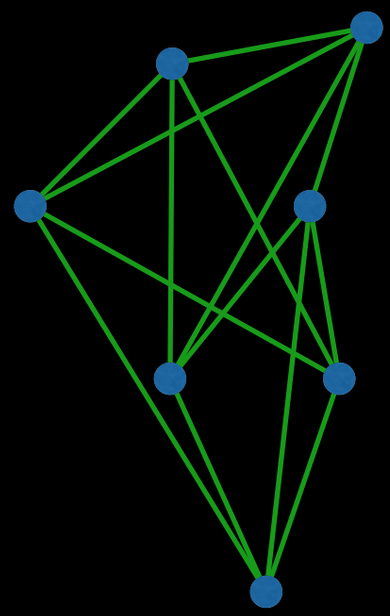
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



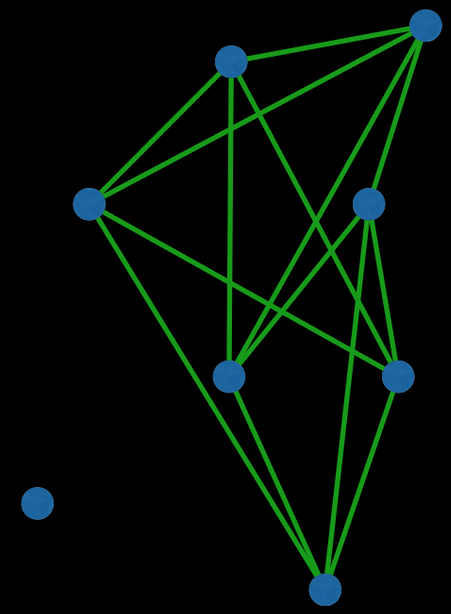
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



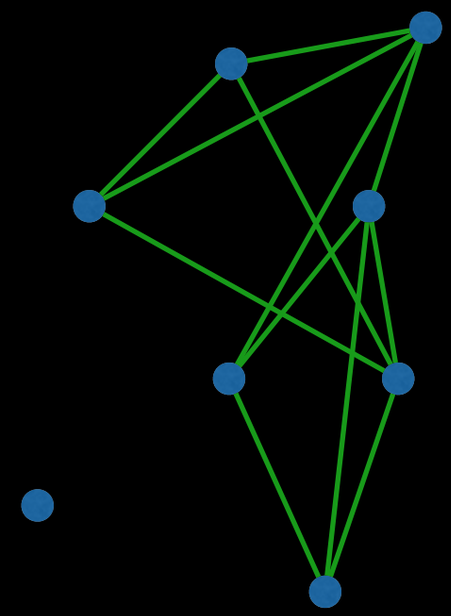
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



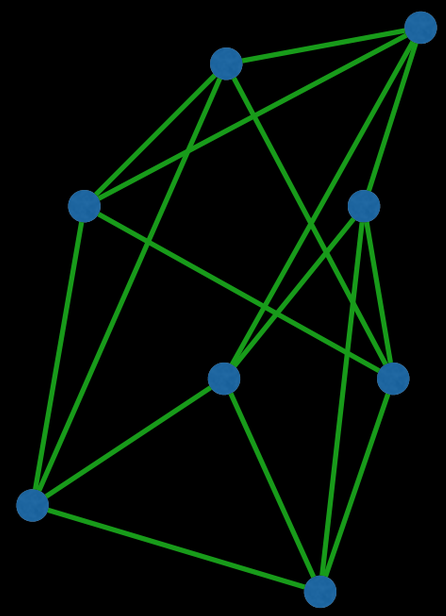
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



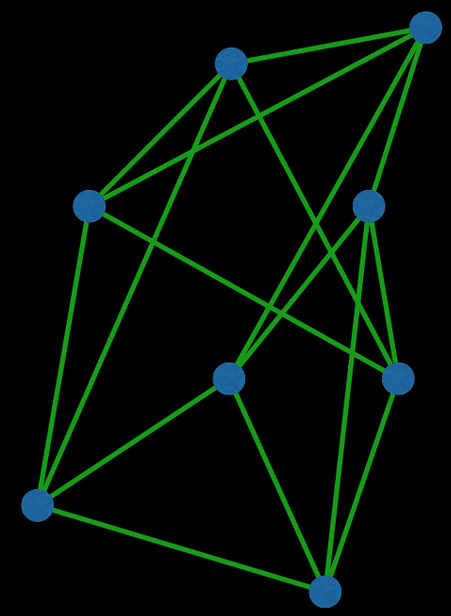
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node



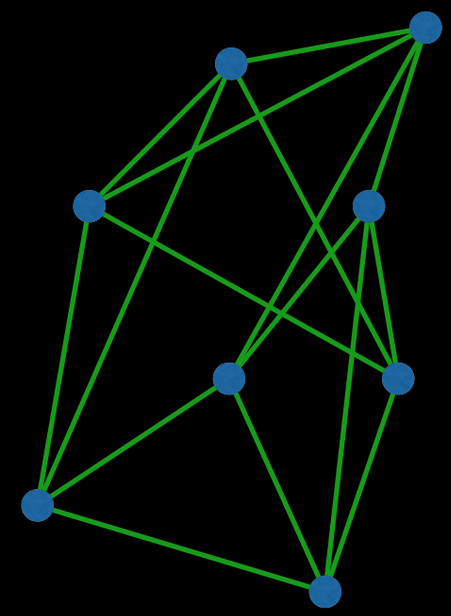
Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node
- Works great in simulation — only theory for uniform random regular graphs (Bollobas)



Random [Jellyfish NSDI'12]

- Construct expanders randomly
- To add node: choose random d/2 matching, remove, connect to new node
- Works great in simulation — only theory for uniform random regular graphs (Bollobas)



- Will companies actually use random datacenters?
- Can we get same guarantees with deterministic constructions?

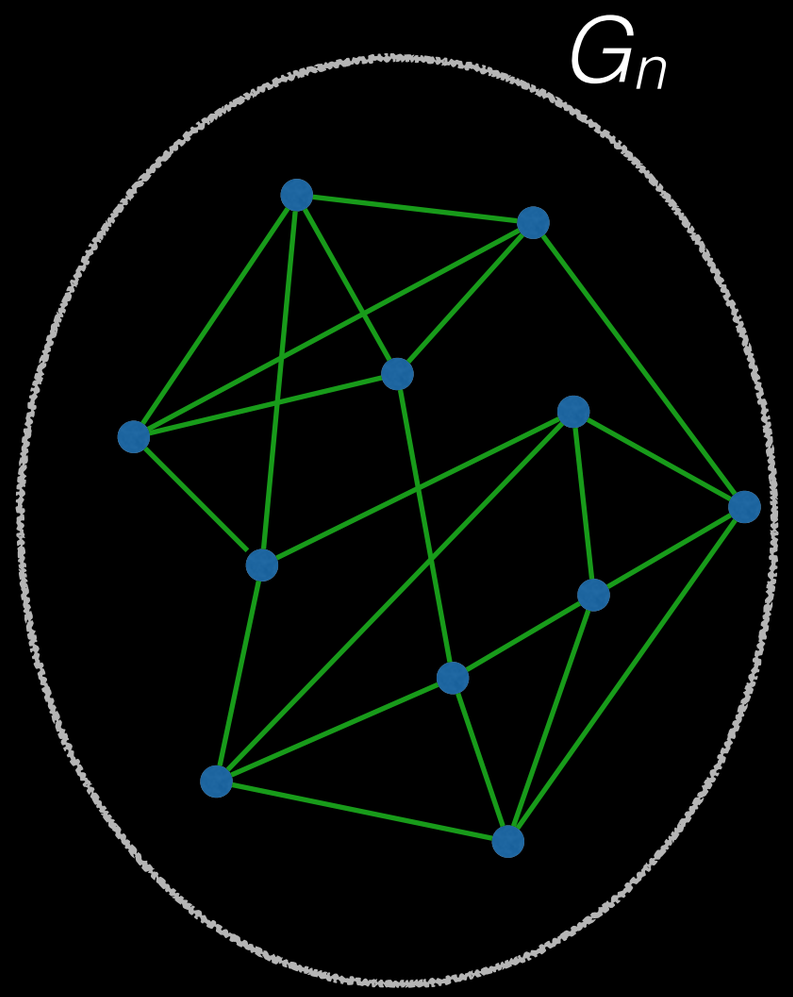
Expanding Expanders

Expanding Expanders

- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**

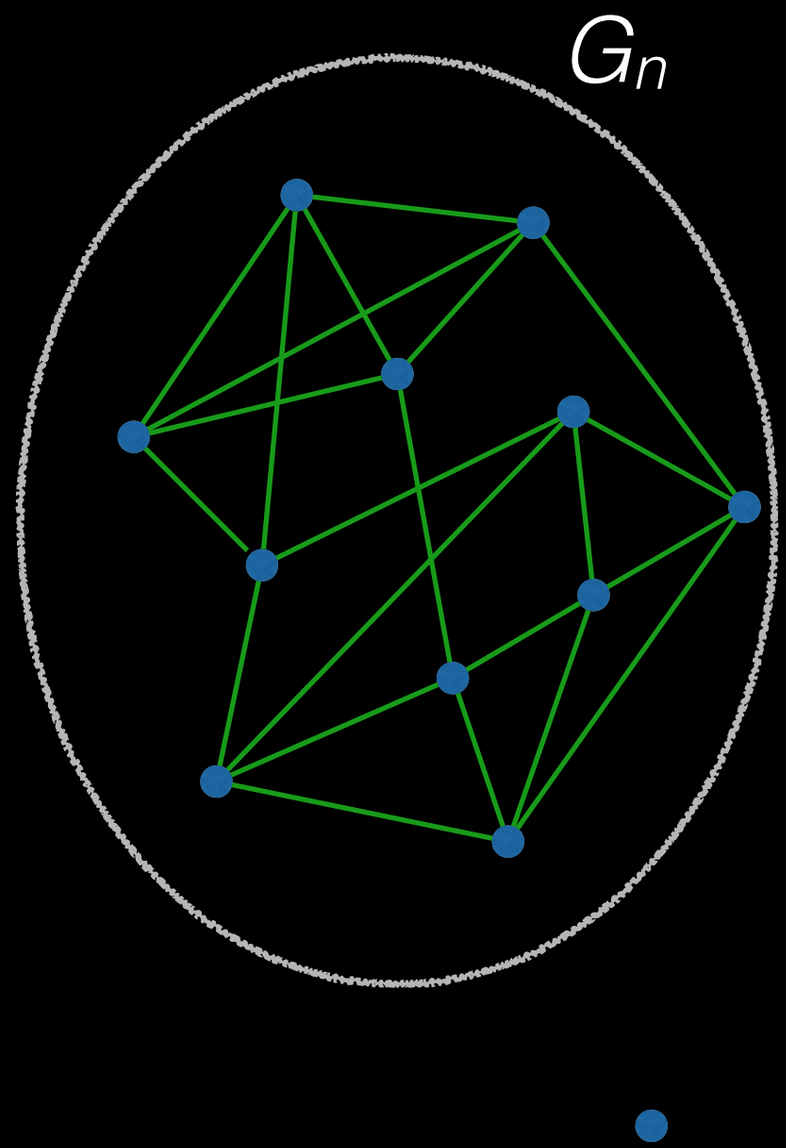
Expanding Expanders

- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**



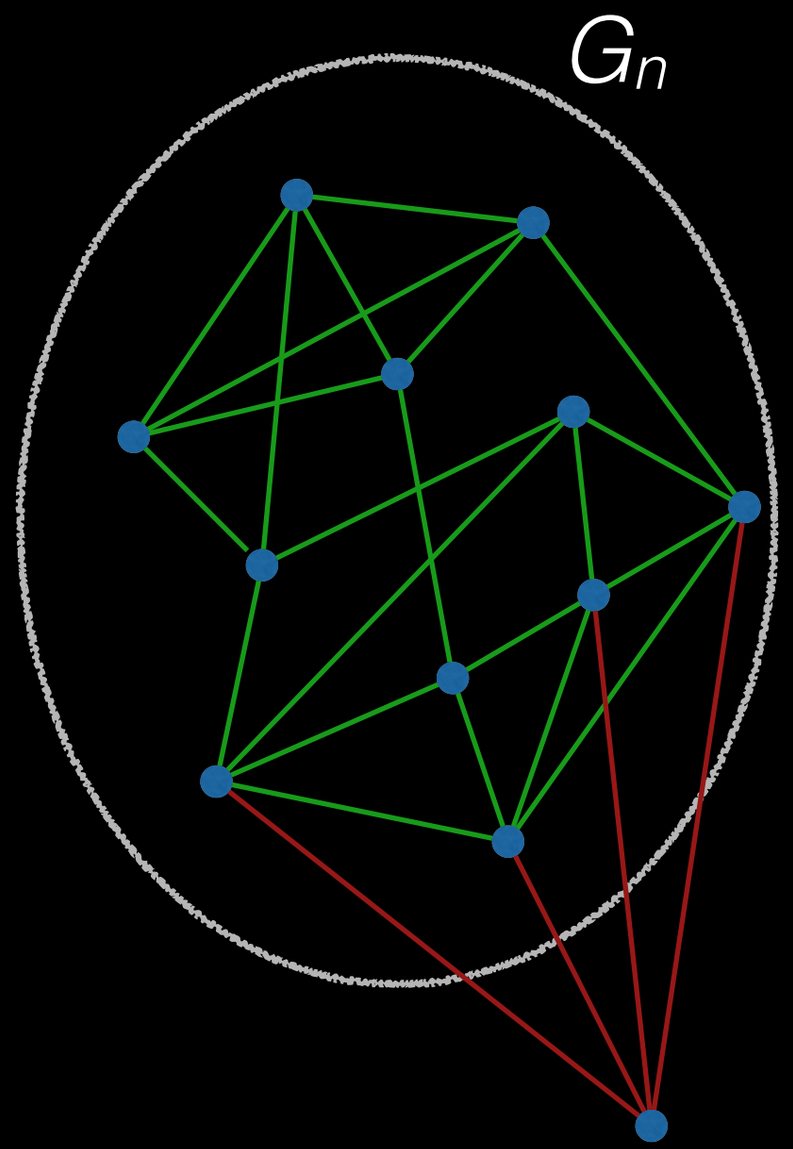
Expanding Expanders

- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**



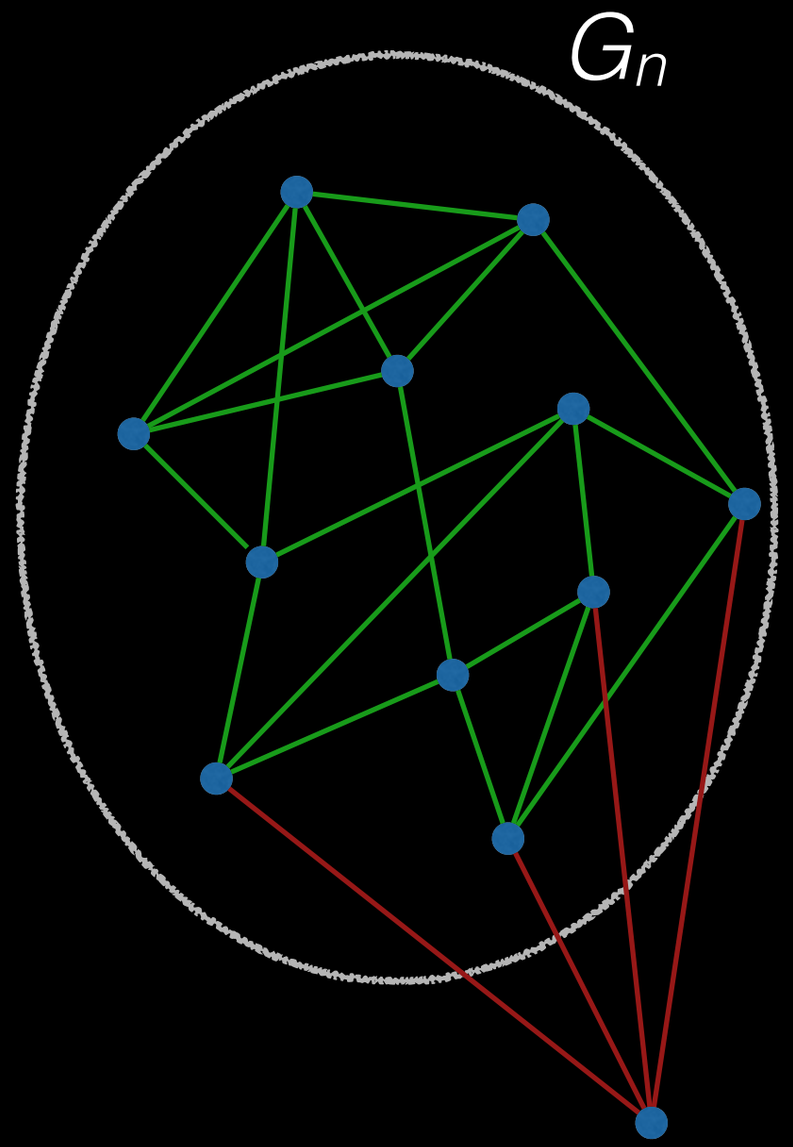
Expanding Expanders

- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**



Expanding Expanders

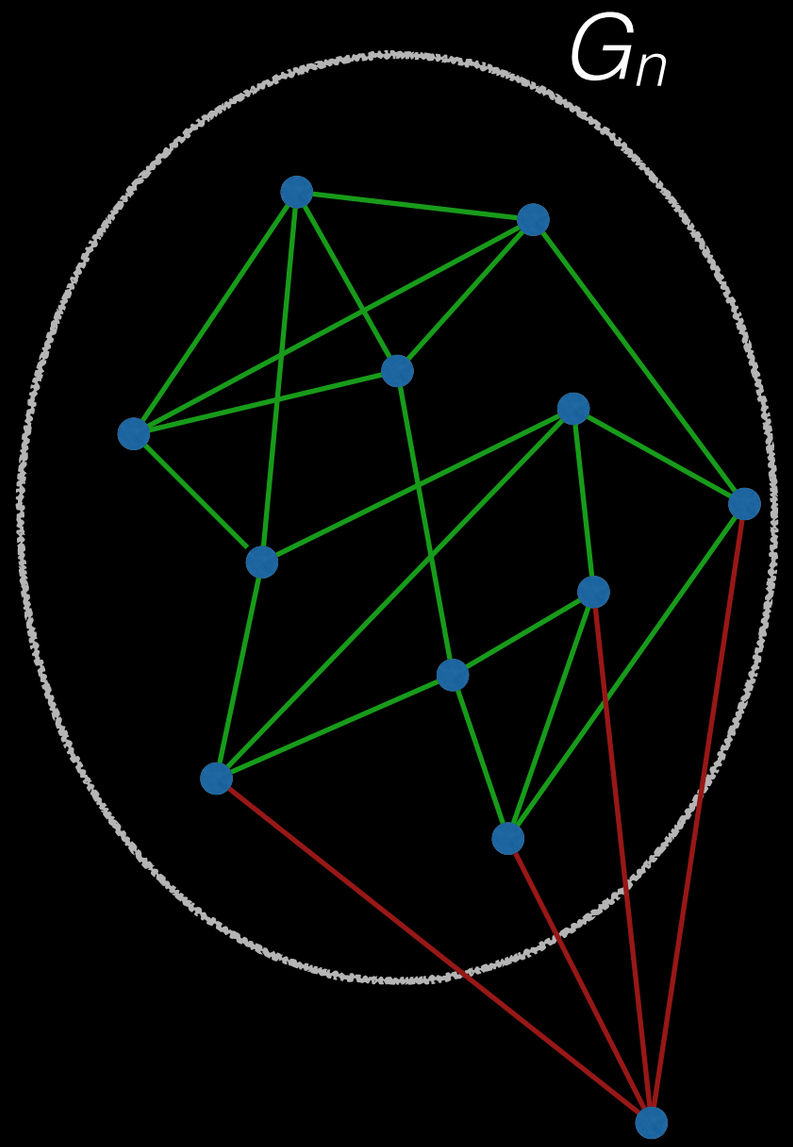
- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**



Expanding Expanders

- Two problems with using existing deterministic expanders as data centers
    1. Need to exist for all *n* (not just primes, powers of 2, etc.)
    2. Need to handle **incremental expansion**

- Goal: infinite series of *d*-regular graphs $G_{d+1}, G_{d+2}, G_{d+3}, \ldots$ where:
    1. G_i has *i* nodes
    2. Each G_i has large edge expansion (approx. *d/2*)
    3. Few edge changes to get from G_i to G_{i+1} (approx. *3d/2*)



Explicit Expanding Expanders

Main Result: graphs G_i where:

- G_i has i nodes
- Expansion approx. $d/3$
- At most $5d/2$ edge changes from G_i to G_{i+1}

Explicit Expanding Expanders

Main Result: graphs G_i where:

- G_i has i nodes
- Expansion approx. $d/3$
- At most $5d/2$ edge changes from G_i to G_{i+1}

- Still room for improvement!
- Technicality: use multiple edges / edge weights

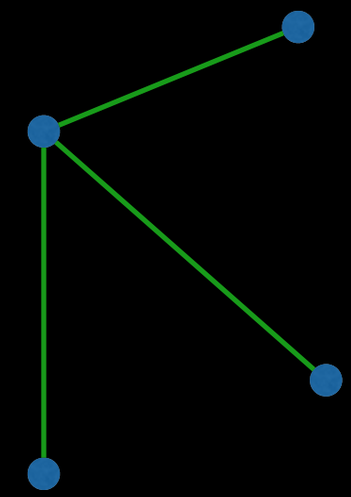
2-Lifts

2-Lifts

- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of G: double every vertex, replace edge by matching

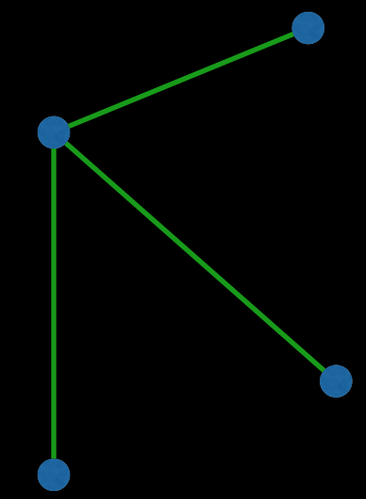
2-Lifts

- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of *G*: double every vertex, replace edge by matching



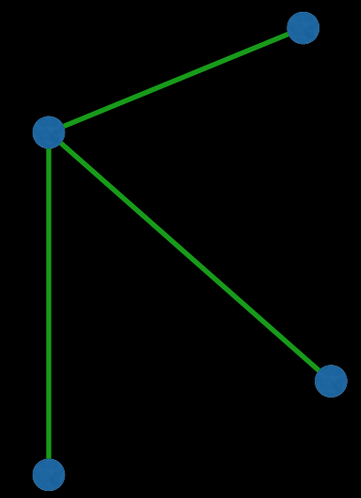
2-Lifts

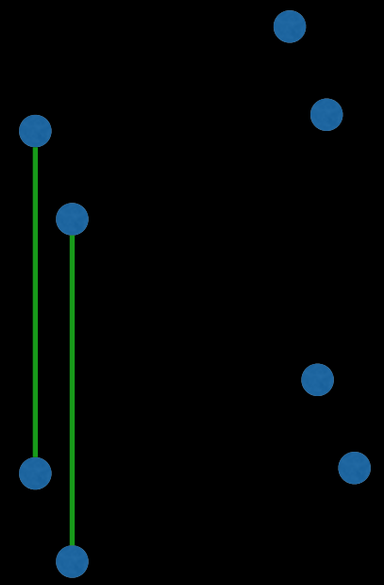
- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of *G*: double every vertex, replace edge by matching



2-Lifts

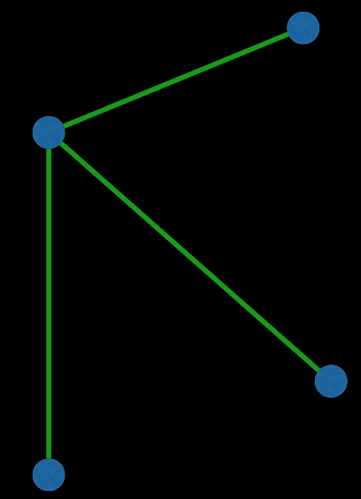
- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of *G*: double every vertex, replace edge by matching

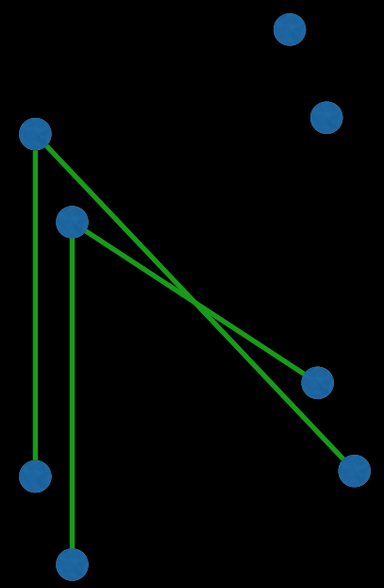




2-Lifts

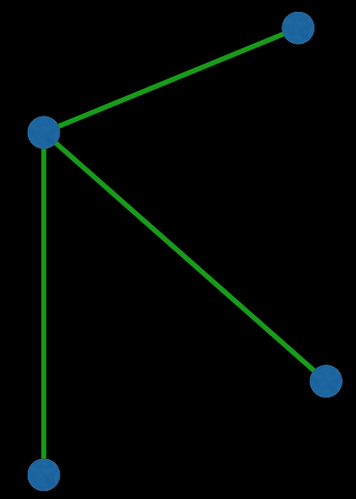
- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of *G*: double every vertex, replace edge by matching

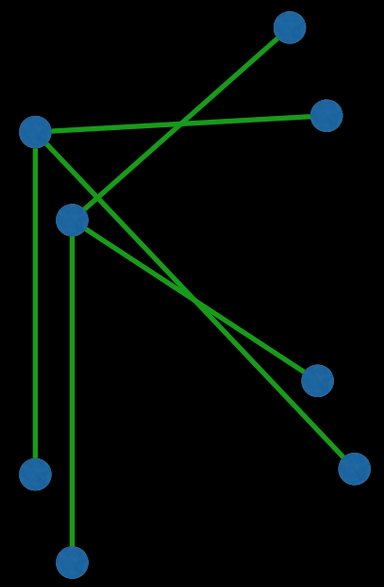




2-Lifts

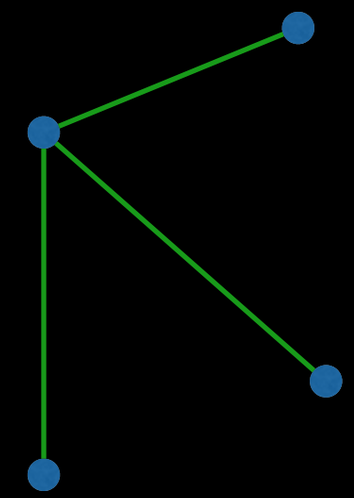
- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of *G*: double every vertex, replace edge by matching

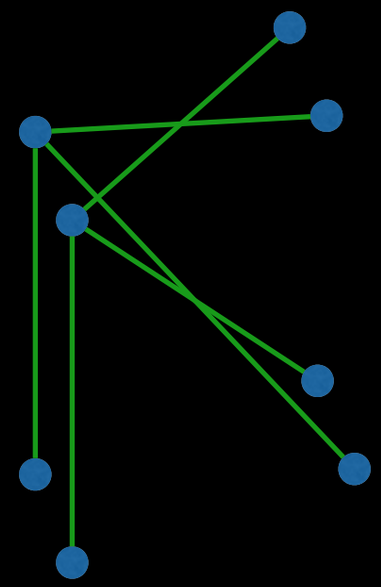




2-Lifts

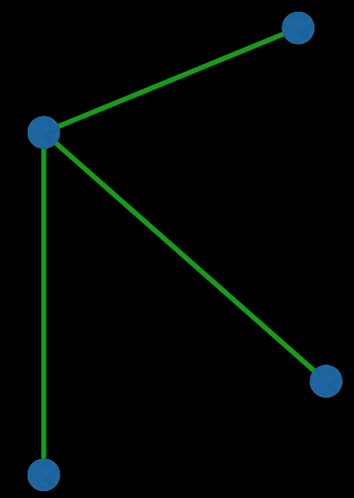
- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of G: double every vertex, replace edge by matching
- Two options for each matching, so $2^{|E|}$ possible 2-lifts

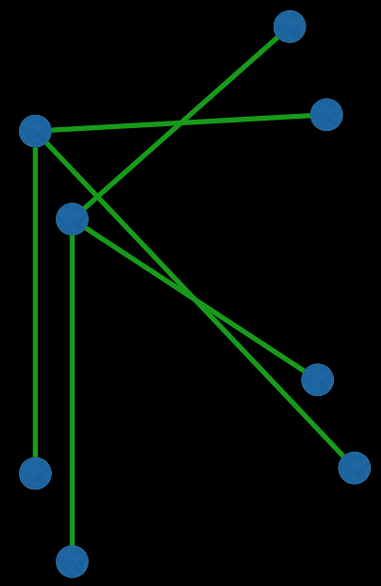




2-Lifts

- Main tool: 2-Lifts [Bilu-Linial]
- 2-Lift of G: double every vertex, replace edge by matching
- Two options for each matching, so $2^{|E|}$ possible 2-lifts
- Thm [BL]: If G an expander, random matchings gives good expander w.h.p.
- Can be derandomized!





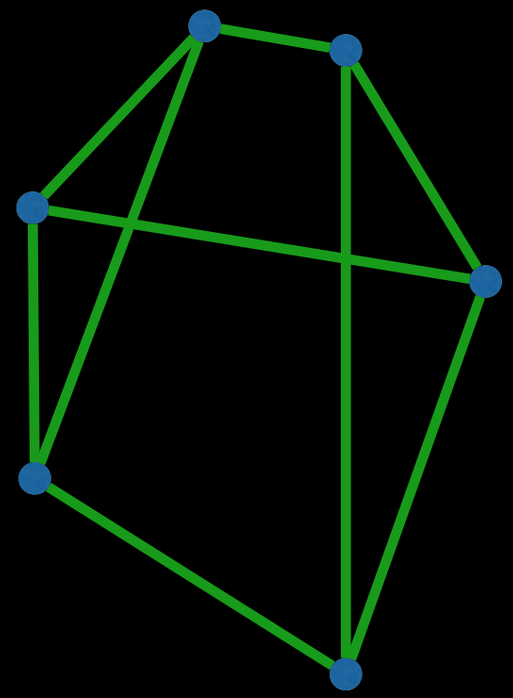
Our Approach

Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*

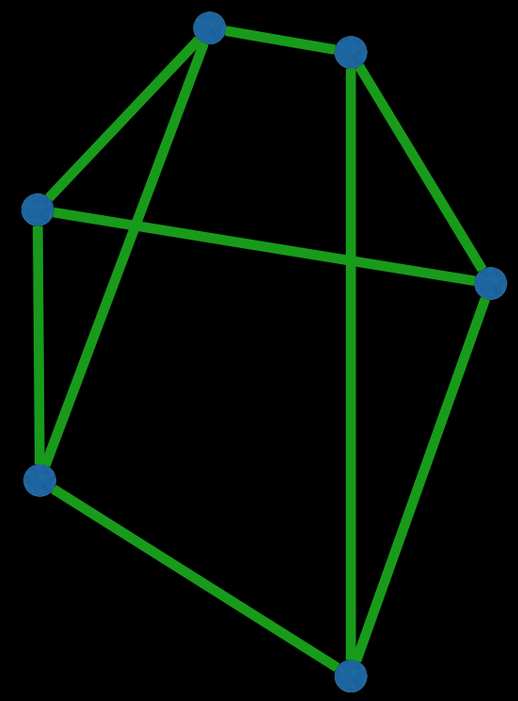
Our Approach

- “Split” each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*



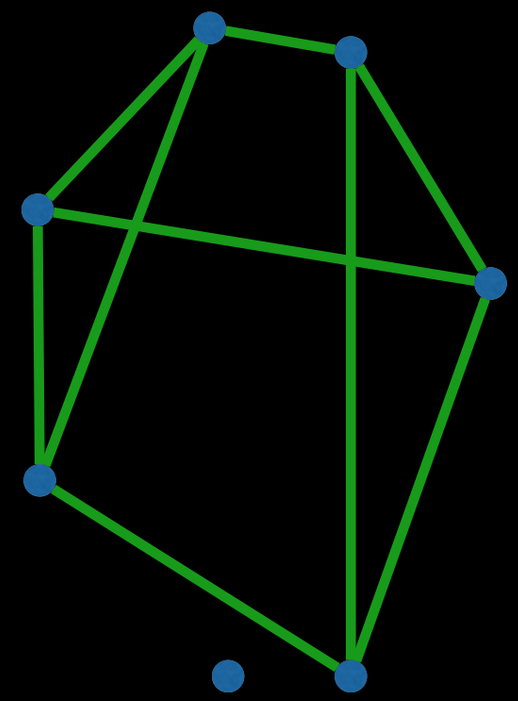
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node



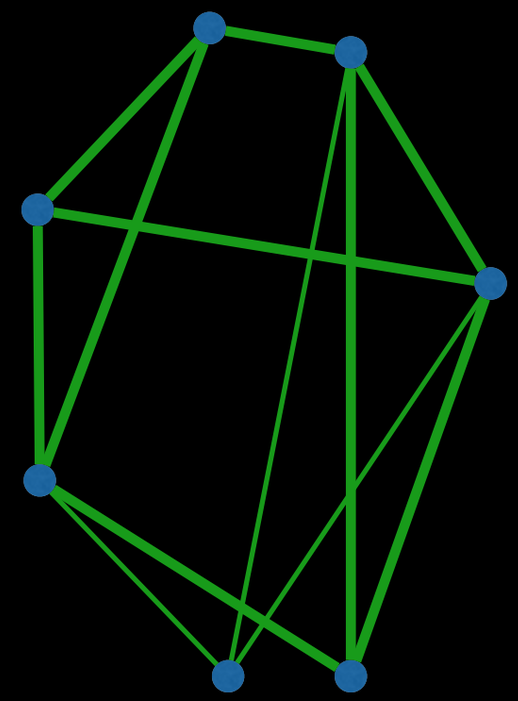
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node



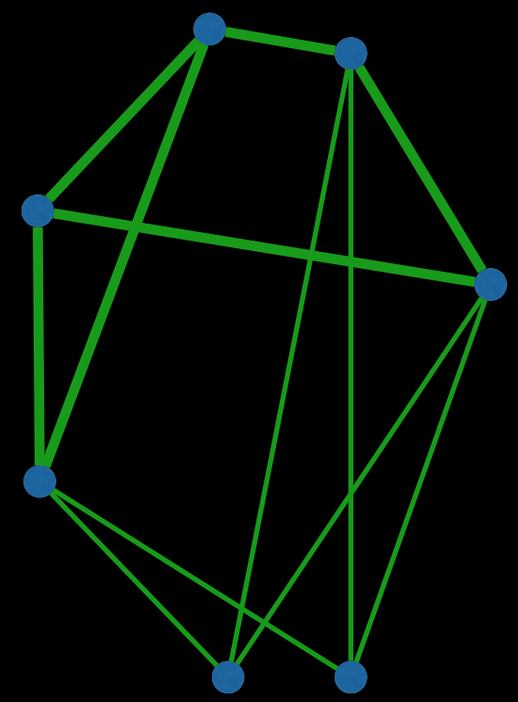
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node



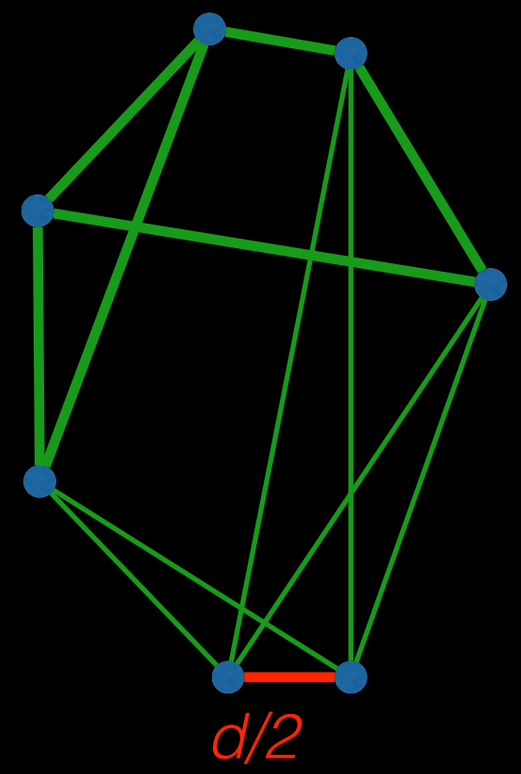
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node



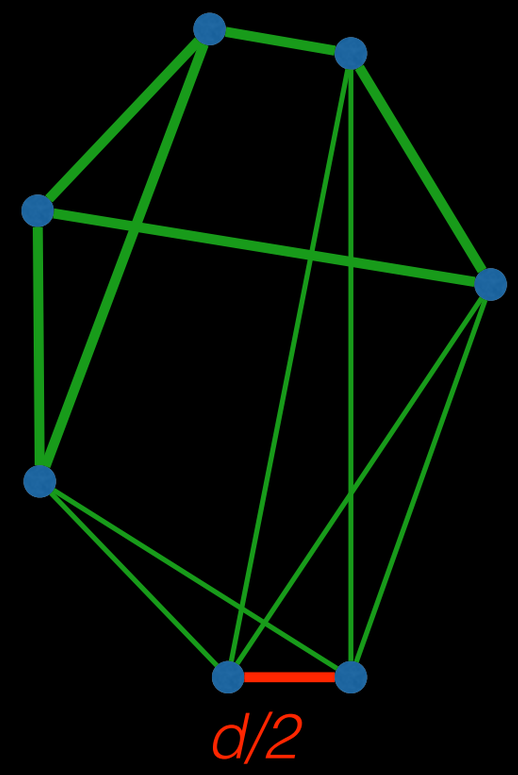
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node



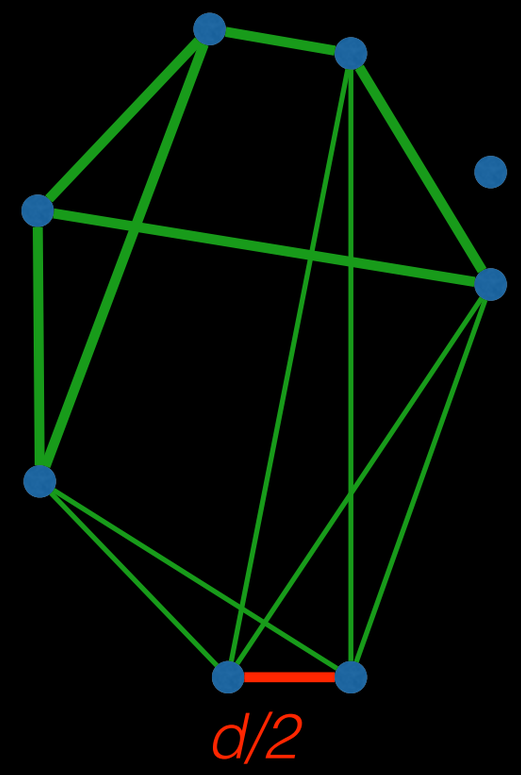
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges



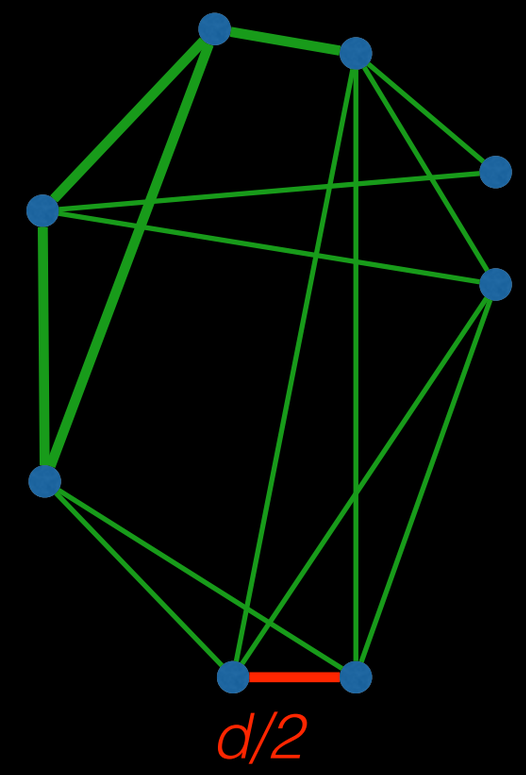
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges



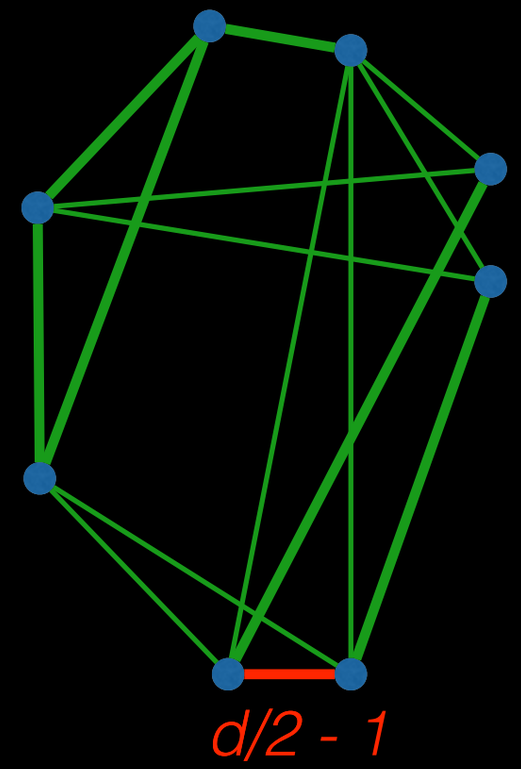
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
- Unsplit neighbors: replace weight *2* edge with two weight *1* edges



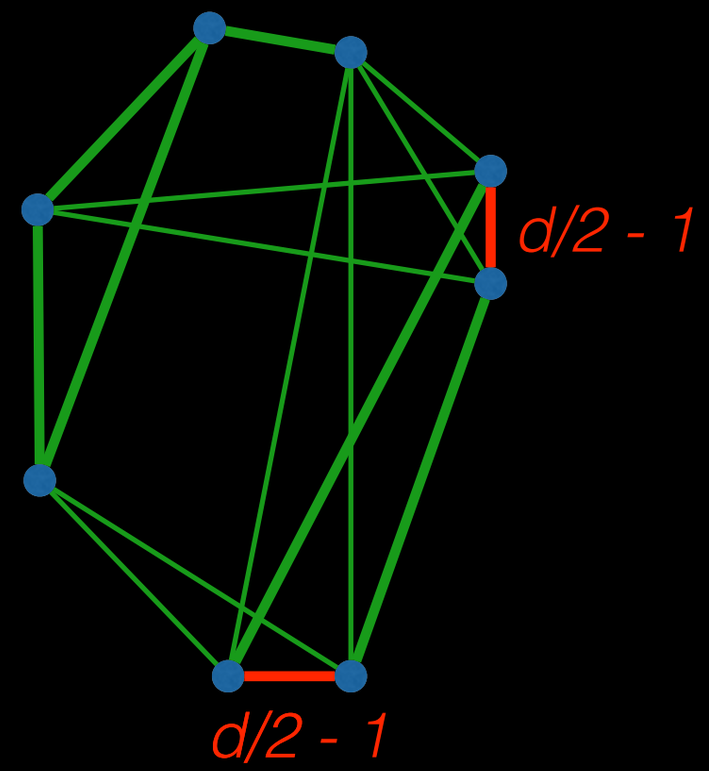
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges



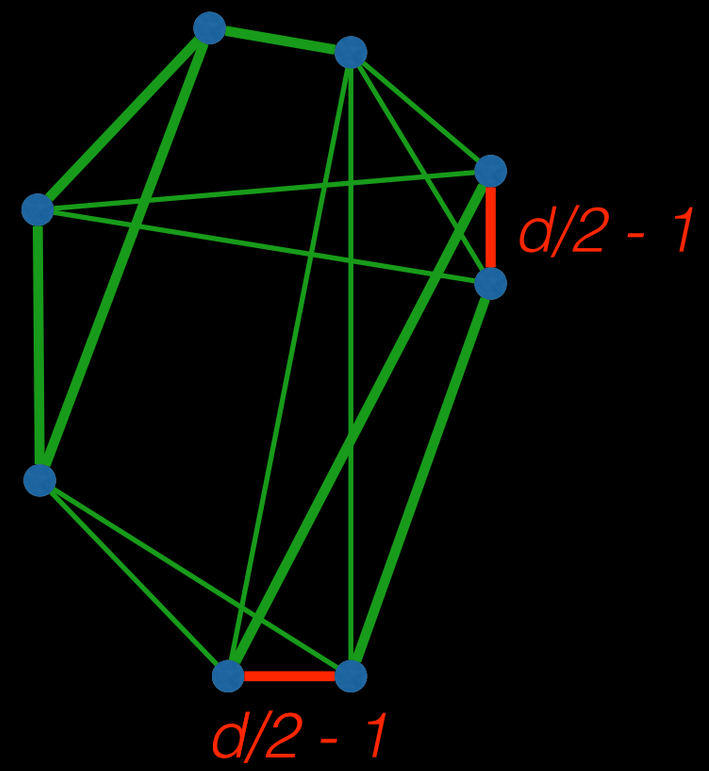
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges



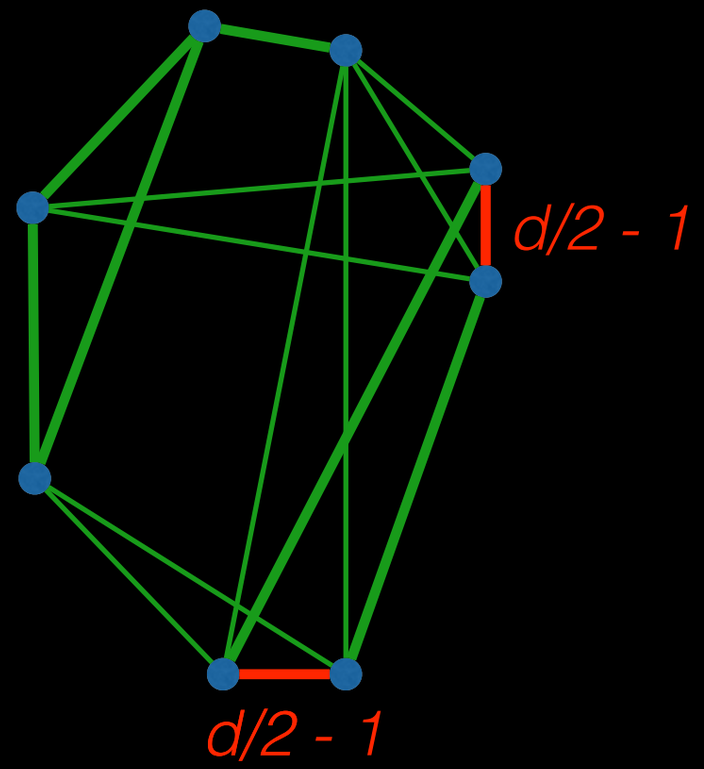
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges
  - Split neighbors: replace two weight *1* edges with matching of weight *2* edges



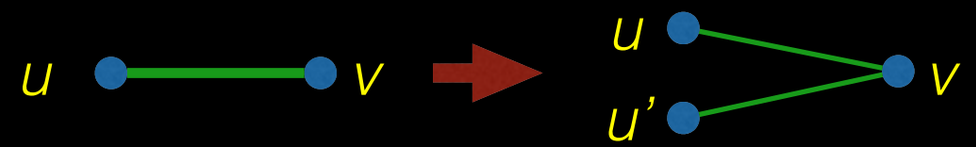
Our Approach

- "Split" each node one at a time, rather than all at once
- Start with *d/2*-regular expander, weights *2*
- Inserting new node: split currently unsplit node
  - Unsplit neighbors: replace weight *2* edge with two weight *1* edges
  - Split neighbors: replace two weight *1* edges with matching of weight *2* edges
- Nice property: after all nodes split, have precisely next BL expander

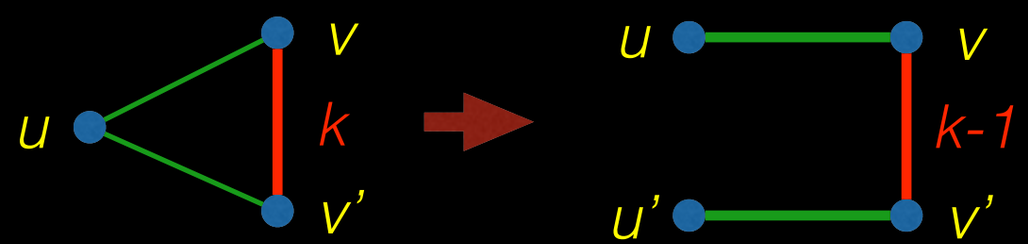


Analysis: Edge Changes

- Split *u* into *u, u'*:
  - Unsplit neighbor *v*: one edge of weight *2* → *2* edges of weight *1*. Cost *2*



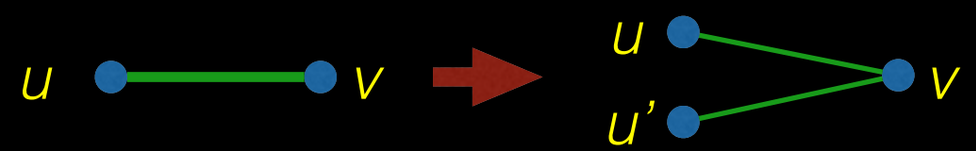
  - Split neighbors *v, v'*: two edges of weight 1 → two edges of weight 2, decrease *{v, v'}* by 1. Cost 5.

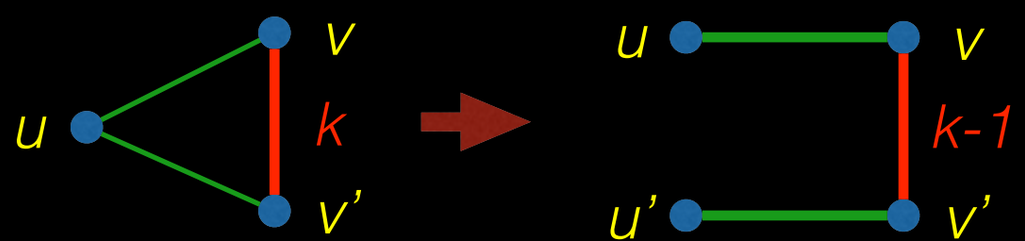


  - Add *{u,u'}* of weight *(# unsplit neighbors)*

Analysis: Edge Changes

- Split *u* into *u, u'*:
  - Unsplit neighbor *v*: one edge of weight *2* → *2* edges of weight *1*. Cost *2*
  - Split neighbors *v, v'*: two edges of weight 1 → two edges of weight 2, decrease *{v, v'}* by 1. Cost 5.
  - Add *{u,u'}* of weight *(# unsplit neighbors)*



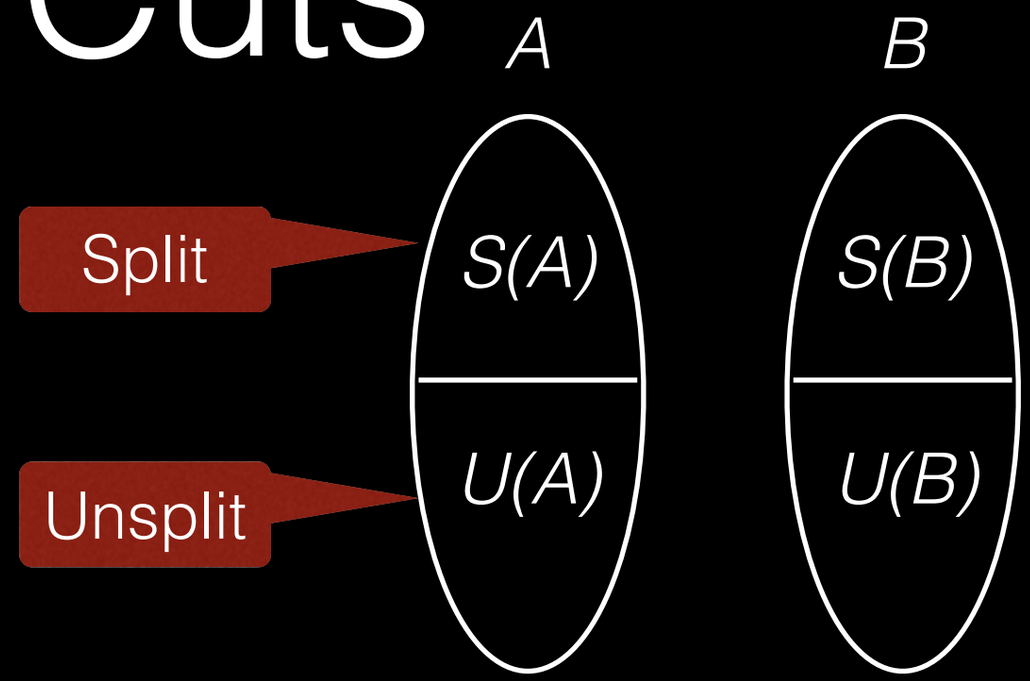


Total cost: *3*(unsplit neighbors) + 5*(split neighbors)*

Know *2*(unsplit) + 2*(split) = d*

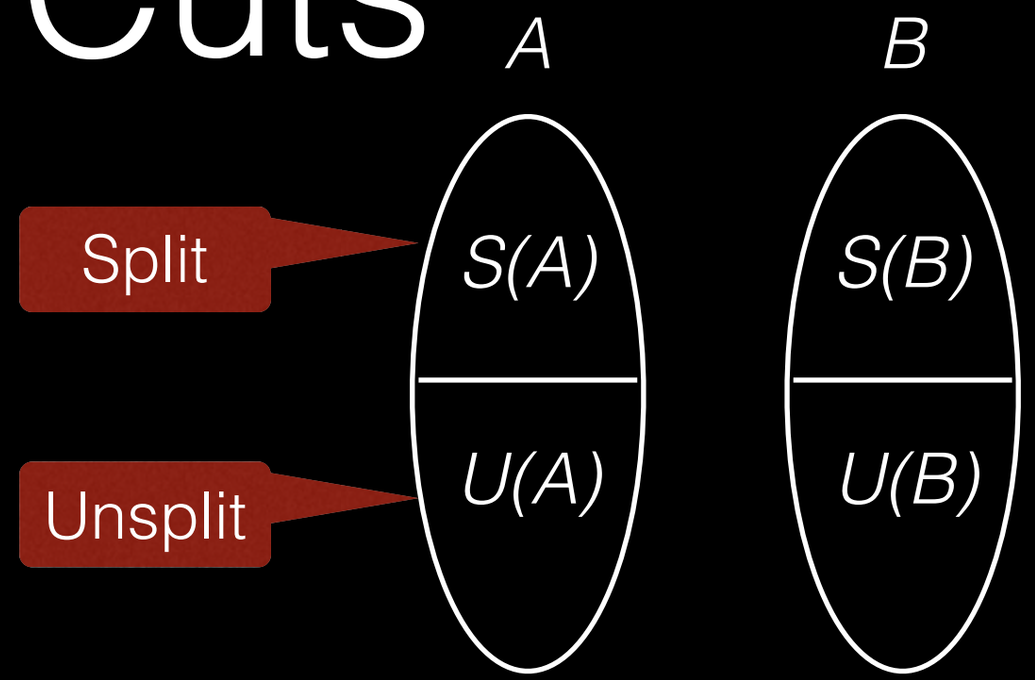
Future Cuts

- Cut *(A, B)*

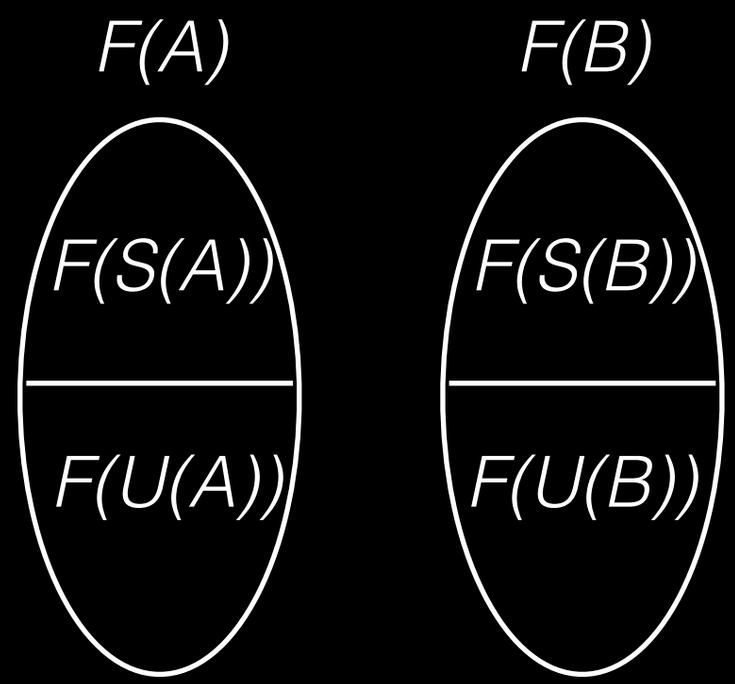


Future Cuts

- Cut *(A, B)*



- Future cut *(F(A), F(B))* of next BL expander:
  - *F(S(·)) = S(·)*
  - *F(U(·)) = U(·)* and splits of *U(·)*



Easy expansion

Easy expansion

- Know that $w(F(A), F(B))$ large ($\approx (d/2)\ |F(A)|$) — argue that $w(A, B)$ close to it

Easy expansion

- Know that *w(F(A), F(B))* large (*≈(d/2)* *|F(A)|*) — argue that *w(A, B)* close to it
- 2 * *w(U(A), U(B))* = *w(F(U(A), F(U(B)))*

Easy expansion

- Know that *w(F(A), F(B))* large (*≈(d/2)* *|F(A)|*) — argue that *w(A, B)* close to it
- 2 * *w(U(A), U(B))* = *w(F(U(A), F(U(B)))*
- *w(S(A), S(B))* = *w(F(S(A)), F(S(B)))*

Easy expansion

- Know that *w(F(A), F(B))* large (*≈(d/2)* *|F(A)|*) — argue that *w(A, B)* close to it
- 2 * *w(U(A), U(B))* = *w(F(U(A), F(U(B)))*
- *w(S(A), S(B))* = *w(F(S(A)), F(S(B)))*
- *2 * w(S(A), U(B))* = *w(F(S(A)), F(U(B)))*

Easy expansion

- Know that *w(F(A), F(B))* large (*≈(d/2)* *|F(A)|*) — argue that *w(A, B)* close to it
- 2 * *w(U(A), U(B))* = *w(F(U(A), F(U(B)))*
- *w(S(A), S(B))* = *w(F(S(A)), F(S(B)))*
- *2 * w(S(A), U(B))* = *w(F(S(A)), F(U(B)))*

Half the weight, so at least half the expansion (*d/4*)!

Real expansion

- Use fact that fewer vertices than in future cut

- Combine with Expander Mixing Lemma, get expansion *d/3*
  - Need to use strong spectral expansion
  - Fact: there are graphs in sequence with $\lambda_2 \approx d/2$, so cannot get expansion bound directly from Cheeger

Conclusion (Part II)

- Can get good incremental expansion by incrementally 2-lifting
  - Can even do this to **any** starting expander

- Many open questions
  - Tight or improved bounds?
  - Heterogeneous nodes? Edges?

Degree-Diameter Graphs as Expanders

Large Fixed-Diameter Graphs are Good Expanders.
Michael Dinitz, Michael Schapira, Gal Shahaf. arXiv ‘17

Different Intuitions

- Expanders
  - Good because data can't get "bottlenecked" anywhere
  - Ensure this by making cuts "large"
- Alternative
  - Long paths are wasteful: flow of size α uses (*length* $\times \alpha$) total capacity
  - So minimize distances: try to make diameter small

Degree-Diameter Graphs

Degree-Diameter Graphs

- Three parameters: size *n*, degree *d*, diameter *k*
- What are the extremal graphs?
  - Fix *d*, *k*. What is largest possible value of *n*?
- "Degree/Diameter problem"

Degree-Diameter Graphs

- Three parameters: size *n*, degree *d*, diameter *k*
- What are the extremal graphs?
  - Fix *d*, *k*. What is largest possible value of *n*?
- "Degree/Diameter problem"
  - "Intuitively, the best known degree-diameter topologies should support a large number of servers with high network bandwidth and low cost (small degree)... Thus, we propose the best-known degree-diameter graphs as a benchmark for comparison." — Singla et al, NSDI '12
  - Slim Fly [Besta-Hoefler, SC '14]: Uses near-optimal degree-diameter graphs for *k=2* (MMS graphs) and *k=3* (BDF and Delorme graphs)

Informal Result

- So what are the “best” datacenter topologies?
  - Optimal expanders (Ramanujan graphs)? Or optimal degree-diameter graphs (Moore graphs)?
  - Similar performance in simulation (degree-diameter graphs slightly worse)

Informal Result

- So what are the “best” datacenter topologies?
  - Optimal expanders (Ramanujan graphs)? Or optimal degree-diameter graphs (Moore graphs)?
  - Similar performance in simulation (degree-diameter graphs slightly worse)

Informal result: Any sufficiently good degree-diameter graph is a good expander!

Informal Result

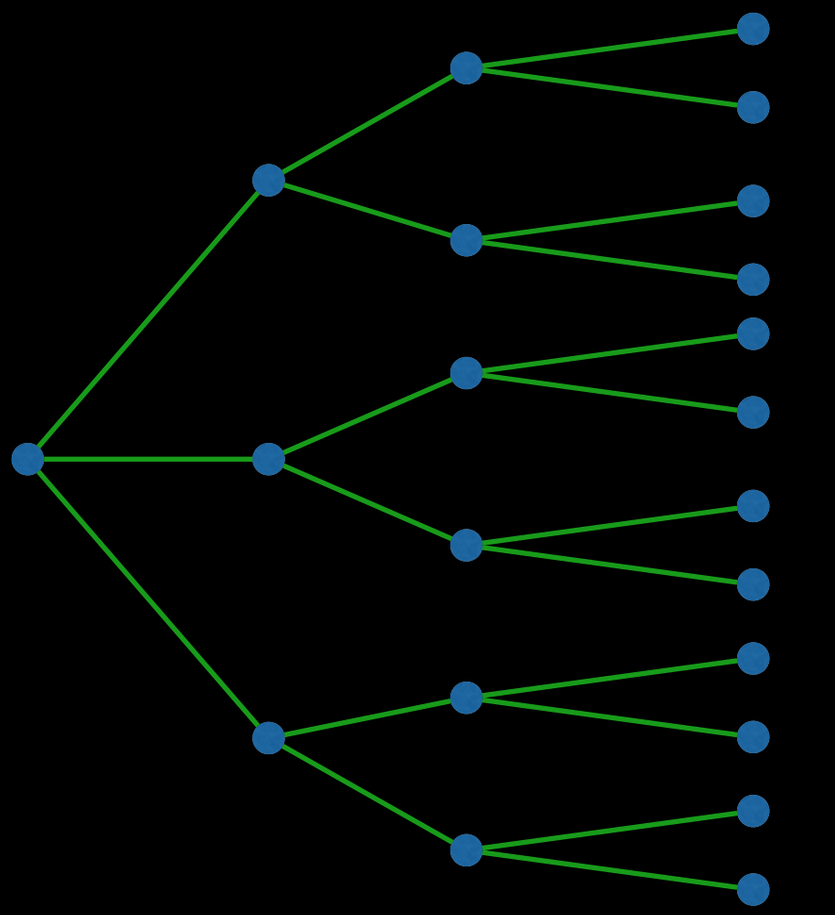
- So what are the “best” datacenter topologies?
  - Optimal expanders (Ramanujan graphs)? Or optimal degree-diameter graphs (Moore graphs)?
  - Similar performance in simulation (degree-diameter graphs slightly worse)

Informal result: Any sufficiently good degree-diameter graph is a good expander!

- So finding good degree-diameter graphs involves finding good expanders
- Expanders already very good — just use them

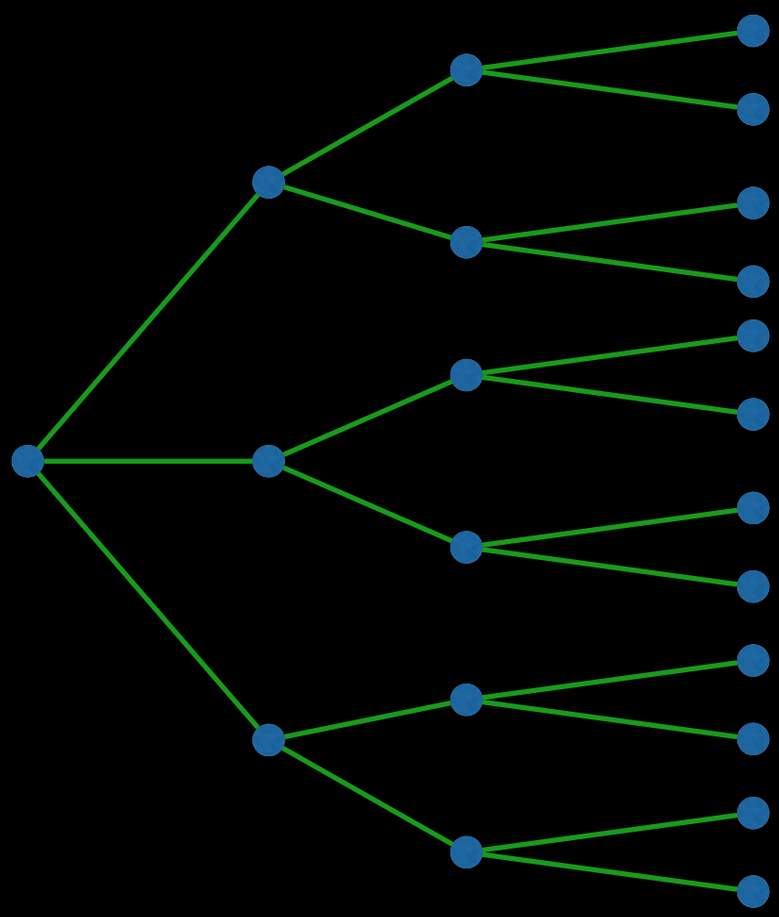
Moore Bound

- Fix *d*, *k*. Obvious upper bound on *n*:



Moore Bound

- Fix *d*, *k*. Obvious upper bound on *n*:



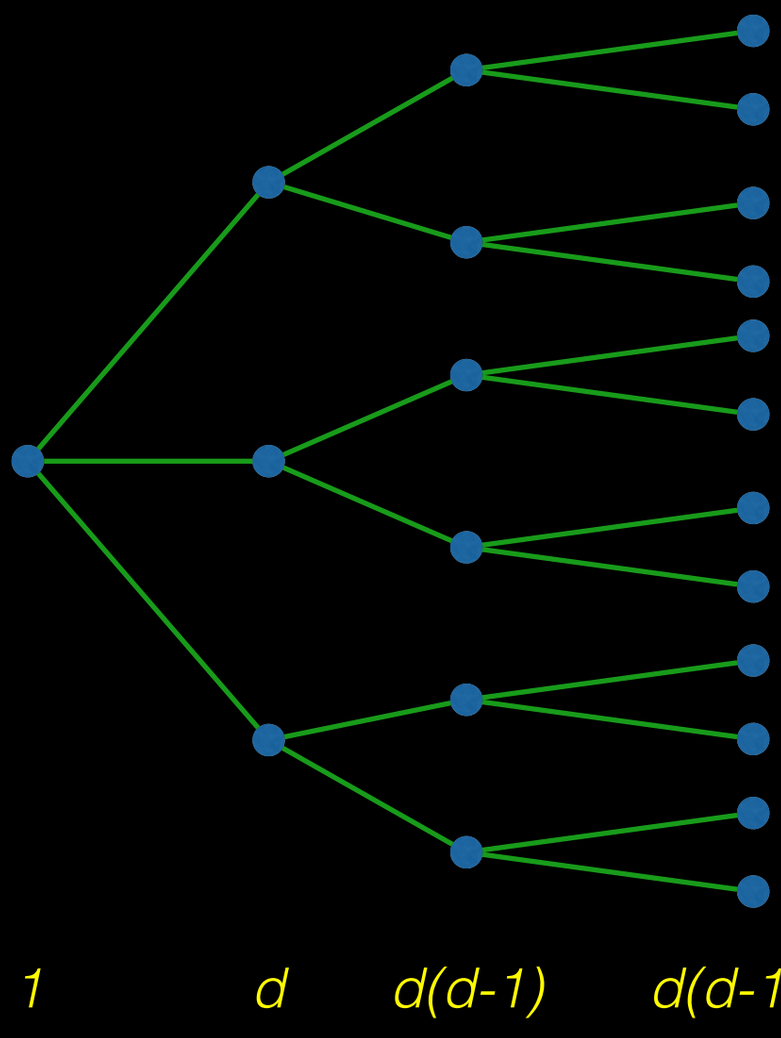
Moore bound:

$$\mu_{d,k} = 1 + d + d(d-1) + d(d-1)^2 + \cdots + d(d-1)^{k-1}$$

$$= 1 + d\sum_{i=0}^{k-1}(d-1)^i$$

Moore Bound

- Fix *d*, *k*. Obvious upper bound on *n*:



Moore bound:

$$\mu_{d,k} = 1 + d + d(d-1) + d(d-1)^2 + \cdots + d(d-1)^{k-1}$$

$$= 1 + d\sum_{i=0}^{k-1}(d-1)^i$$

- Not achievable in general
- Lots of work by graph theorists
- Can get arbitrarily close (as *n* gets large) for *k = 2, 3, 5*

Formal Results

- Algebraic / spectral expansion: $\lambda(G)$ = second largest eigenvalue of adj. matrix
- Cheeger inequality: $h(G) \geq (d-\lambda(G))/2$

Formal Results

- Algebraic / spectral expansion: $\lambda(G)$ = second largest eigenvalue of adj. matrix
- Cheeger inequality: $h(G) \geq (d-\lambda(G))/2$

Theorem: Any graph with degree d, diameter k, and $n \geq (1-\varepsilon)\ \mu_{d,k}$ has $\lambda(G) \leq O(\varepsilon^{1/k})\ d$

Theorem: Any graph with degree d, diameter k, and $n \geq \mu_{d,k} - O(d^{k/2})$ has $\lambda(G) = O(d^{1/2})$

Techniques

- Connection to **Geronimus Polynomials**
  - Formal polynomials $P_t(x)$, where $P_t(A)$ is matrix giving # "irreducible walks" of length t

Theorem: Let G be graph with degree d, diameter k, and size n. Then for every nontrivial eigenvalue λ,

$$\left| \sum_{t=0}^{k} P_t(\lambda) \right| \leq \mu_{d,k} - n$$

Conclusion (Part III)

- Expanders and degree/diameter graphs two different proposals, from different intuitions
- But and good degree/diameter graph is a good expander!
  - Suggests that good performance of degree/diameter graphs *because* of good expansion
  - Expanders easier to construct: just use a good expander
- Open questions:
  - What is true relationship? No reason to think our bounds tight
  - Moore bound possibly weak: is an optimal degree/diameter graph a good expander even if not close to Moore?

Thanks!