Self-Assembling Distributed Internet Software

Yuriy Brun

`http://www.cs.washington.edu/homes/brun`
brun@cs.washington.edu

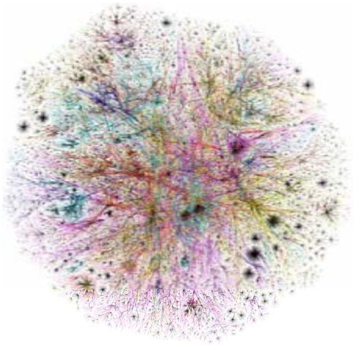
Biological Systems

- Self-healing
- Fault-tolerant
- Decentralized

In Contrast: Software

- Less complexity
- Fault-tolerance is "intelligently designed"
- Not expected to recover from catastrophes

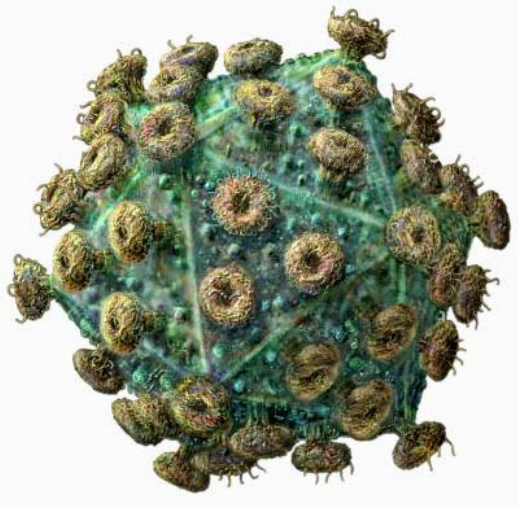
Computation is Evolving



Privacy: no entity should gain access to the data.

Security: no entity may undetectably compromise the computation.

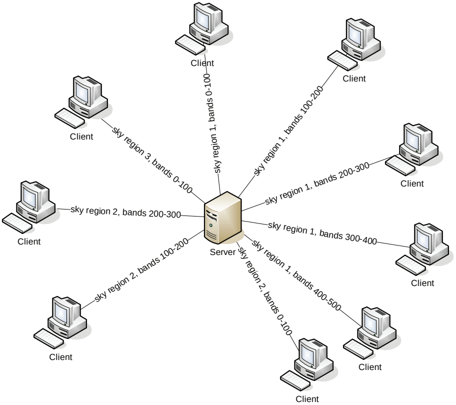
Parts of Real-World Problems Are Computational



- Protein folding
- Resource allocation
- Scheduling
- Encryption breaking
- ...

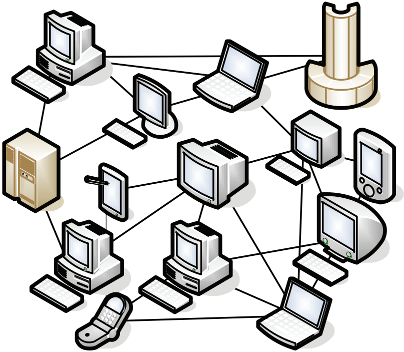
Distributed Computation

- Computation on the Internet
  - SETI@home [KWA+96]
  - Folding@Home [LSSP02]
  - Rosetta@home [Ros07]
- Grid Computing & Clouds
  - MapReduce [DG04]
  - OrganicGrid [CB04]
- Do not preserve privacy



My Approach: Intelligent Distribution

Obstacle: Discreet classical NP-hard computation is impossible [Chi05].



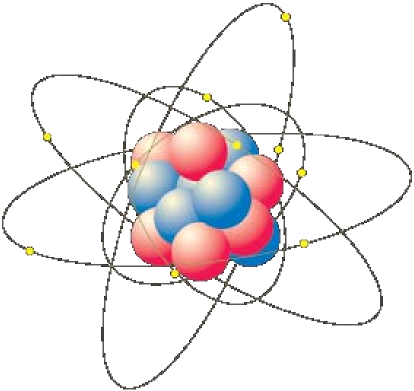
Solution: Distribute a computation so that no small group of machines knows too much or has too much power.

Outline

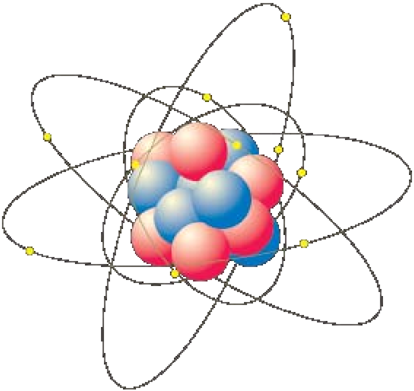
1. Private Distributed Computation
2. Self-Assembly
3. Tile Architectural Style
4. Contributions

Outline

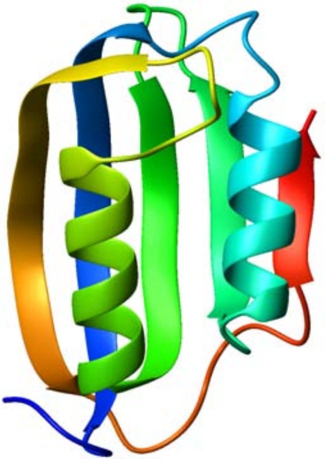
Self-Assembly in Nature



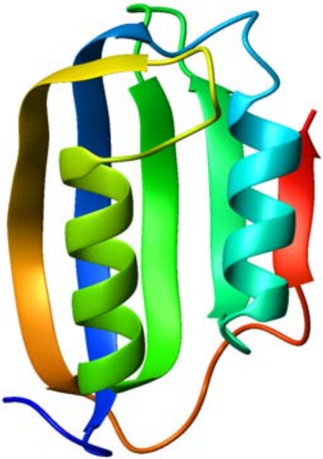
Self-Assembly in Nature

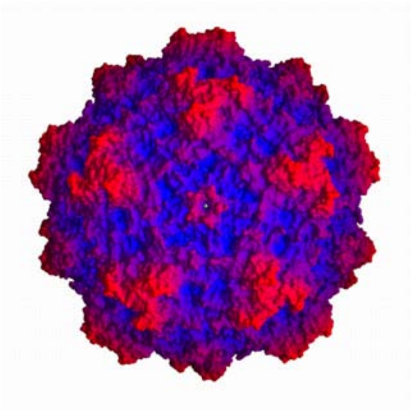


Self-Assembly in Nature



Self-Assembly in Nature



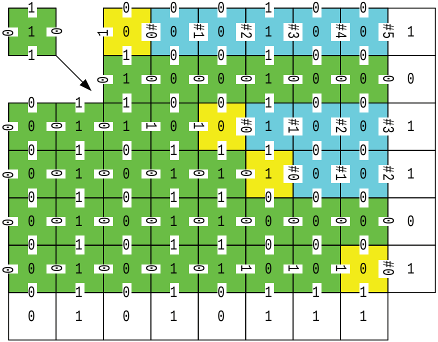


Self-Assembly in Nature

Self-Assembly in Nature

Tile Assembly Model [Win98b]

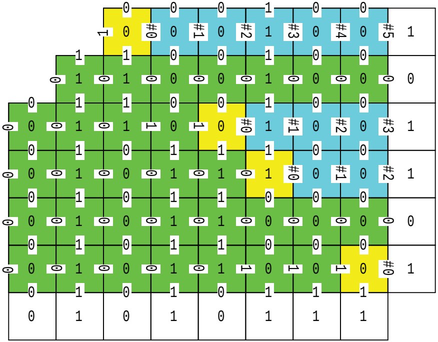
- Tile: a square with labels
- Each label has a strength
- Tiles attach if labels are strong enough



Tiles are Turing Universal [Win98b]

Tile Assembly Model [Win98b]

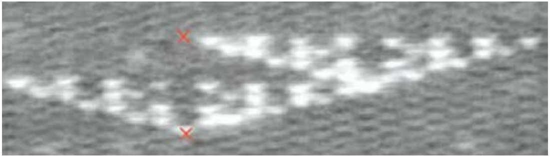
- Tile: a square with labels
- Each label has a strength
- Tiles attach if labels are strong enough



Tiles are Turing Universal [Win98b]

Efficiency in Tile Systems

- Assemble
  - linear polymers [ACG+01]
  - squares [RW00, AGHM02, ACG+02]
  - computable shapes [SW07]
- Count [Win98a, Moi05, BRW05]
- Compute Binomial Coefficients [Win98a, RPW04]



Efficient Computation

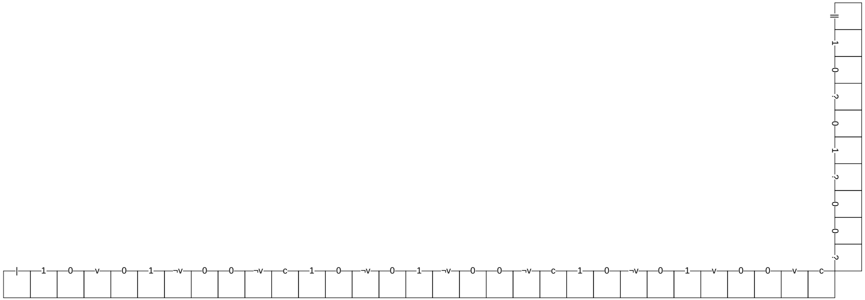
- Add [Bru07]
- Multiply [Bru07]
- Factor [Bru08a]
- Solve SubsetSum [Bru08b]
- Solve k-SAT [Bru08c]

Efficient Computation

- Add [Bru07]
- Multiply [Bru07]
- Factor [Bru08a]
- Solve SubsetSum [Bru08b]
- Solve k-SAT [Bru08c]

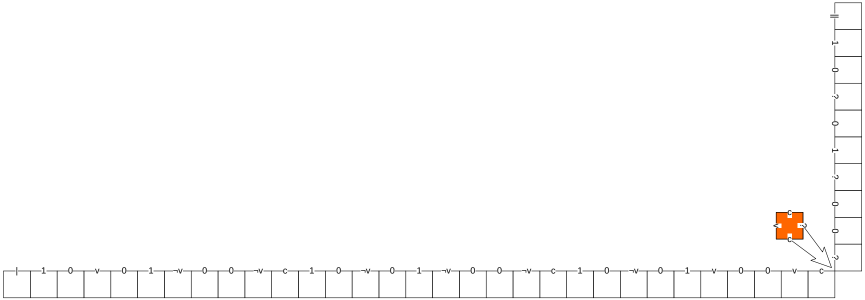
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



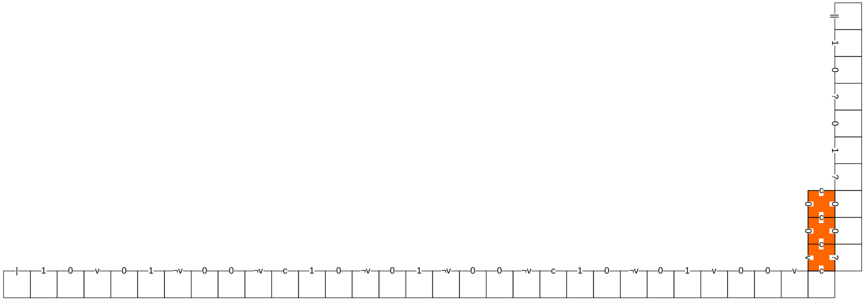
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



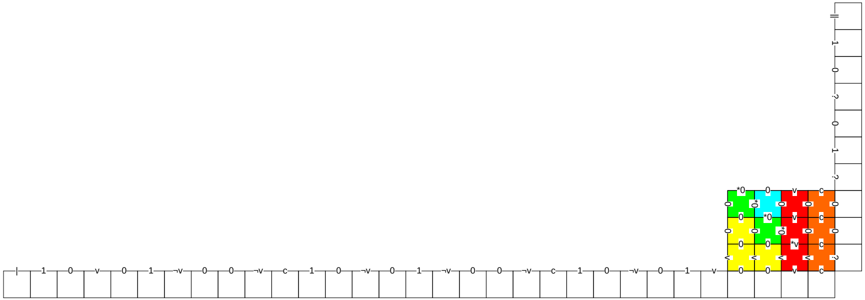
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



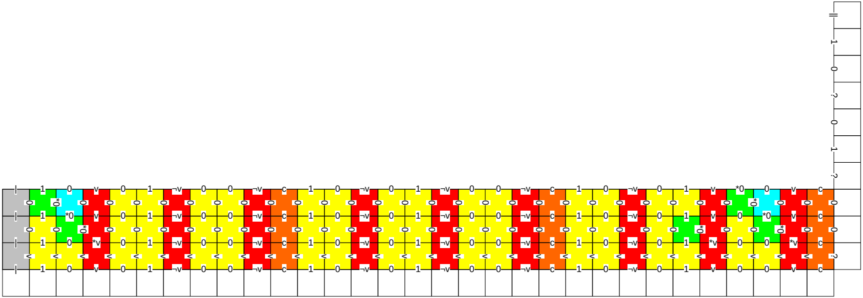
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



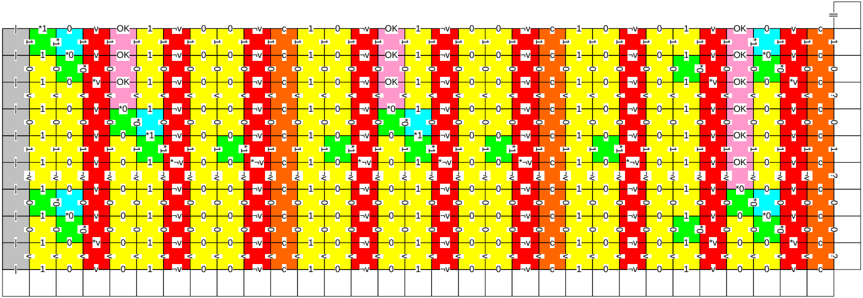
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



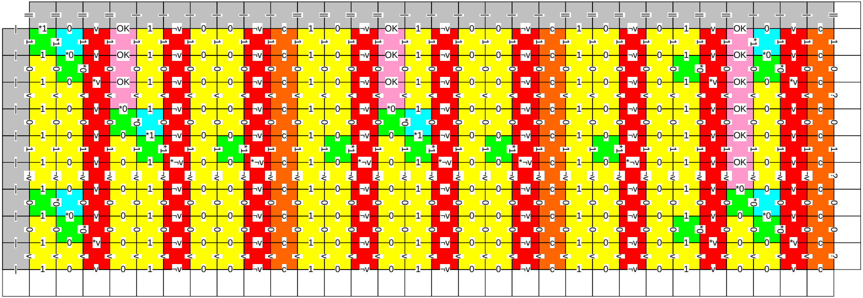
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



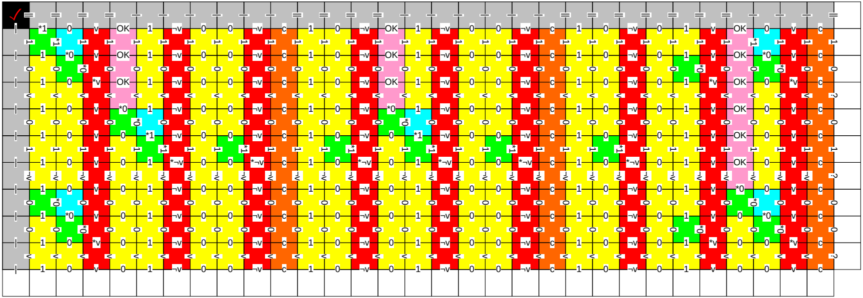
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



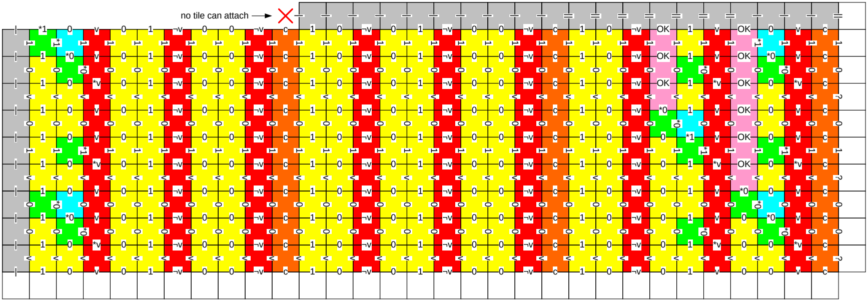
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



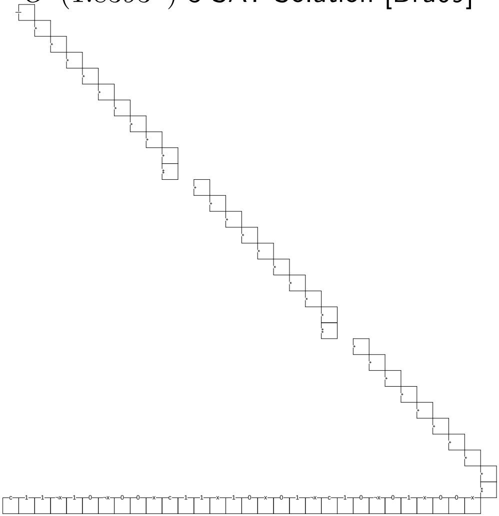
Solving 3-SAT

$$(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$$



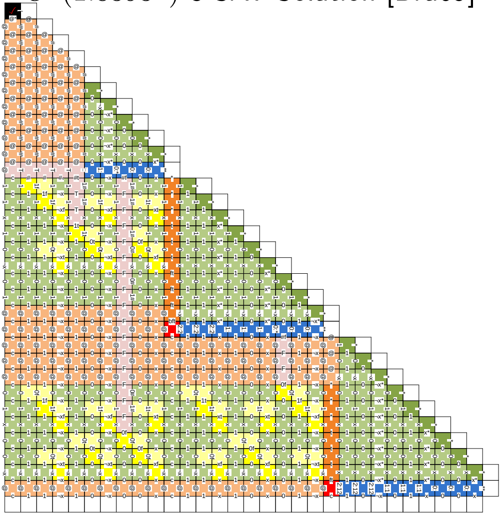
Can Tiles Implement More-Efficient Algorithms?

$O^{\star}(1.8393^{n})$ 3-SAT Solution [Bru09]



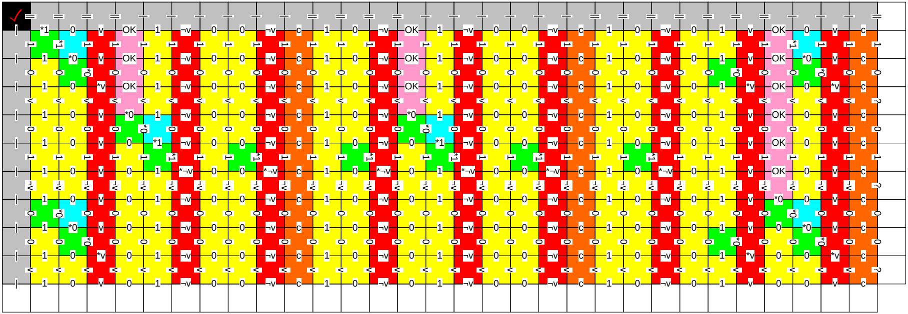
Can Tiles Implement More-Efficient Algorithms?

$O^\star(1.8393^n)$ 3-SAT Solution [Bru09]

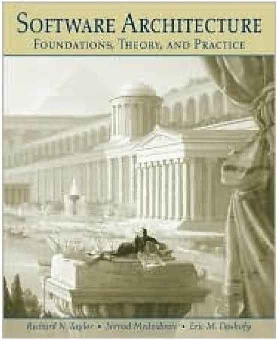


Outline

Tile Style Intuition

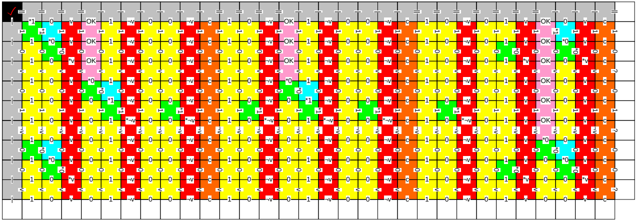


Software Architecture



"Software architecture: the set of principal design decisions made about a system." [TMD09]

Converting the Model to an Architecture

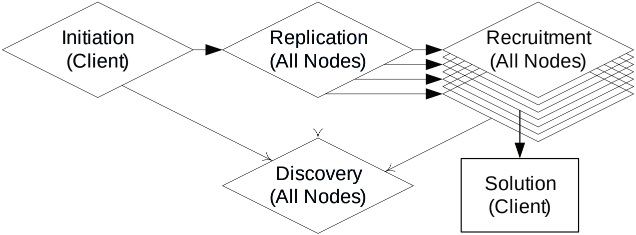


Architectural Elements [MRMM02]

- Components: tiles
- Interfaces: side labels
- Topology: 2-D grid
- Behaviors: identifying nodes, recruiting attachments, replicating, and reporting the solution
- Interaction: recruitment data exchange

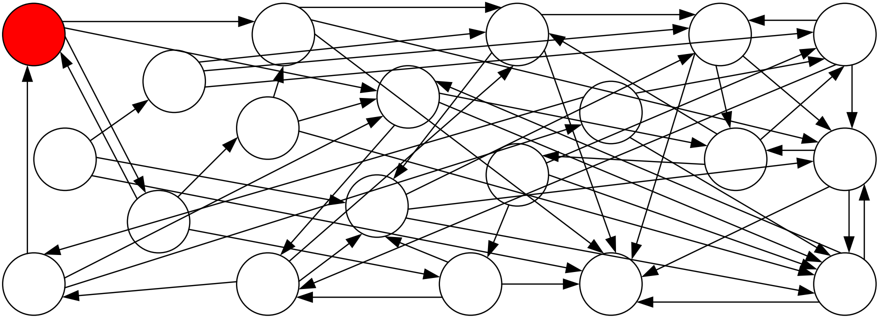
Node Operations [BM07]

- Initiation (by the client)
- Node Discovery
- Replication
- Recruitment



Node Discovery

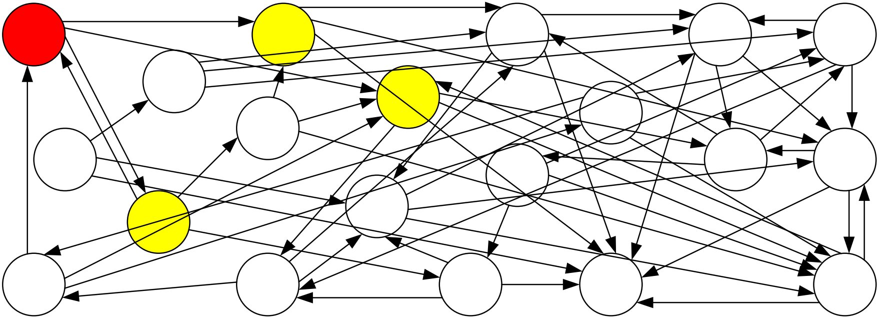
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

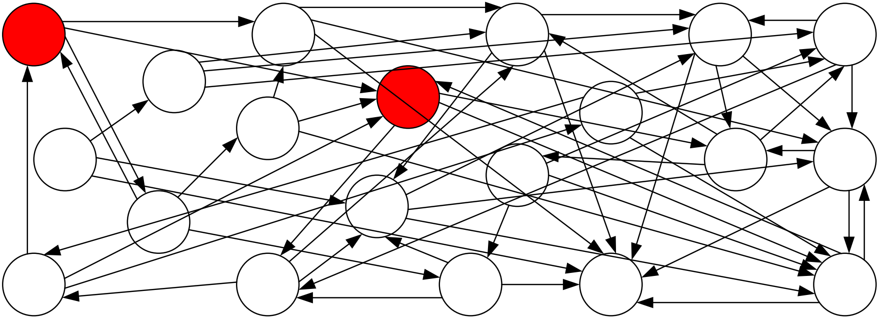
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

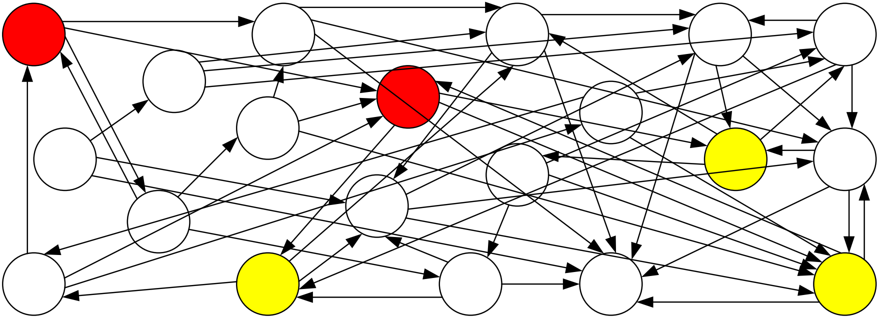
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

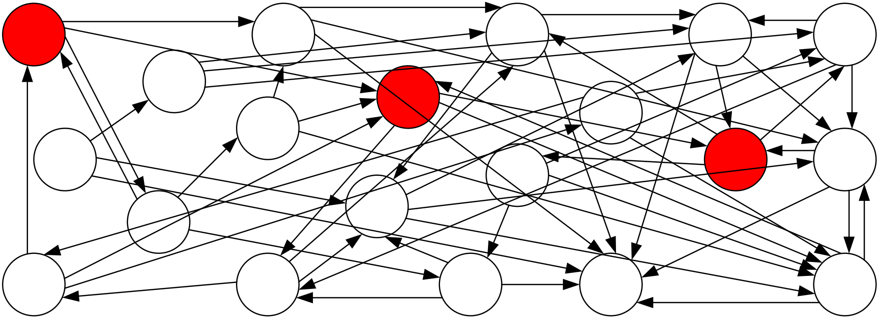
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

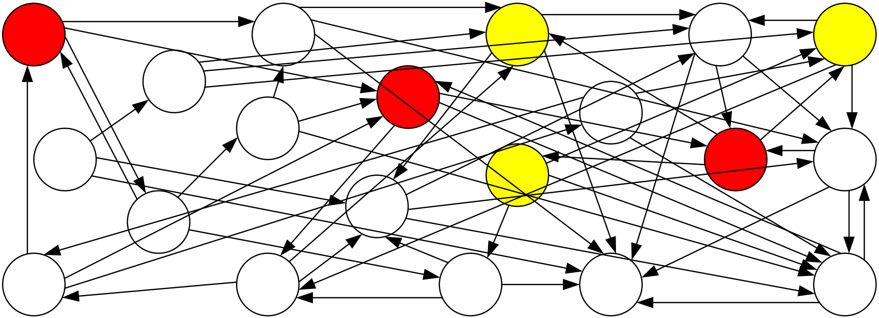
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

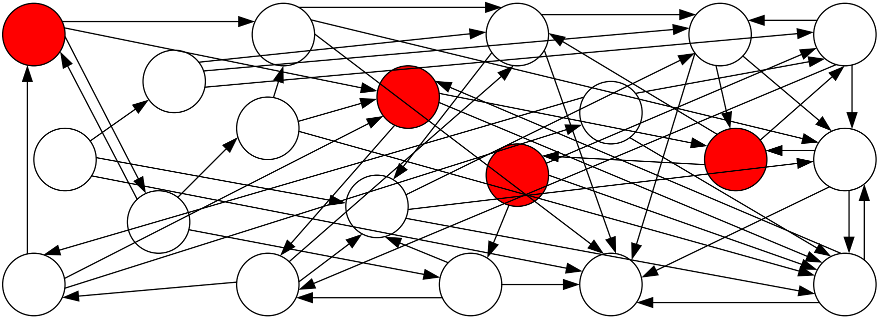
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

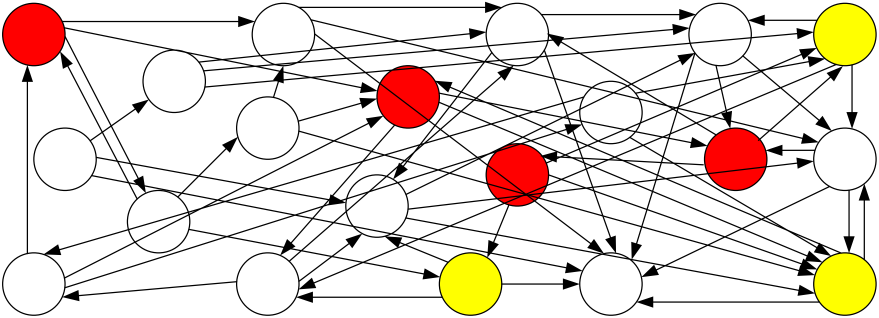
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

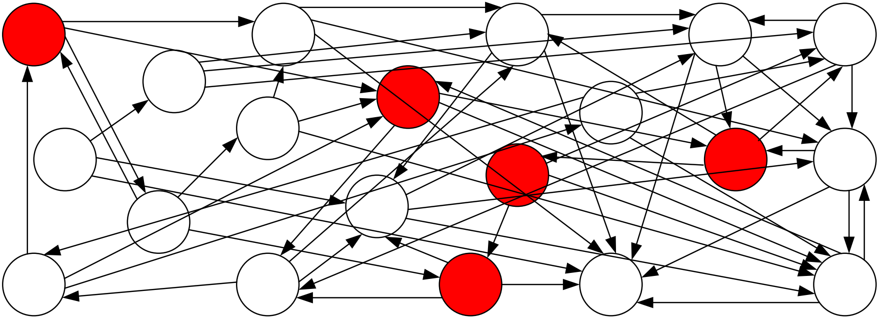
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

Node Discovery

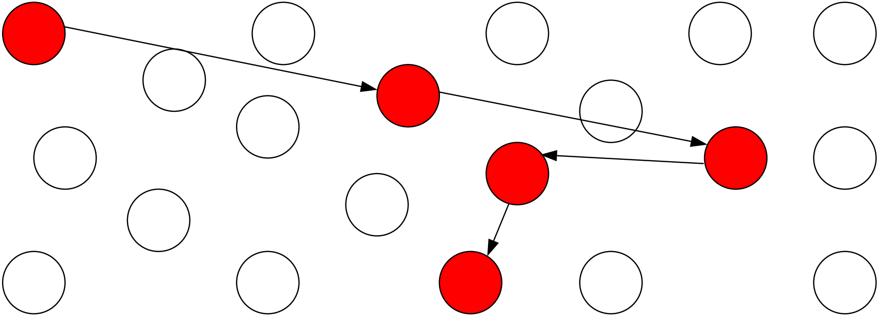
Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(logN)$ requests [MR95].

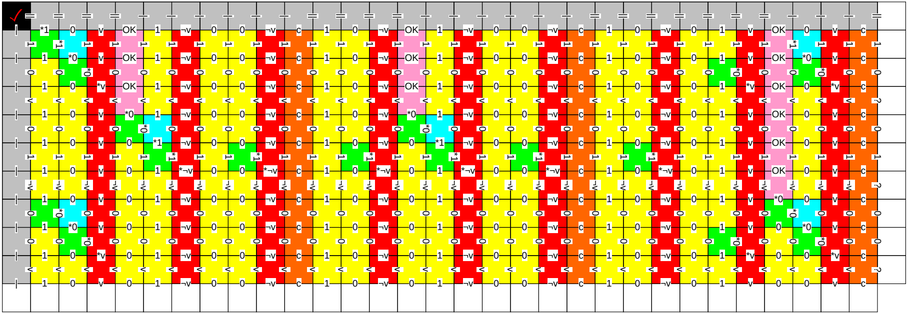
Node Discovery

Goal: Selecting a uniformly random node.



A random walk on the graph provably returns a uniformly random node after only $\Theta(log N)$ requests [MR95].

Data: Each Node Knows Very Little



Less than 1 bit of information per tile.

Data: It Is Hard to Control the Entire Input

$$\boxed{1 - (1 - c^n)^s}$$

n — bits in input

c — compromised fraction

s — number of seeds

Data: It Is Hard to Control the Entire Input

$$\boxed{1-(1-c^n)^s}$$

n — bits in input c — compromised fraction s — number of seeds

TeraGrid Case Study

- $\sim 100{,}000$ machines
- 17-variable 100-clause 3-SAT problem

Compromised Fraction	Confidence Level
$\frac{1}{8}$	$1-10^{-10}$
$\frac{1}{4}$	$1-10^{-5}$
$\frac{1}{3}$	$1-10^{-3}$

Experimental Setup

- Mahjong: tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: http://csse.usc.edu/~ybrun/Mahjong

Experimental Setup

- Mahjong: tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: http://csse.usc.edu/~ybrun/Mahjong
- Networks
  - 11-node private cluster (P4 1.5GHz, 512MiB, WinXP/2000)
  - 186-node USC HPCC cluster [Hig] (P4 Xeon 3GHz, Linux)
  - 100-node PlanetLab [PACR03] (global, varying speeds and resources)

Experimental Setup

- Mahjong: tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: http://csse.usc.edu/~ybrun/Mahjong
- Networks
  - 11-node private cluster (P4 1.5GHz, 512MiB, WinXP/2000)
  - 186-node USC HPCC cluster [Hig] (P4 Xeon 3GHz, Linux)
  - 100-node PlanetLab [PACR03] (global, varying speeds and resources)
- Sample problems:

  $\mathfrak{A}$: 5-number 21-bit $SubsetSum$

  $\mathfrak{B}$: 11-number 28-bit $SubsetSum$

  $\mathfrak{C}$: 20-variable 20-clause 3-SAT

  $\mathfrak{D}$: 33-variable 100-clause 3-SAT

Scalability: Speed $\propto$ Network Size

Network & Problem	# of Nodes	Execution Time	Speed-up Ratio
Private Cluster	5	43 sec.	
$\mathfrak{A}$	10	23 sec.	1.9
HPCC	93	220 min.	
$\mathfrak{C}$	186	116 min.	1.9
PlanetLab	50	9.2 min.	
$\mathfrak{B}$	100	4.8 min.	1.9
Simjong	125,000	8.7 hours	
	250,000	4.5 hours	1.9
$\mathfrak{D}$	500,000	2.1 hours	2.1
	1,000,000	64 min.	2.0

Scalability: Speed $\propto$ Network Size

Network & Problem	# of Nodes	Execution Time	Speed-up Ratio
Private Cluster	5	43 sec.	
$\mathfrak{A}$	10	23 sec.	1.9
Simjong	125,000	8.7 hours	
	250,000	4.5 hours	1.9
$\mathfrak{D}$	500,000	2.1 hours	2.1
	1,000,000	64 min.	2.0

System speed scales almost linearly with network size

Robustness to Network Delay

Problem	# of Nodes	Network Delay	Execution Time
Mahjong			
$\mathfrak{A}$	11	Private Cluster	20.1 sec.
		HPCC	19.3 sec.
		PlanetLab	18.5 sec.
$\mathfrak{B}$	11	Private Cluster	41.6 min.
		HPCC	41.2 min.
		PlanetLab	43.9 min.
Simjong			
$\mathfrak{D}$	1,000,000	0ms	65 min.
		10ms	57 min.
		100ms	64 min.
		500ms	60 min.
		Gaussian	68 min.
		Distance-based	59 min.

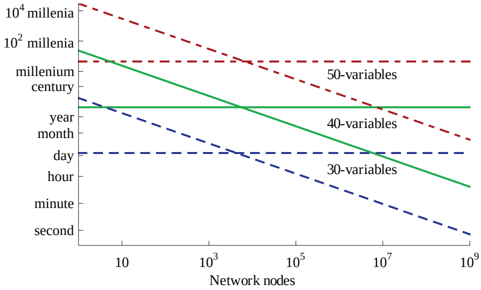
Robustness to Network Delay

Problem	# of Nodes	Network Delay	Execution Time
Mahjong			
$\mathfrak{A}$	11	Private Cluster	20.1 sec.
		HPCC	19.3 sec.
		PlanetLab	18.5 sec.

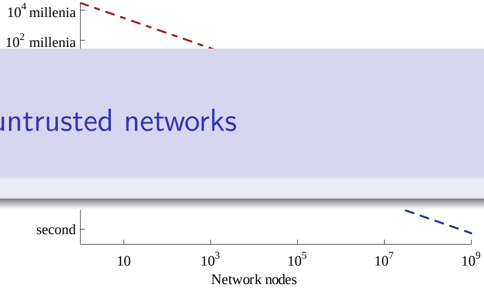
Network latency does not affect system throughput

$\mathfrak{D}$	1,000,000	0ms	65 min.
		10ms	57 min.
		100ms	64 min.
		500ms	60 min.
		Gaussian	68 min.
		Distance-based	59 min.

Efficiency: Solving Real-World-Sized Problems



Efficiency: Solving Real-World-Sized Problems



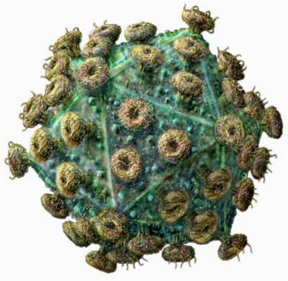
Aimed at large untrusted networks

Related Work

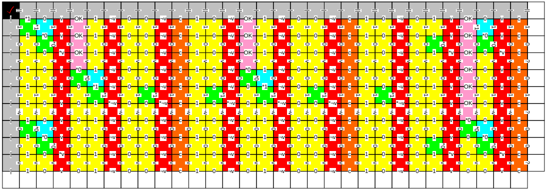
- Private computation in quantum computing through entanglement [Chi05]
- Homomorphic encryption for private computation [Gen09]
- Plethora of non-private distributed computation work [BOI09, KWA$^+$96, LSSP02, Ros07, DG04, CB04]
- ...and fault-tolerant computation work [Sar02, BGX93, BCGX02, FS01, KK07, HK03]

Contributions

- Developed self-assembling systems to solve complex computational problems



- Designed the tile architectural style for deploying tile systems on large networks



Leonard Adleman, Qi Cheng, Ahish Goel, Ming-Deh Huang, and Hal Wasserman.
Linear self-assemblies: Equilibria, entropy, and convergence rates.
In *Proceedings of the 6th International Conference on Difference Equations and Applications (ICDEA01)*, Augsburg, Germany, June 2001.

Leonard Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund.
Combinatorial optimization problems in self-assembly.
In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC02)*, pages 23–32, Montreal, Quebec, Canada, May 2002.

Leonard Adleman, Ashish Goel, Ming-Deh Huang, and Pablo Moisset de Espanés.
Running time and program size for self-assembled squares.
In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC02)*, pages 740–748, Montreal, Quebec, Canada, May 2002.

Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico, and Jie Xu.
An adaptive approach to achieving hardware and software fault tolerance in a distributed computing environment.
Journal of Systems Architecture, 47(9):763–781, 2002.

Andrea Bondavalli, Felicita Di Giandomenico, and Jie Xu.
A cost-effective and flexible scheme for software fault tolerance.
Journal of Computer Systems Science and Engineering, 8(4):234–244, 1993.

Yuriy Brun and Nenad Medvidovic.
An architectural style for solving computationally intensive problems on large networks.
In *Proceedings of Software Engineering for Adaptive and Self-Managing Systems (SEAMS07)*, Minneapolis, MN, USA, May 2007.

BOINC.
The Berkeley open infrastructure for network computing.
`http://boinc.berkeley.edu`, 2009.

Yuriy Brun.
Arithmetic computation in the tile assembly model: Addition and multiplication.
Theoretical Computer Science, 378(1):17–31, June 2007.

Yuriy Brun.
Nondeterministic polynomial time factoring in the tile assembly model.
Theoretical Computer Science, 395(1):3–23, April 2008.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2007-707.

Yuriy Brun.
Solving NP-complete problems in the tile assembly model.
Theoretical Computer Science, 395(1):31–46, April 2008.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2007-703.

Yuriy Brun.
Solving satisfiability in the tile assembly model with a constant-size tileset.
Journal of Algorithms, 63(4):151–166, 2008.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2008-801.

Yuriy Brun.
Improving efficiency of 3-sat-solving tile systems.
In Submission, 2009.

Robert Barish, Paul W. K. Rothemund, and Erik Winfree.
Two computational primitives for algorithmic self-assembly: Copying and counting.
Nano Letters, 5(12):2586–2592, 2005.

Arjav J. Chakravarti and Gerald Baumgartner.
The organic grid: Self-organizing computation on a peer-to-peer network.
In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC04)*, pages 96–103, New York, NY, USA, 2004.

Andrew M. Childs.
Secure assisted quantum computation.
Quantum Information and Computation, 5(456), 2005.

Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.
In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI04)*, San Francisco, CA, USA, December 2004.

Pascal Felber and Andre Schiper.
Optimistic active replication.
In *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS01)*, pages 333–341, Phoenix, AZ, USA, 2001. IEEE Computer Society.

Craig Gentry.
Fully homomorphic encryption using ideal lattices.
In *Proceedings of the 41st annual ACM Symposium on Theory of Computing (STOC09)*, pages 169–178, Bethesda, MD, USA, 2009.

High performance computing and communications.
`http://www.usc.edu/hpcc`.

Soonwook Hwang and Carl Kesselman.
A flexible framework for fault tolerance in the grid.
Journal of Grid Computing, 1(3):251–272, September 2003.

Israel Koren and C. Mani Krishna.
Fault-Tolerant Systems.
Elsvier, Inc., 2007.

Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky.
SETI@home — massively distributed computing for SETI.
IEEE MultiMedia, 3(1):78–83, 1996.

Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande.
Folding@Home and Genome@Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology.
Horizon Press, 2002.

Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic.
A style-aware architectural middleware for resource-constrained, distributed systems.
IEEE Transactions on Software Engineering, 31(3):256–272, 2005.

Pablo Moisset de Espanés.
Computerized exhaustive search for optimal self-assembly counters.
In *Proceedings of the 2nd Foundations of Nanoscience: Self-Assembled Architectures and Devices (FNANO05)*, pages 24–25, Snowbird, UT, USA, April 2005.

Rajeev Motwani and Prabhakar Raghavan.
Randomized Algorithms.
Cambridge University Press, New York, NY, USA, 1995.

Marija Mikic-Rakic, Nikunj R. Mehta, and Nenad Medvidovic.
Architectural style requirements for self-healing systems.
In *Proceedings of 1st Workshop on Self-Healing Systems*, Charleston, SC, USA, November 2002.

Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe.
A blueprint for introducing disruptive technology into the Internet.
ACM SIGCOMM Computer Communication Review, 33(1):59–64, 2003.

Rosetta@home.
`http://boinc.bakerlab.org/rosetta`, 2007.

Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree.
Algorithmic self-assembly of DNA Sierpinski triangles.
PLoS Biology, 2(12):e424, 2004.

Paul W. K. Rothemund and Erik Winfree.
The program-size complexity of self-assembled squares.
In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC00)*, pages 459–468, Portland, OR, USA, May 2000.

Luis F. G. Sarmenta.
Sabotage-tolerance mechanisms for volunteer computing systems.
Future Generation Computer Systems, 18(4):561–572, 2002.

David Soloveichik and Erik Winfree.
Complexity of self-assembled shapes.
SIAM Journal on Computing, 36(6):1544–1569, 2007.

Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy.
Software Architecture: Foundations, Theory, and Practice.
John Wiley & Sons, 2009.

Erik Winfree.
Algorithmic Self-Assembly of DNA.
PhD thesis, California Institute of Technology, Pasadena, CA, USA, June 1998.

Erik Winfree.
Simulations of computing by self-assembly of DNA.
Technical Report CS-TR:1998:22, California Institute of Technology, Pasadena, CA, USA, 1998.