Software Architecture III
Leveraging Nature to Build Better Systems

Yuriy Brun

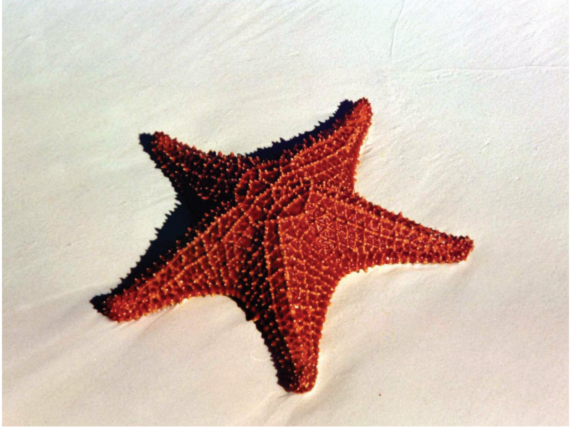
`http://www.cs.washington.edu/homes/brun/`

Outline

Outline

Systems in Nature

- Resilient to
  - death
  - malfunction
  - malicious agents
- Self-healing
- Fault-tolerant



In Contrast: Software

- Less-complex systems
- Fault-tolerance is “intelligently designed”
- Not expected to recover from catastrophes

Genetic Algorithms

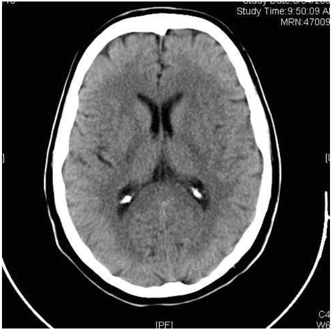
Have been used to:

- Design of fighter-planes airfoils [HO03]
- Train scheduling
- Automatic software bug patching [WNGF09]
- Data mirroring [RKCM09]

Neural Networks

Have been used to:

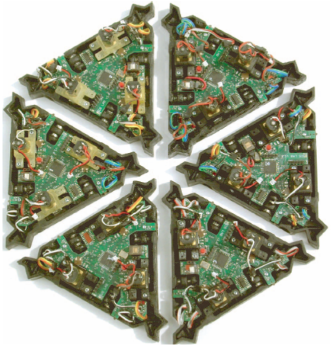
- Classification
- Sales forecasting / marketing
- Medical diagnoses [SKR01]
- Credit evaluation [Wes00]



Distributed Robotics

Have been used to:

- Search and rescue scenarios [MEB+10]
- Vacuum design
- Sensor networks [AAC+00]
- Education



Robofish

Outline

1. Why Nature?
2. Using Nature to Compute
   - DNA Computing
   - Bacteria Gates
   - Tile Assembly Model
3. Tiles
4. Tile Software
5. Conclusions

Leonard M. Adleman

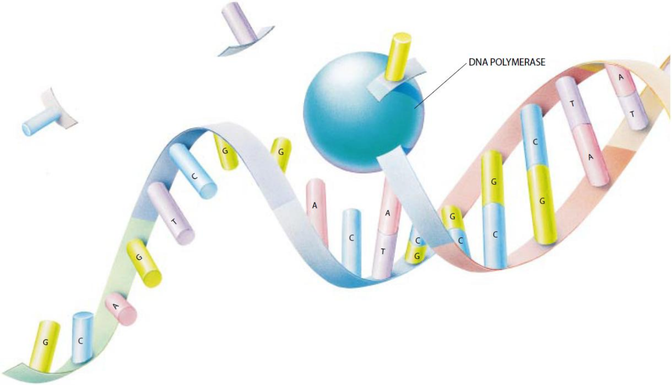
"The manipulation of DNA to solve mathematical problems is redefining what is meant by 'computation'."

A Bit of History

Adleman's research

- RSA public key cryptosystem [RSA78]
- Computer viruses [Adl90]
- HIV modeling [AW93]
- DNA computation [Adl94]

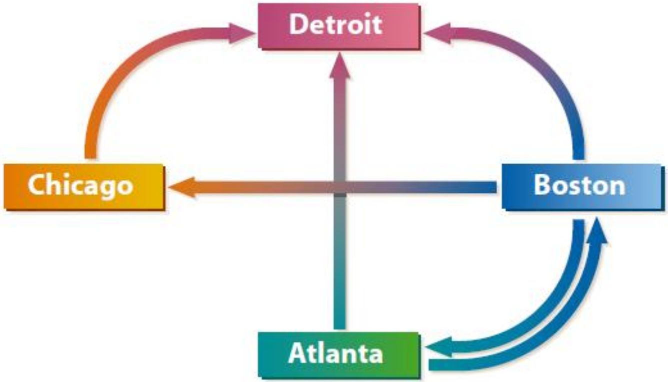
DNA Turing Machine



[Adl98]

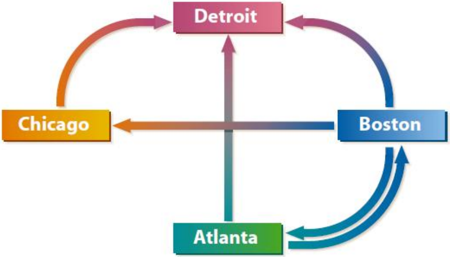
- A, T, C, and G can encode information
- A DNA strand is a data-storing tape
- Enzymes can encode states and rules for manipulating the tape

Hamiltonian Path Problem



[Adl98]

Hamiltonian Path Problem with DNA



CITY	DNA NAME	COMPLEMENT
ATLANTA	ACTTGCAG	TGAACGTC
BOSTON	TCGGACTG	AGCCTGAC
CHICAGO	GGCTATGT	CCGATACA
DETROIT	CCGAGCAA	GGCTCGTT

FLIGHT	DNA FLIGHT NUMBER
ATLANTA - BOSTON	GCAGTCGG
ATLANTA - DETROIT	GCAGCCGA
BOSTON - CHICAGO	ACTGGGCT
BOSTON - DETROIT	ACTGCCGA
BOSTON - ATLANTA	ACTGACTT
CHICAGO - DETROIT	ATGTCCGA

[Adl98]

Implementing the DNA Algorithm

	Conventional algorithm	**DNA algorithm**
1.	Generate a set of random paths	Mix city and flight strands
2.	Select paths that start and end at proper cities	PCR
3.	Select proper-length paths	Electrophoresis gel
4.	Select paths that visit each city	Watson & Crick pairing
5.	The remaining paths represent the solution	PCR, electrophoresis, and sequencing

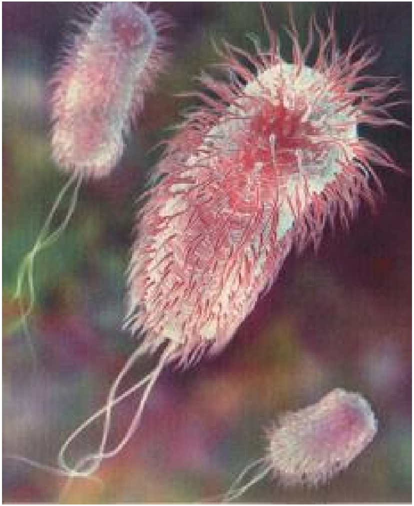
3-SAT With DNA

- In 2002, Braich et al. [BCJ+02] developed a DNA computer to solve 20-variable 3-SAT problems.
  - Worked most of time
  - Error rates grew proportionally to the number of variables
- A few other models emerged
  - Sticker model
  - Tile assembly model
    - more on this later...

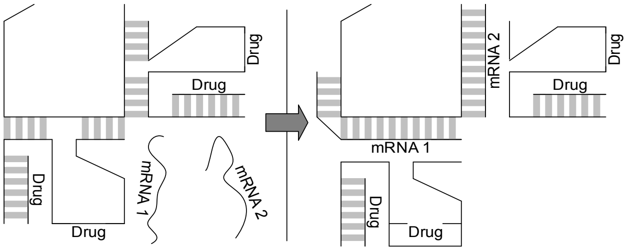
Protein Production Control

BioBricks [KS97]

- Controlling what proteins a cell produces
- Basis for the International Genetically Engineered Machine (iGEM) competition

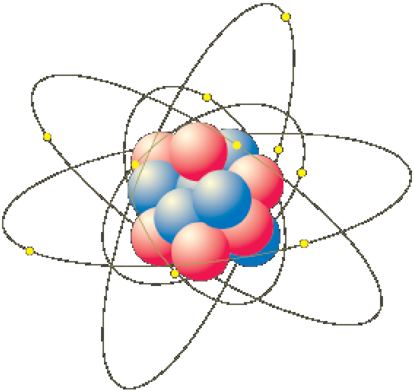


DNA Gates

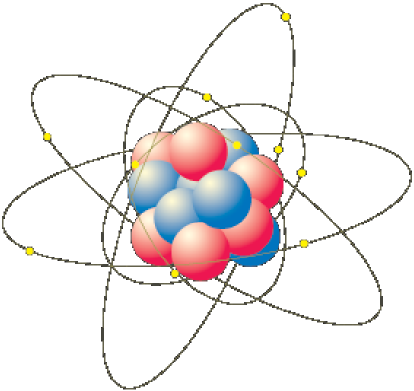


- Binary gates that act on DNA-strand inputs [BG06]
- Previous work used enzymes [BGBD$^+$04]
- Later work at Caltech improved the design [QW08]

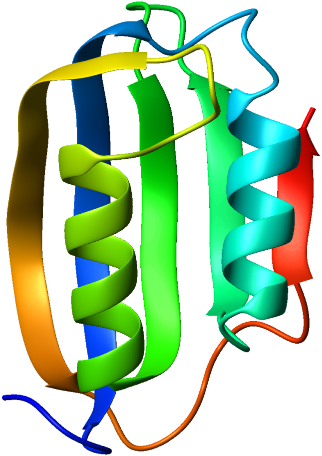
Self-Assembly in Nature



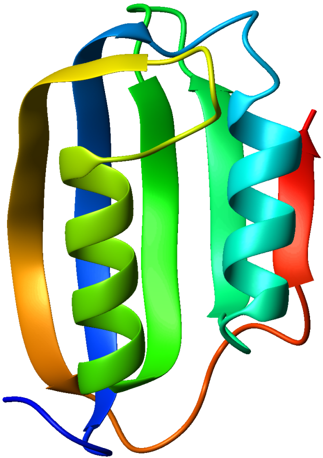
Self-Assembly in Nature

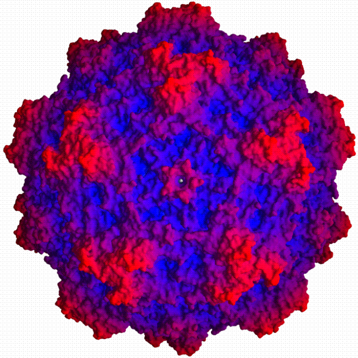


Self-Assembly in Nature

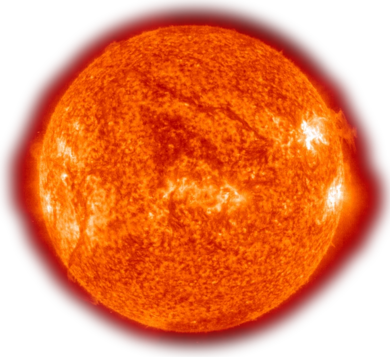


Self-Assembly in Nature

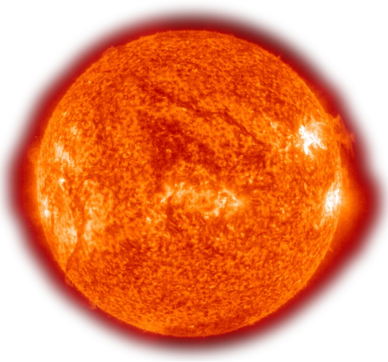


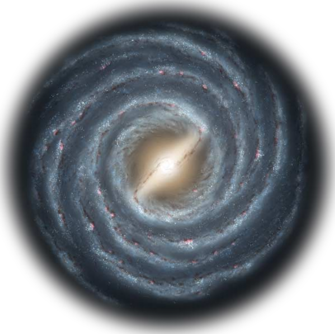


Self-Assembly in Nature



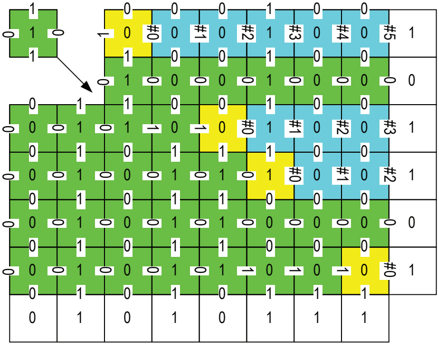
Self-Assembly in Nature





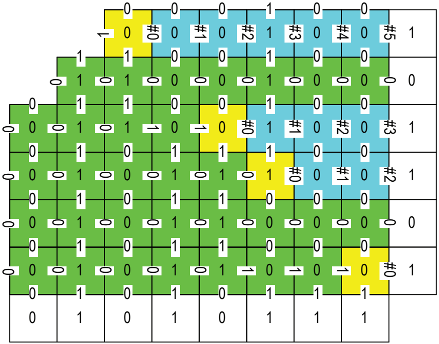
Tile Assembly Model [Win98b]

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



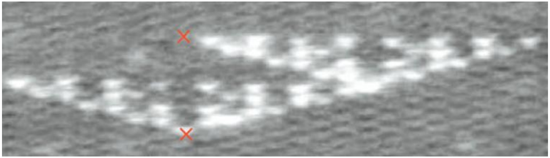
Tile Assembly Model [Win98b]

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



Tiles Can:

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

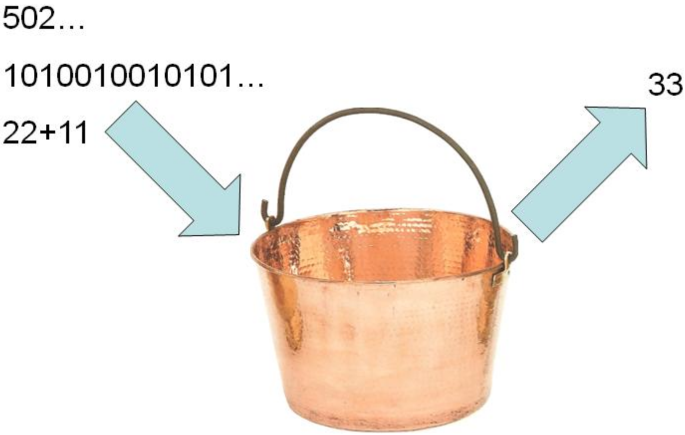


- Emulate Turing Machines [Win98b]

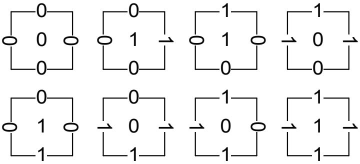
Outline

1. Why Nature?
2. Using Nature to Compute
3. Tiles
   - Adding and Multiplying
   - Solving 3-SAT
4. Tile Software
5. Conclusions

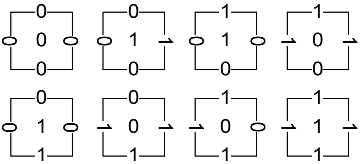
Computing with Tiles



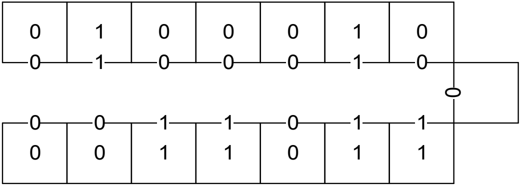
Adding with Tiles [Bru07]



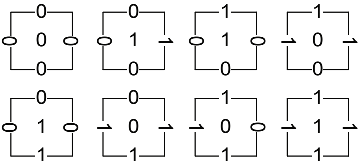
Adding with Tiles [Bru07]



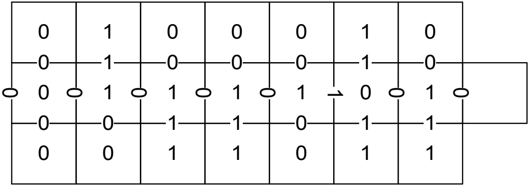
$$34 + 27$$



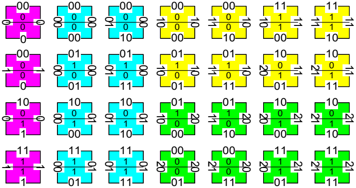
Adding with Tiles [Bru07]



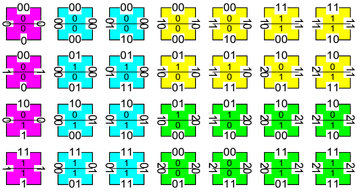
$$34 + 27 = 61$$



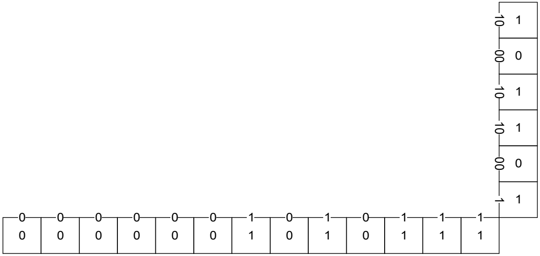
Multiplying with Tiles [Bru07]



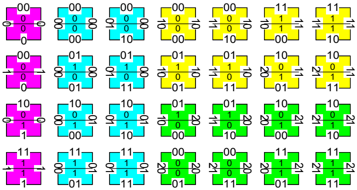
Multiplying with Tiles [Bru07]



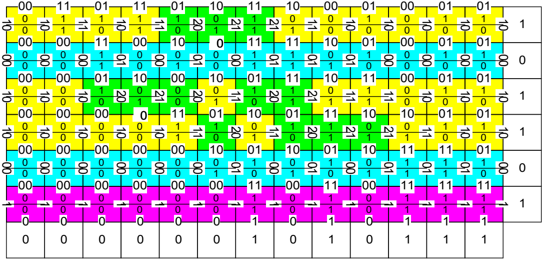
$$87 \times 45$$



Multiplying with Tiles [Bru07]



$$87 \times 45 = 3915$$



3-SAT

Variables: x_0, x_1, x_2 ...

Literals: x_0, $\neg x_0$, x_1, $\neg x_1$, ...

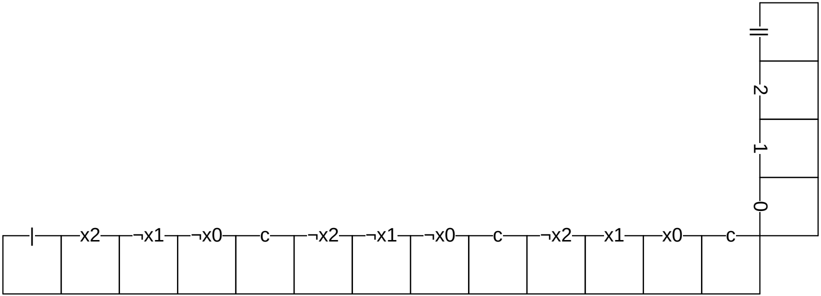
Clauses: $(x_2 \vee \neg x_1 \vee \neg x_0)$

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

The question: Does there exist an assignment of $TRUE$ / $FALSE$ values to the variables that makes the formula $TRUE$?

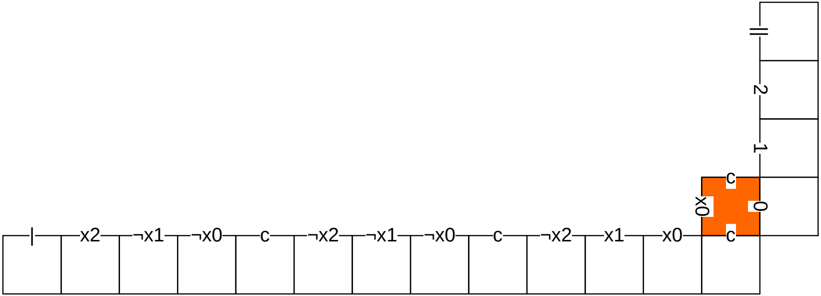
$\Theta(n^2)$-Tileset Approach [LL99]

$$(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)$$



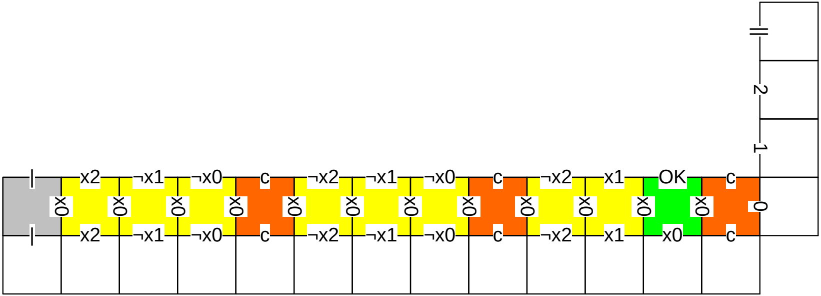
$\Theta(n^2)$-Tileset Approach [LL99]

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



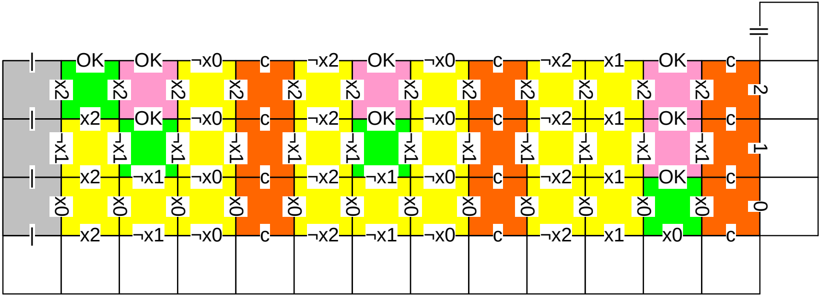
$\Theta(n^2)$-Tileset Approach [LL99]

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



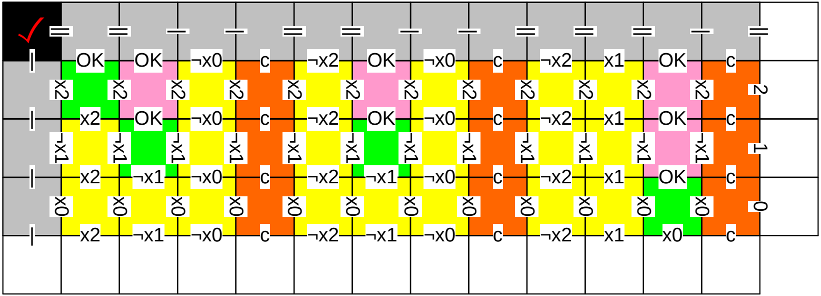
$\Theta(n^2)$-Tileset Approach [LL99]

$$(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)$$



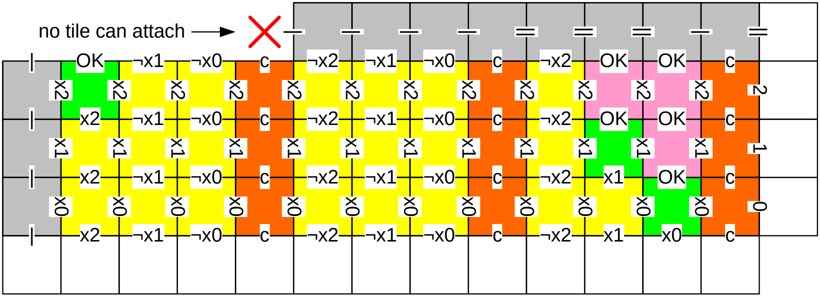
$\Theta(n^2)$-Tileset Approach [LL99]

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



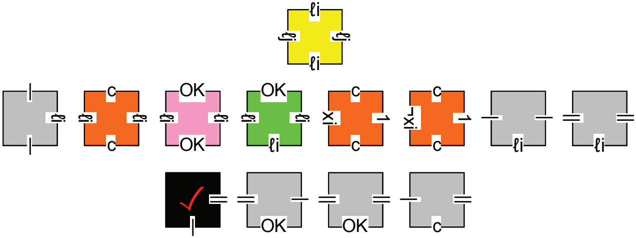
$\Theta(n^2)$-Tileset Approach [LL99]

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



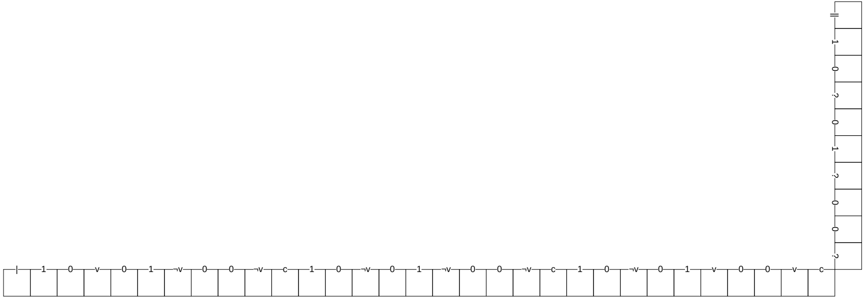
The $\Theta(n^2)$ Tileset 3-SAT Solution [LL99]

- $\Theta(n^2)$ tile types
- Probability of success $\geq \left(\frac{1}{2}\right)^n$



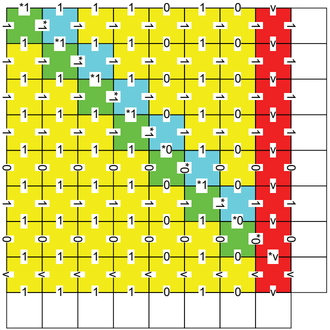
Encoding Formulae with a $\Theta(1)$ Tileset

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

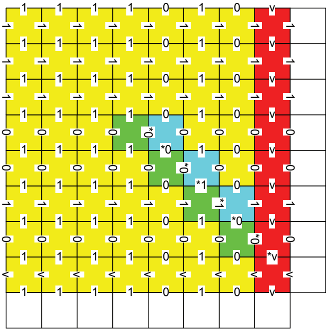


Comparing Literals

x_{122} vs. x_{122}

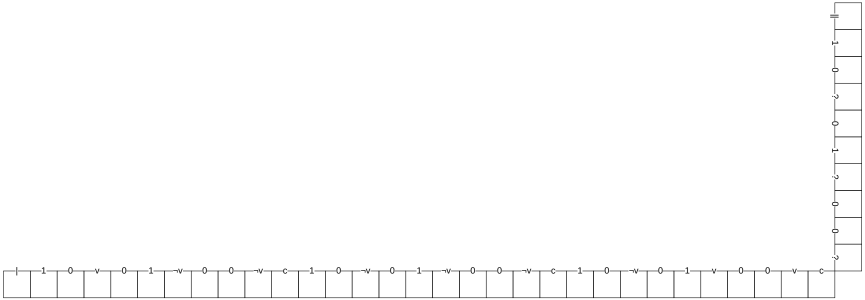


x_{122} vs. x_{114}



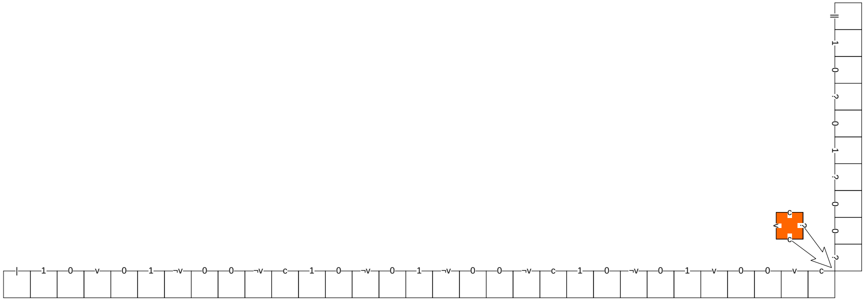
Solving 3-SAT

$$(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)$$



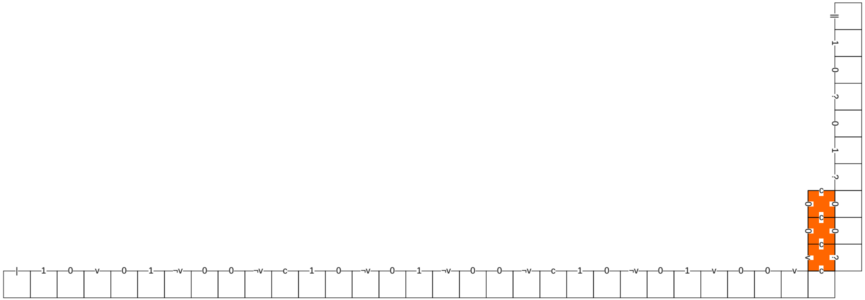
Solving 3-SAT

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



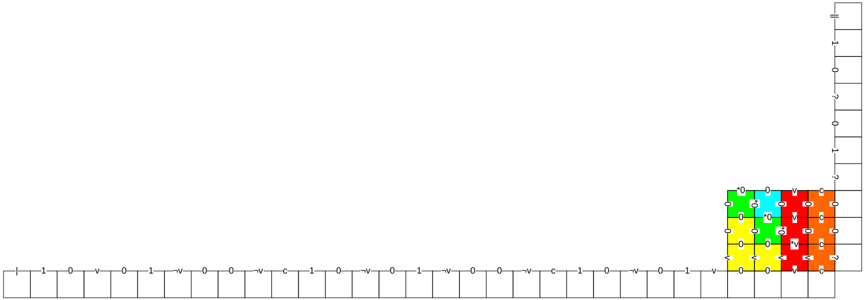
Solving 3-SAT

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



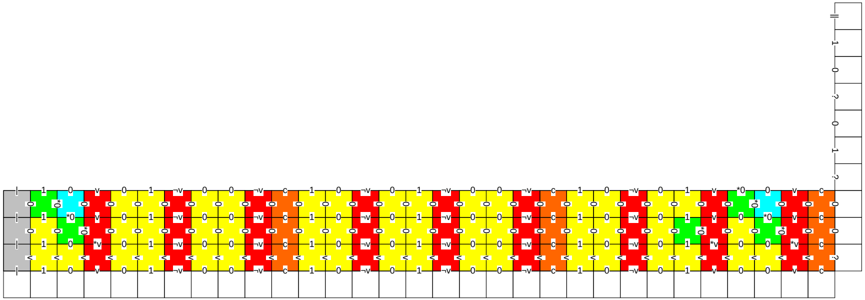
Solving 3-SAT

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



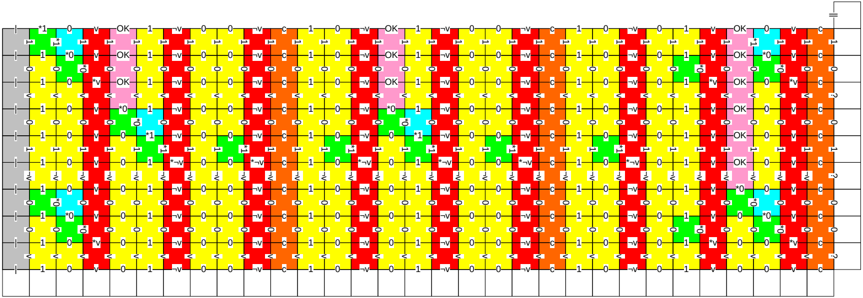
Solving 3-SAT

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



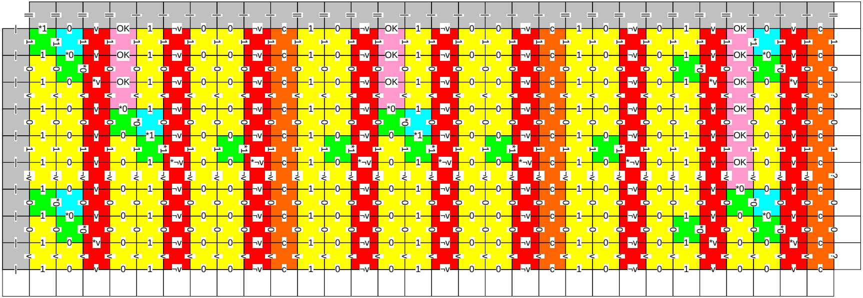
Solving 3-SAT

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



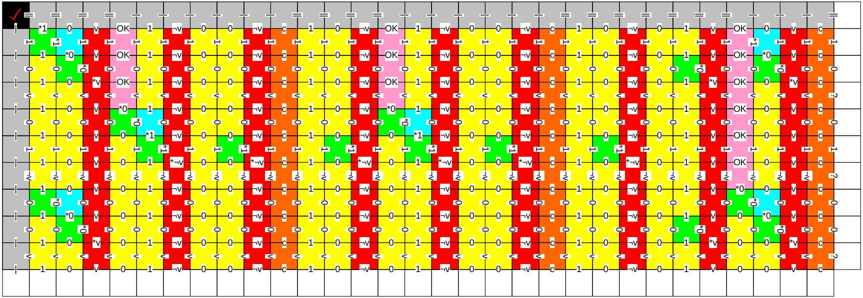
Solving 3-SAT

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



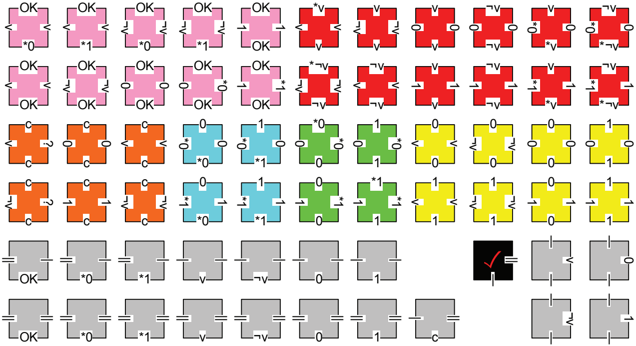
Solving 3-SAT

$$(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)$$



$\Theta(1)$-Tileset 3-SAT Solution [Bru08c]

- 64 tile types
- Probability of success $\geq \left(\frac{1}{2}\right)^n$



Improving the 3-SAT Algorithm Runtime

Some algorithms reduce the base of the exponent

Fastest known: $O^\star(1.3333^n)$ [Woe03].

An $O^\star(1.8393^n)$ algorithm [Woe03]

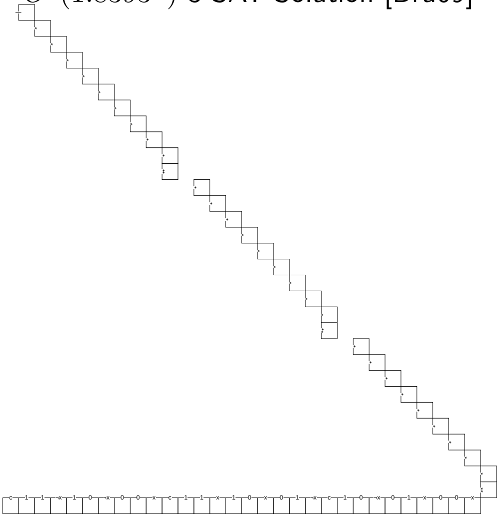
Suppose $\phi = (x_1 \vee \neg x_2 \vee x_3) \cdots$.
There are 3 relevant possibilities: either,

- first literal is $TRUE$, or
- first literal is $FALSE$ and second literal is $TRUE$, or
- first two literals are $FALSE$ and third literal is $TRUE$.

$$T(n,m) = c + \sum_{i=1}^{3} T(n-i, m-1) = O^\star(1.8393^n m).$$

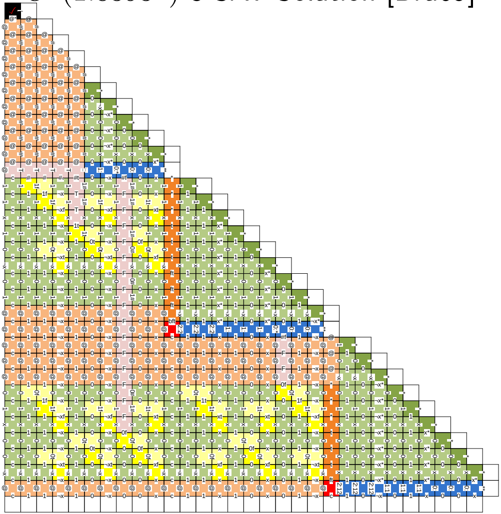
Can Tiles Implement More-Efficient Algorithms?

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



Can Tiles Implement More-Efficient Algorithms?

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



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

- 150 tile types
- Probability of success $\geq \left(\frac{1}{1.8393}\right)^n$

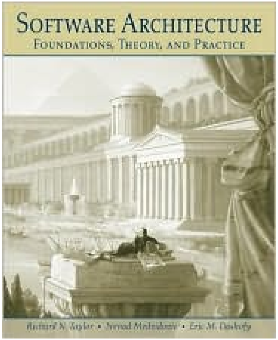
Efficient Tile Systems

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

Outline

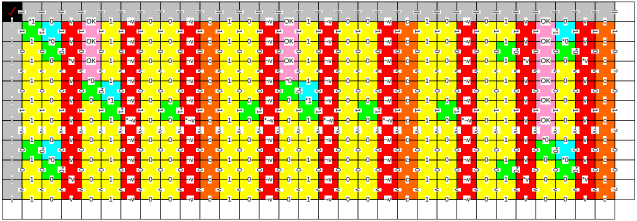
1. Why Nature?
2. Using Nature to Compute
3. Tiles
4. Tile Software
   - Leveraging Software Architecture to Build Tile-Inspired Software
   - Problem Statement: Private Computation
   - Tile Architectural Style
   - Tile Style Analysis
5. Conclusions

Software Architecture



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

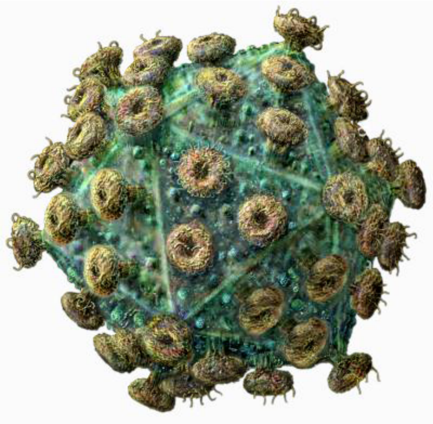
Converting the Model to an Architecture



Architectural Elements [MRMM02]

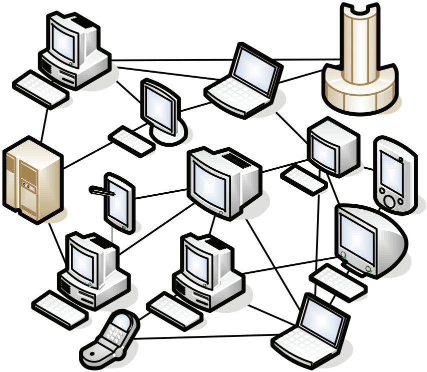
- Components: tiles
- Interfaces: side labels
- Topology: 2-D grid
- Behaviors: identifying nodes, recruiting attachments, replicating, and reporting the solution
- Interaction: recruitment data exchange

Computationally Intensive Problems



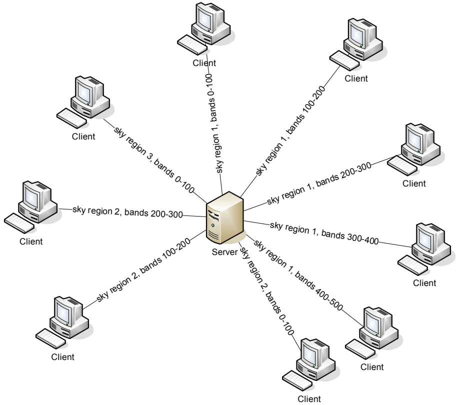
Internet as a Computing Medium

- Billion machines
- Mostly idle
- Insecure



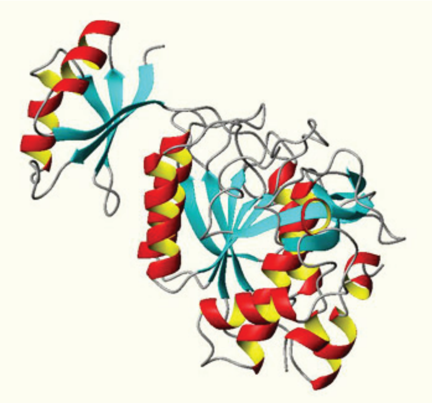
Distributed Computation

- 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

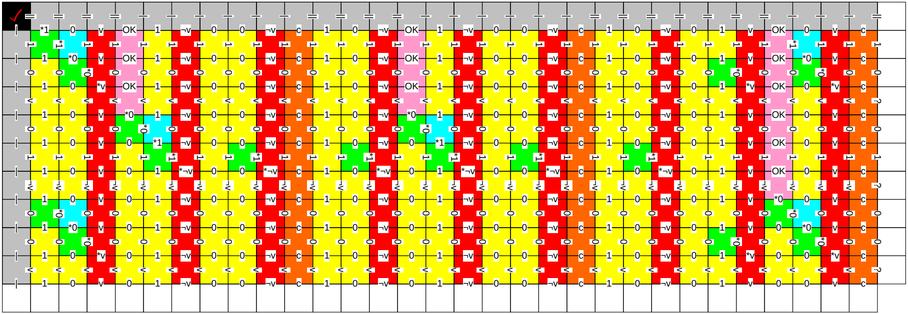


Example Scenario

- Possible cancer cure
- Find minimal-free-energy configuration
- Keep amino acid sequence private

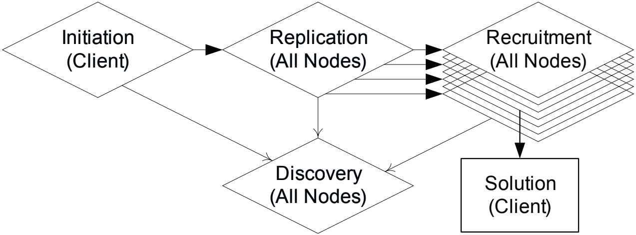


Tile Style Intuition



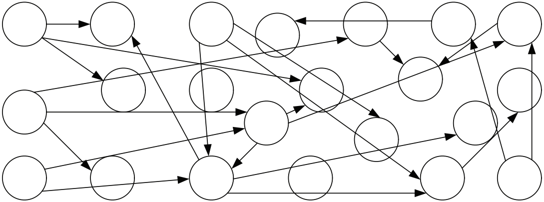
Node Operations [BM07a]

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



Node Discovery

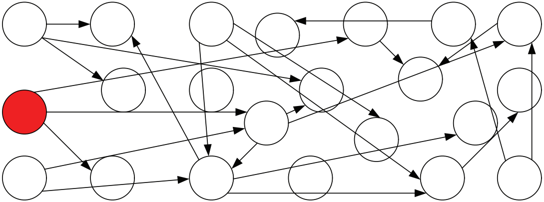
- Each node can return a randomly-uniform node of each tile component type



- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Node Discovery

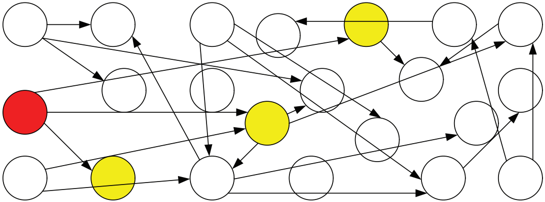
- Each node can return a randomly-uniform node of each tile component type



- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Node Discovery

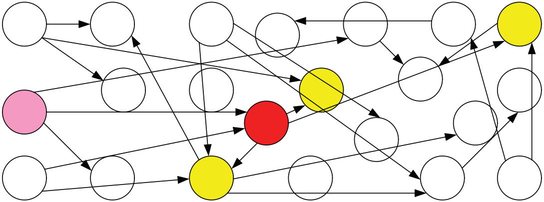
- Each node can return a randomly-uniform node of each tile component type



- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Node Discovery

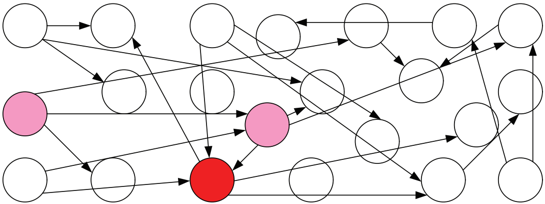
- Each node can return a randomly-uniform node of each tile component type



- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Node Discovery

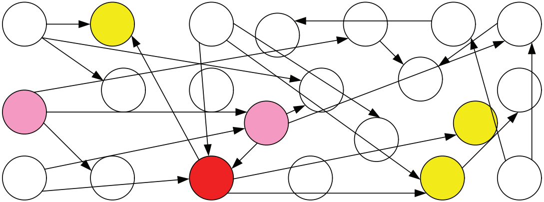
- Each node can return a randomly-uniform node of each tile component type



- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Node Discovery

- Each node can return a randomly-uniform node of each tile component type



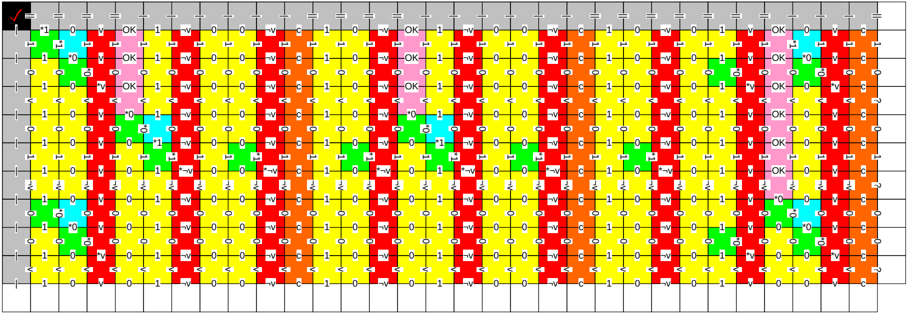
- Each node, for each tile type, keeps a list of 3 nodes that deploy that type
- When queried, a node returns one of the 3 elements at random, and replaces its list with that nodes list of 3

- Result: the algorithm returns a uniformly-random IP after only $\Theta(logN)$ requests [MR95]

Privacy Preservation

- Data
  1. Each node knows very little
  2. It is hard to control the entire input
- Algorithm
  3. One tile type implies nothing
  4. It is hard to learn all the tile types
  5. Knowing the tile types does not reveal the algorithm

Data: Each Node Knows Very Little



Less than 1 bit of information per tile

Data: It Is Hard to Control the Entire Input

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

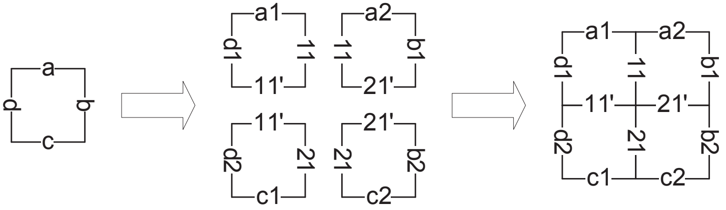
n — bits in input c — compromised fraction s — number of seeds

- TeraGrid $(100,000$ machines$)$
- 17-variable 100-clause 3-SAT problem

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

Fault-Tolerant Tile Style [BM07b]

- Tile systems can be designed to be tolerant to misbehaving tiles
- For example, [WB03]



Provably Correctable Errors

- Failing tiles
- Misbehaving tiles
- Byzantine tiles
- Service attacks
- Privacy attacks
- . . . probably many more

Tile Style Hypotheses

1. Speed $\propto$ network size
2. Robust to network delay
3. Can solve real-world-sized problems

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

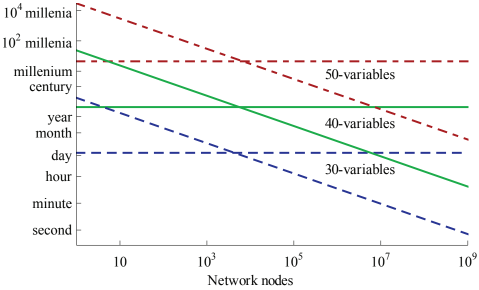
Scalability: Speed $\propto$ Network Size

Network & Problem	# of Nodes	Execution Time	Speed-up Ratio
Private Cluster	5	43.2 sec.	
$\mathfrak{A}$	10	22.9 sec.	1.89
HPCC	93	220 min.	
$\mathfrak{C}$	186	116 min.	1.90
PlanetLab	50	9.2 min.	
$\mathfrak{B}$	100	4.8 min.	1.92
Simjong	125,000	8.7 hours	
	250,000	4.5 hours	1.93
$\mathfrak{D}$	500,000	2.1 hours	2.14
	1,000,000	64 min.	1.97

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.

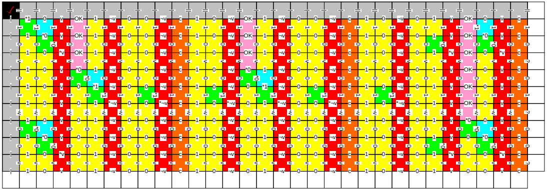
Efficiency: Solving Real-World-Sized Problems



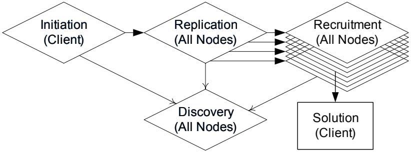
# of Nodes	Execution Time	
	Simjong	Theoretical Calculation
125,000	8.7 hours	9.1 hours
250,000	4.5 hours	4.5 hours
500,000	2.1 hours	2.3 hours
1,000,000	64 min	68 min.

Tile Style

- Developed self-assembling systems to solve complex computational problems



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



The Big Picture

Nature

- Bring forward novel, well-tested, well-scaling, robust mechanisms
- Present outside-the-box solutions

Software Architecture

- Facilitate translation of a nature-inspired model to software
- Aid design, implementation, and evaluation

Software Architecture III
Leveraging Nature to Build Better Systems

Yuriy Brun

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

Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss.
Amorphous computing.
Communications of the ACM, 43(5):74–82, May 2000.

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 M. Adleman.
An abstract theory of computer viruses.
In *Proceedings on Advances in Cryptology (CRYPTO88)*, pages 354–374, Santa Barbara, CA, USA, 1990.

Leonard Adleman.
Molecular computation of solutions to combinatorial problems.
Science, 266:1021–1024, 1994.

Leonard M. Adleman.
Computing with DNA.
Scientific American Magazine, pages 54–61, August 1998.

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.

Leonard M. Adleman and David Wofsy.

T-cell homeostasis: implications in HIV infection.
Journal of Acquired Immune Deficiency Syndromes, 6(2):133–152, February 1993.

Ravinderjit Braich, Nickolas Chelyapov, Cliff R. Johnson, Paul W. K. Rothemund, and Leonard Adleman.
Solution of a 20-variable 3-SAT problem on a DNA computer.
Science, 296(5567):499–502, 2002.

Yuriy Brun and Manoj Gopalkrishnan.
Toward in vivo disease diagnosis and treatment using DNA.
In *Proceedings of the 2006 International Conference on Bioinformatics & Computational Biology (BIOCOMP06)*, pages 182–186, Las Vegas, NV, USA, June 2006.

Yaakov Benenson, Binyamin Gil, Uri Ben-Dor, Rivka Adar, and Ehud Shapiro.
An autonomous molecular computer for logical control of gene expression.
Nature, 429:423–429, 2004.

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.

Yuriy Brun and Nenad Medvidovic.
Fault and adversary tolerance as an emergent property of distributed systems' software architectures.
In *Proceedings of the 2nd International Workshop on Engineering Fault Tolerant Systems (EFTS07)*, pages 38–43, Dubrovnik, Croatia, September 2007.

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.

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.

High performance computing and communications.

http://www.usc.edu/hpcc.

Abdurrahman Hacioglu and Ibrahim Ozkol.
Transonic airfoil design and optimisation by using vibrational genetic algorithm.
Aircraft Engineering and Aerospace Technology, 75(4):350–357, 2003.

Thomas F. Knight, Jr. and Gerald Jay Sussman.
Cellular gate technology.
Unconventional Models of Computation, pages 257–272, 1997.

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.

Michail G. Lagoudakis and Thomas H. LaBean.
2D DNA self-assembly for satisfiability.
DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 54:141–154, 1999.

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, George Edwards, Yuriy Brun, Hossein Tajalli, Joshua Garcia, Ivo Krka, Nenad Medvidovic, Marija Mikic-Rakic, and Gaurav Sukhatme.
An architecture-driven software mobility framework.
Journal of Systems and Software, In Press, 2010.

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.

Lulu Qian and Erik Winfree.
A simple DNA gate motif for synthesizing large-scale circuits.
In *In Proceedings of the 14th International Meeting on DNA Computing, (DNA08)*, pages 70–89, Prague, Czech Republic, June 2008.

Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, and Philip K. McKinley.
Applying genetic algorithms to decision making in autonomic computing systems.
In *Proceedings of the 6th International Conference on Autonomic Computing (ICAC06)*, pages 97–106, Barcelona, Spain, 2009.

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.

Ronald Linn Rivest, Adi Shamir, and Leonard Adleman.

A method for obtaining digital signatures and public-key cryptosystems.
Communications of the ACM, 21(2):120–126, 1978.

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.

M. Sinha, C. S. Kennedy, and M.L. Ramundo.
Artificial neural network predicts CT scan abnormalities in pediatric patients with closed head injury.
The Journal of Trauma, 50(2):308–312, 2001.

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 and Renat Bekbolatov.
Proofreading tile sets: Error correction for algorithmic self-assembly.
In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS02)*, volume 2943, pages 126–144, Madison, WI, USA, June 2003.

David West.
Neural network credit scoring models.
Computers & Operations Research, 27(11–12):1131–1152, 2000.

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.

Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest.
Automatically finding patches using genetic programming.
In *Proceedings of the ACM/IEEE 31st International Conference on Software Engineering (ICSE09)*, pages 364–374, Vancouver, Canada, 2009.

Gerhard J. Woeginger.
Exact algorithms for NP-hard problems: a survey.
Combinatorial Optimization - Eureka, You Shrink!, pages 185–207, 2003.