Software Architecture III
Leveraging Nature to Build Better Systems

Yuriy Brun

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

1. Why Nature?
2. Using Nature to Compute
3. Tiles
4. Tile Software
5. Conclusions
Outline

1. Why Nature?
2. Using Nature to Compute
3. Tiles
4. Tile Software
5. Conclusions
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]
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

Why Nature?
Nature Computes
Tiles
Tile Software
Conclusions

Hamiltonian Path Problem with DNA

[Cytology

From Nature to Software
CSEP 504 Winter 2010
15 / 62

[Adl98]
## Implementing the DNA Algorithm

<table>
<thead>
<tr>
<th>Conventional algorithm</th>
<th>DNA algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generate a set of random paths</td>
<td>Mix city and flight strands</td>
</tr>
<tr>
<td>2. Select paths that start and end at proper cities</td>
<td>PCR</td>
</tr>
<tr>
<td>3. Select proper-length paths</td>
<td>Electrophoresis gel</td>
</tr>
<tr>
<td>4. Select paths that visit each city</td>
<td>Watson &amp; Crick pairing</td>
</tr>
<tr>
<td>5. The remaining paths represent the solution</td>
<td>PCR, electrophoresis, and sequencing</td>
</tr>
</tbody>
</table>
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 the 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 \([\text{ACG}^+01]\)
  - squares \([\text{RW}00, \text{AGHM}02, \text{ACG}^+02]\)
  - computable shapes \([\text{SW}07]\)

- Count \([\text{Win}98a, \text{Moi}05, \text{BRW}05]\)

- Compute Binomial Coefficients \([\text{Win}98a, \text{RPW}04]\)

- Emulate Turing Machines \([\text{Win}98b]\)
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

502...
1010010010101...
22+11

33
Adding with Tiles [Bru07]
Adding with Tiles [Bru07]

\[
\begin{align*}
0 & \quad 0 \\
0 & \quad 0 \\
1 & \quad 0 \\
1 & \quad 0 \\

\end{align*}
\]

\[
\begin{align*}
1 & \quad 1 \\
0 & \quad 1 \\
0 & \quad 1 \\
1 & \quad 1 \\

\end{align*}
\]

\[
\begin{align*}
34 + 27
\end{align*}
\]
Adding with Tiles [Bru07]

\[ 34 + 27 = 61 \]
Multiplying with Tiles [Bru07]
Multiplying with Tiles [Bru07]

\[
87 \times 45
\]
Multiplying with Tiles [Bru07]

\[
\begin{array}{cccccccccc}
00 & 00 & 00 & 00 & 11 & 11 & 11 & 11 & 11 & 00 \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\
00 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
00 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
00 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
00 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
\end{array}
\]

\[
87 \times 45 = 3915
\]
3-SAT

Variables: \( x_0, x_1, x_2, \ldots \)

Literals: \( x_0, \neg x_0, x_1, \neg x_1, \ldots \)

Clauses: \((x_2 \lor \neg x_1 \lor \neg x_0)\)

Formula: \((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0)\)

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]

$$(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)$$
$\Theta(n^2)$-Tileset Approach [LL99]

\[
(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)
\]
\(\Theta(n^2)\)-Tileset Approach [LL99]

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
\( \Theta(n^2) \)-Tileset Approach [LL99]

\[
(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)
\]
$\Theta(n^2)$-Tileset Approach [LL99]
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

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

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\)
Solving 3-SAT

\((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\)
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\)
Θ(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: $O^*(1.3333^n)$ [Woe03].

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

Suppose $\phi = (x_1 \lor \neg x_2 \lor x_3) \cdots$.

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 $TRUE$.

$$T(n, m) = c + \sum_{i=1}^{3} T(n-i, m-1) = O^*(1.8393^n m).$$
Can Tiles Implement More-Efficient Algorithms?

\[ O^*(1.8393^n) \] 3-SAT Solution [Bru09]
Can Tiles Implement More-Efficient Algorithms?

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

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

![Diagram of tile types and their arrangement](image-url)
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

Yuriy Brun (brun@cs.washington.edu)
From Nature to Software
CSEP 504 Winter 2010
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
Why Nature?

Nature Computes

Tiles

Tile Software

Conclusions

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 $\Theta(\log N)$ 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 node's list of 3.

- Result: the algorithm returns a uniformly-random IP after only $\Theta(\log N)$ 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 $\Theta(\log N)$ 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 $\Theta(\log N)$
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 node's list of 3
- Result: the algorithm returns a uniformly-random IP after only $\Theta(\log N)$ 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 $\Theta(\log N)$ 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

\[ 1 - (1 - cn)^s \]

- \( n \) — bits in input
- \( c \) — compromised fraction
- \( s \) — number of seeds

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

<table>
<thead>
<tr>
<th>Compromised Fraction</th>
<th>Confidence Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{1}{8} )</td>
<td>( 1 - 10^{-10} )</td>
</tr>
<tr>
<td>( \frac{1}{4} )</td>
<td>( 1 - 10^{-5} )</td>
</tr>
<tr>
<td>( \frac{1}{3} )</td>
<td>( 1 - 10^{-3} )</td>
</tr>
</tbody>
</table>
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](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](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:
  - $\mathcal{A}$: 5-number 21-bit $SubsetSum$
  - $\mathcal{B}$: 11-number 28-bit $SubsetSum$
  - $\mathcal{C}$: 20-variable 20-clause 3-$SAT$
  - $\mathcal{D}$: 33-variable 100-clause 3-$SAT$
### Scalability: Speed $\propto$ Network Size

<table>
<thead>
<tr>
<th>Network &amp; Problem</th>
<th># of Nodes</th>
<th>Execution Time</th>
<th>Speed-up Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Cluster</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>5</td>
<td>43.2 sec.</td>
<td>1.89</td>
</tr>
<tr>
<td>$\mathcal{B}$</td>
<td>10</td>
<td>22.9 sec.</td>
<td></td>
</tr>
<tr>
<td>HPCC</td>
<td>93</td>
<td>220 min.</td>
<td>1.90</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>186</td>
<td>116 min.</td>
<td></td>
</tr>
<tr>
<td>PlanetLab</td>
<td>50</td>
<td>9.2 min.</td>
<td>1.92</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>100</td>
<td>4.8 min.</td>
<td></td>
</tr>
<tr>
<td>Simjong</td>
<td>125,000</td>
<td>8.7 hours</td>
<td>1.93</td>
</tr>
<tr>
<td>250,000</td>
<td>4.5 hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500,000</td>
<td>2.1 hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td>64 min.</td>
<td></td>
<td>1.97</td>
</tr>
</tbody>
</table>
## Robustness to Network Delay

<table>
<thead>
<tr>
<th>Problem</th>
<th># of Nodes</th>
<th>Network Delay</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahjong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>11</td>
<td>Private Cluster</td>
<td>20.1 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPCC</td>
<td>19.3 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlanetLab</td>
<td>18.5 sec.</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>Private Cluster</td>
<td>41.6 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPCC</td>
<td>41.2 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlanetLab</td>
<td>43.9 min.</td>
</tr>
<tr>
<td>Simjong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1,000,000</td>
<td>0ms</td>
<td>65 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10ms</td>
<td>57 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100ms</td>
<td>64 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500ms</td>
<td>60 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gaussian</td>
<td>68 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distance-based</td>
<td>59 min.</td>
</tr>
</tbody>
</table>
Efficiency: Solving Real-World-Sized Problems

<table>
<thead>
<tr>
<th># of Nodes</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simjong</td>
</tr>
<tr>
<td>125,000</td>
<td>8.7 hours</td>
</tr>
<tr>
<td>250,000</td>
<td>4.5 hours</td>
</tr>
<tr>
<td>500,000</td>
<td>2.1 hours</td>
</tr>
<tr>
<td>1,000,000</td>
<td>64 min</td>
</tr>
</tbody>
</table>

Yuriy Brun (brun@cs.washington.edu)
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.

Leonard Adleman, Qi Cheng, Ahish Goel, Ming-Deh Huang, and Hal Wasserman.
Linear self-assemblies: Equilibria, entropy, and convergence rates.

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.

Leonard M. Adleman.
An abstract theory of computer viruses.

Leonard Adleman.
Molecular computation of solutions to combinatorial problems.

Leonard M. Adleman.
Computing with DNA.

Leonard Adleman, Ashish Goel, Ming-Deh Huang, and Pablo Moisset de Espanés.
Running time and program size for self-assembled squares.

Leonard M. Adleman and David Wofsy.
T-cell homeostasis: implications in HIV infection.

Solution of a 20-variable 3-SAT problem on a DNA computer.

Yuriy Brun and Manoj Gopalkrishnan.
Toward in vivo disease diagnosis and treatment using DNA.

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

Yuriy Brun and Nenad Medvidovic.
An architectural style for solving computationally intensive problems on large networks.

Yuriy Brun and Nenad Medvidovic.
Fault and adversary tolerance as an emergent property of distributed systems’ software architectures.

Yuriy Brun.
Arithmetic computation in the tile assembly model: Addition and multiplication.

Yuriy Brun.
Nondeterministic polynomial time factoring in the tile assembly model.
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.  

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.  

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.  

**Arjav J. Chakravarti and Gerald Baumgartner.**

The organic grid: Self-organizing computation on a peer-to-peer network.  

**Jeffrey Dean and Sanjay Ghemawat.**

Mapreduce: Simplified data processing on large clusters.  

High performance computing and communications.
Abdurrahman Hacioglu and Ibrahim Ozkol.
Transonic airfoil design and optimisation by using vibrational genetic algorithm.

Thomas F. Knight, Jr. and Gerald Jay Sussman.
Cellular gate technology.

Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky.
SETI@home — massively distributed computing for SETI.

Michail G. Lagoudakis and Thomas H. LaBean.
2D DNA self-assembly for satisfiability.

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*.

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.

Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic.
A style-aware architectural middleware for resource-constrained, distributed systems.

Pablo Moisset de Espanés.
Computerized exhaustive search for optimal self-assembly counters.

Rajeev Motwani and Prabhakar Raghavan.
*Randomized Algorithms.*

Marija Mikic-Rakic, Nikunj R. Mehta, and Nenad Medvidovic.
Architectural style requirements for self-healing systems.

Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe.
A blueprint for introducing disruptive technology into the Internet.

Lulu Qian and Erik Winfree.
A simple DNA gate motif for synthesizing large-scale circuits.

Applying genetic algorithms to decision making in autonomic computing systems.

Rosetta@home.

Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree.
Algorithmic self-assembly of DNA Sierpinski triangles.

Ronald Linn Rivest, Adi Shamir, and Leonard Adleman.
A method for obtaining digital signatures and public-key cryptosystems.

Paul W. K. Rothemund and Erik Winfree.
The program-size complexity of self-assembled squares.

Artificial neural network predicts CT scan abnormalities in pediatric patients with closed head injury.

David Soloveichik and Erik Winfree.
Complexity of self-assembled shapes.

Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy.
*Software Architecture: Foundations, Theory, and Practice*.

Erik Winfree and Renat Bekbolatov.
Proofreading tile sets: Error correction for algorithmic self-assembly.

David West.
Neural network credit scoring models.

Erik Winfree.
*Algorithmic Self-Assembly of DNA*.

Erik Winfree.
Simulations of computing by self-assembly of DNA.

Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest.
Automatically finding patches using genetic programming.

Gerhard J. Woeginger.
Exact algorithms for NP-hard problems: a survey.