Automated Self-Assembly Programming Paradigm: Initial Investigation

Lin Li, Natalio Krasnogor, Jon Garibaldi
ASAP Group,
University of Nottingham.
{lxl, nxk, jmg}@cs.nott.ac.uk

asap research
automated scheduling optimisation & planning

Outline

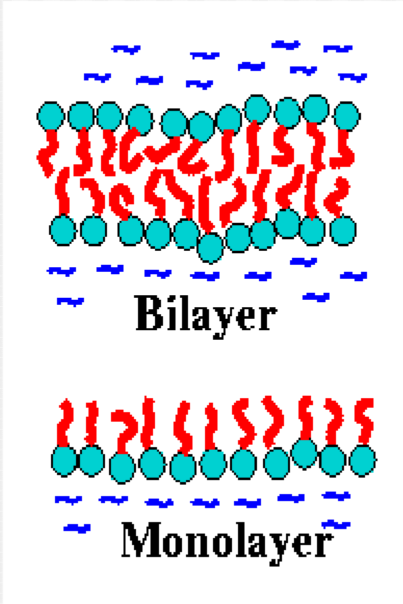
- Brief introduction on self-assembly and software self-assembly
- Automated self-assembly programming paradigm using ideal gas as a metaphor
- Experimental results and analysis
- Conclusion

What is self-assembly?

- definition:
  - Self-assembly is a process in which a disordered set of components self-organises into a specific structure.
  - Components interact with each other and form the global structure without external control.
  - The final structure is 'encoded' in the properties of components and their interactions.
- advantages:
  - Robust: replacement of failed components.
  - Versatile: prone to alternative specification.

Examples of self-assembly systems

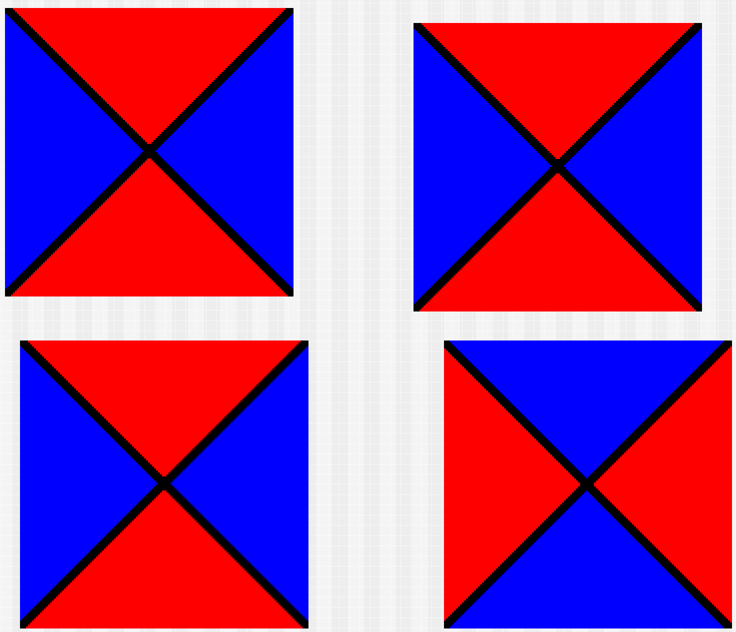
- Amphiphilic molecules
  - Formed by two ends with opposite properties:
    - Hydrophilic head: tend to be close to small water molecules
    - Hydrophobic end: tend to be close to similar chains
  - Amphiphilic molecules self-assemble into a variety of structures in water.



Examples of self-assembly systems

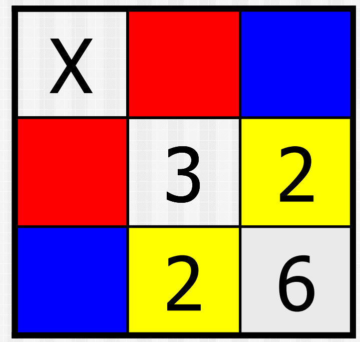
- Wang tiles model

G(blue, blue) > temp - -> tiles stick and standstill



G(blue, red) <= temp -- > tiles do not stick

Stickiness



Temp = 4

Automated Self-assembly programming paradigm (ASAP^2)

- Our automated self-assembly programming paradigm (ASAP^2) is inspired by both natural and artificial self-assembly systems.
- Software self-assembly system features:
  - human-made software components
  - software repositories
  - interaction rules
  - embodiment metaphor
  - no external intervention or central control mechanism

Software self-assembly in relation to GP

- Genetic programming: One of the most popular approaches to automated program synthesis and has been applied to wide range of problems.
- Software self-assembly seeks to provide at least a complement but maybe an ***alternative*** to genetic programming.
- GP uses natural selection as a metaphor.

This Research

- We aim at analysing the potential and limitation of software self-assembly.
- In this talk, we aim to find out how different environment settings affect an *unguided* process of software self-assembly.
- The embodiment we use is a metaphor based on ideal gas theory.

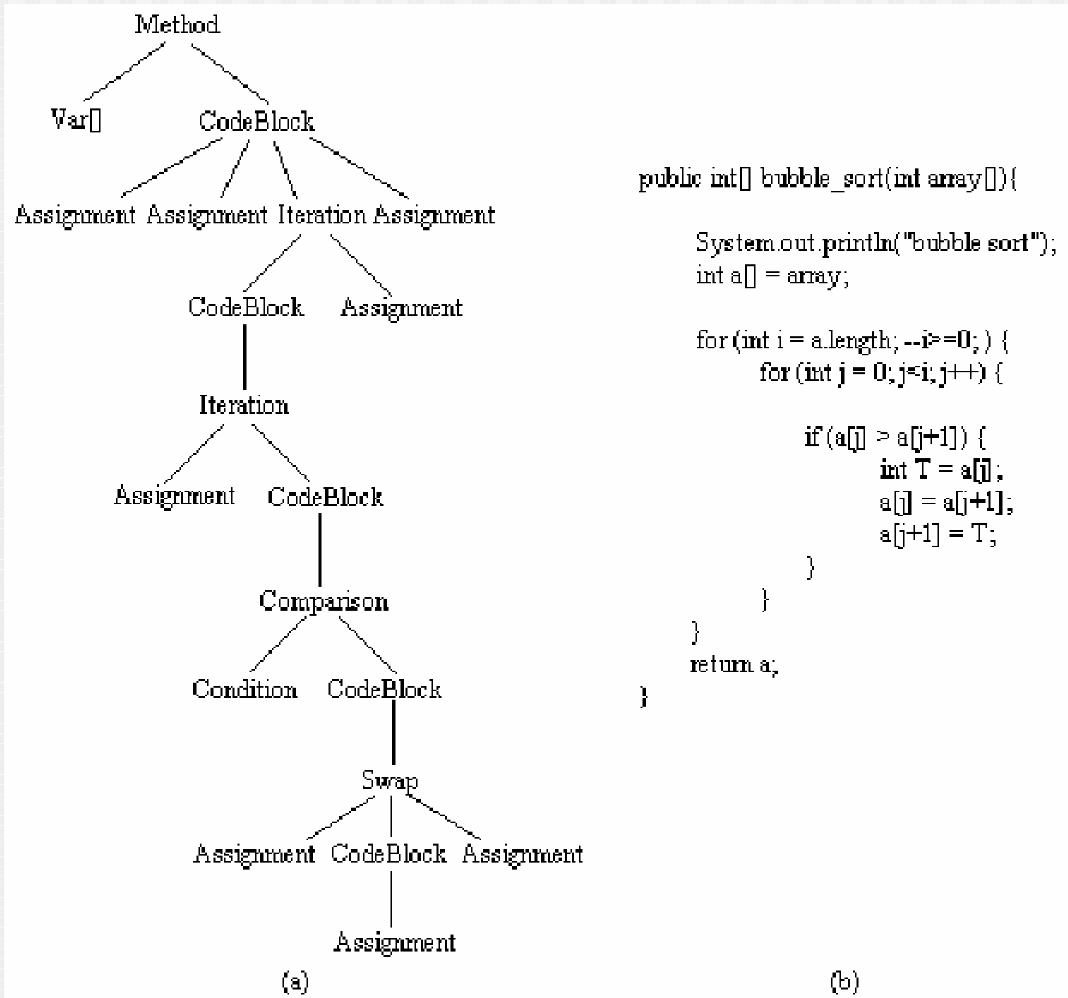
Program gases

- Our software self-assembly system is based on the theory of perfect gases.
  - Manually decomposed software components are placed into a container within which they move randomly.
  - The temperature (T), number of components placed into the pool (n), and the size of the pool (V) are free parameters of the model.
  - Components move faster as temperature increases.
- $PV = nRT$

Program gases

- What the metaphor does NOT capture:
  - Software self-assembly may bind rather than collide.
  - The size of an assembled gas component grows.
  - In perfect gases, it is assumed that the distance between molecules are much greater than their sizes.
- We investigate to what extent the equation for perfect gases holds for program gases.

A parsing tree of a bubble sort program



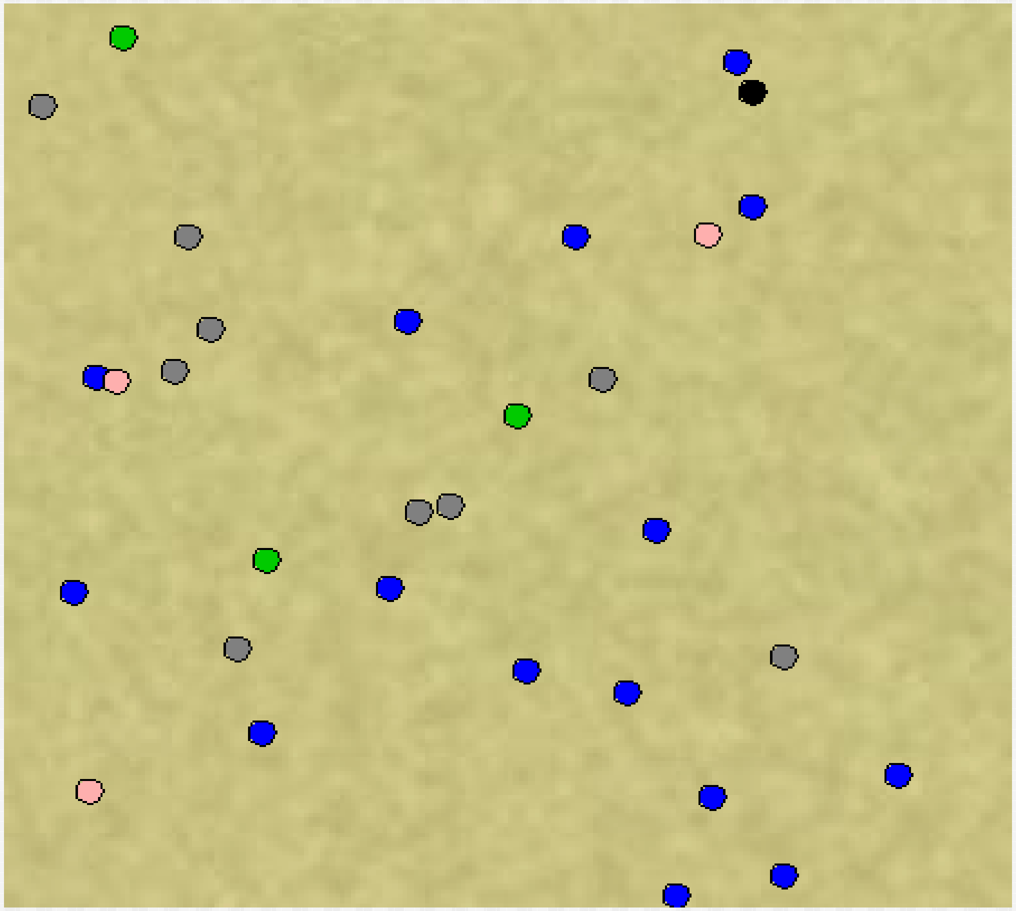
Components and ports

- Components are decomposed from a selected program and stored into a software repository.
- Each component has one input port and can have several output ports.
  - A port is used as a binding site.
  - Each port has a type associated.
- An input port can only connect with an output port of the same type.

Model descriptions

- ASAP^2 starts by placing components retrieved from the repository into the pool.
- Components move around in the pool randomly with a probability which is a function of the temperature.
- When two components are within certain proximity and their types on the connecting ports match, they self-assemble.
- Equilibrium is reached when there are **no more possible binding actions** among components left in the pool.

A gif animation of our system



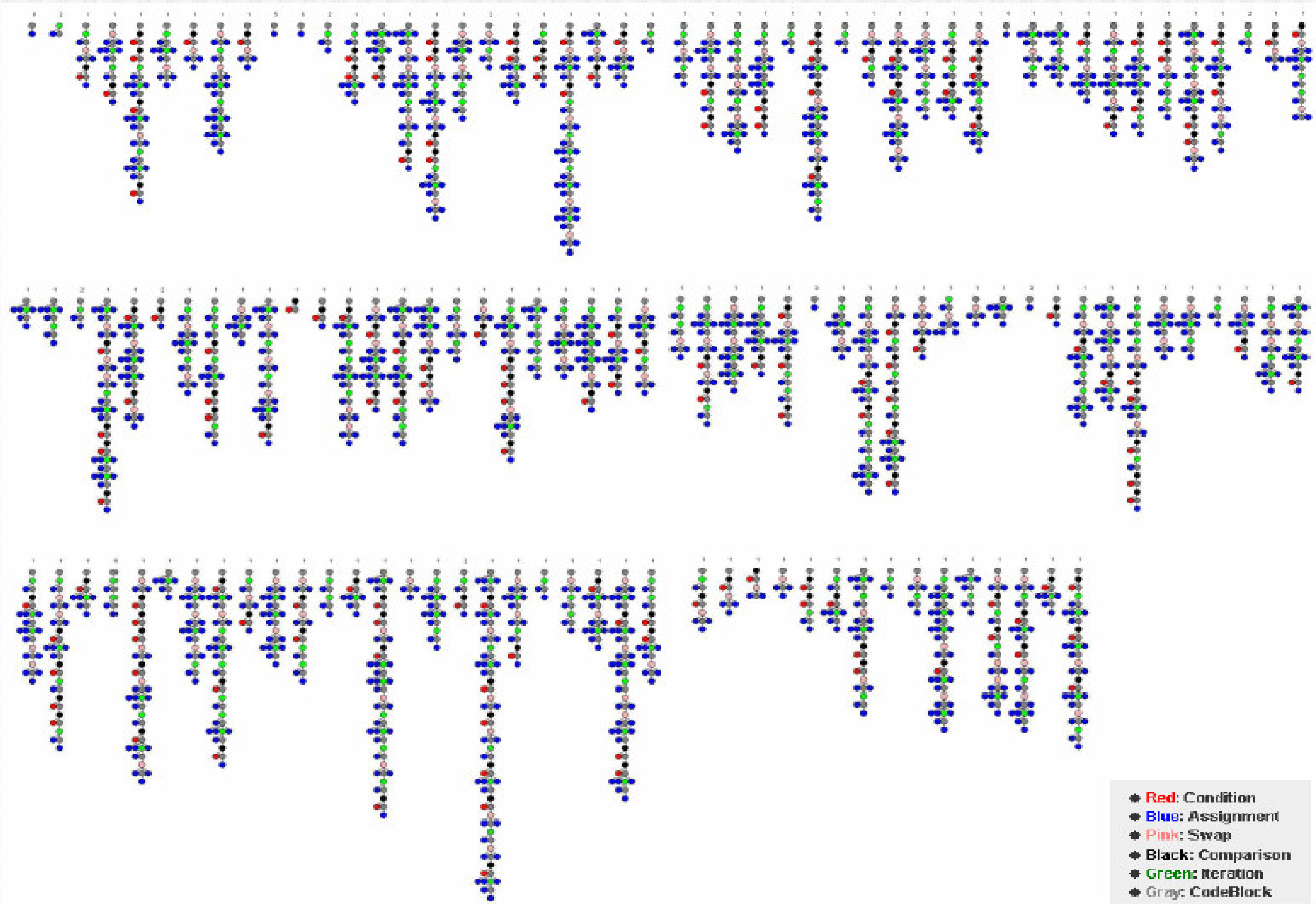
Experiment methods

- Moving probability of a component is affected by temperature: $p(M) = e^{-T}$
- Interaction distance between components is affected by the size of the component (i.e. the number of nodes in the tree).
- With three free parameters T, A and n, we measure:
  - pressure (P): number of hits on the wall
  - time to equilibrium (t_ε): time needed for the system to reach equilibrium
  - diversity of the self-assembled trees at equilibrium (D_ε): total number of different parse tree classes.

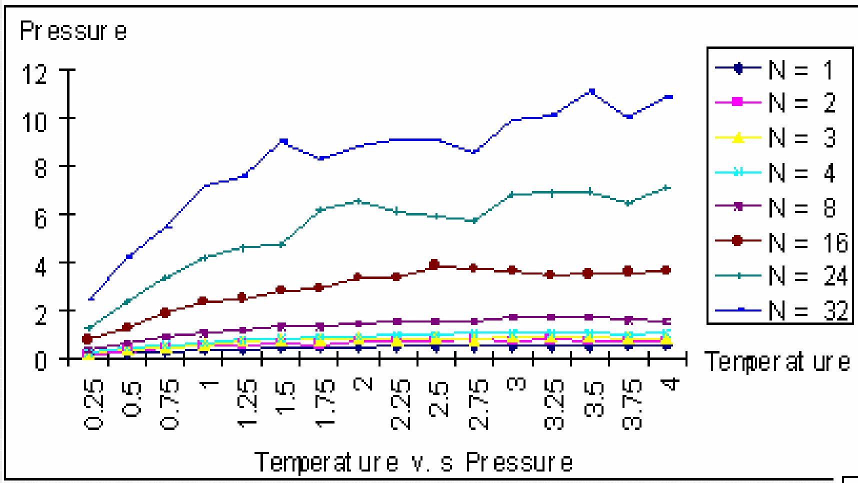
Experiment methods

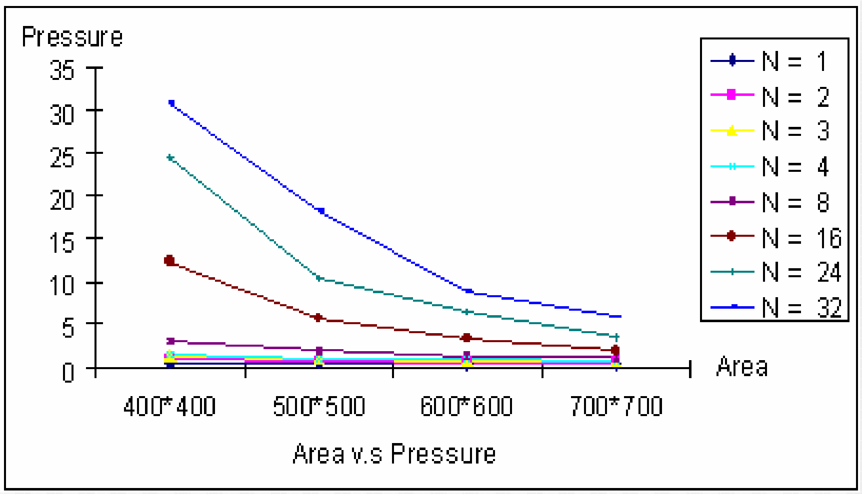
- Three standard sorting algorithms are chosen as sources of software components.
- For each of software component repository, we run 20 replicas for each (A, T, n) triplet.
- $A \in \{400, 500, 600, 700\}$, $T \in \{0.25, 0.5, \ldots, 3.75, 4.0\}$, $n \in \{1, 2, 3, 4, 8, 16, 24, 32\}$.

An example "forest" of program trees

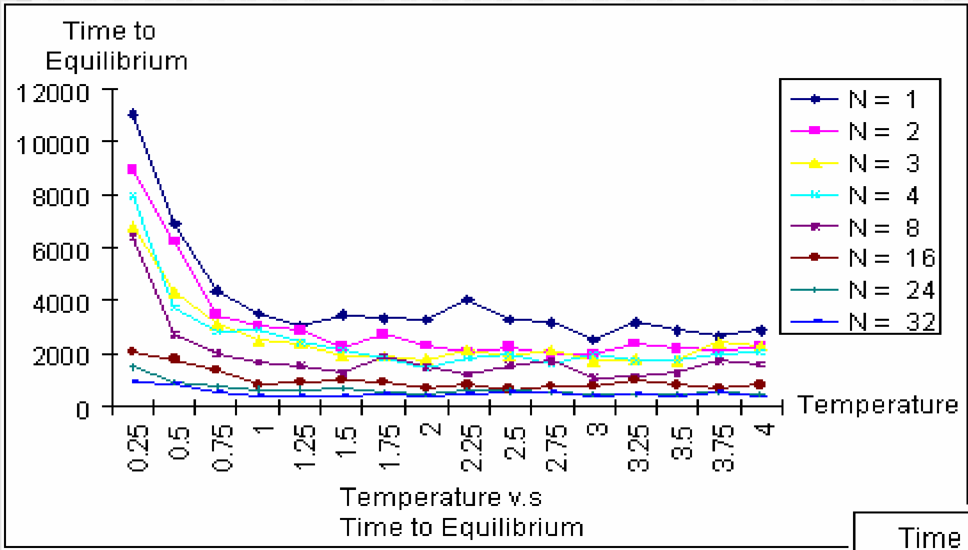


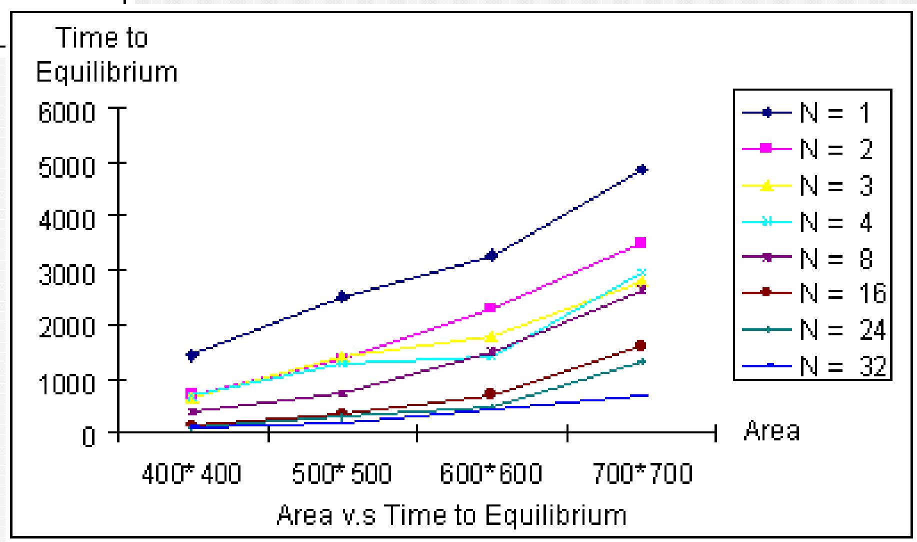
Experiment results (bubble sort)



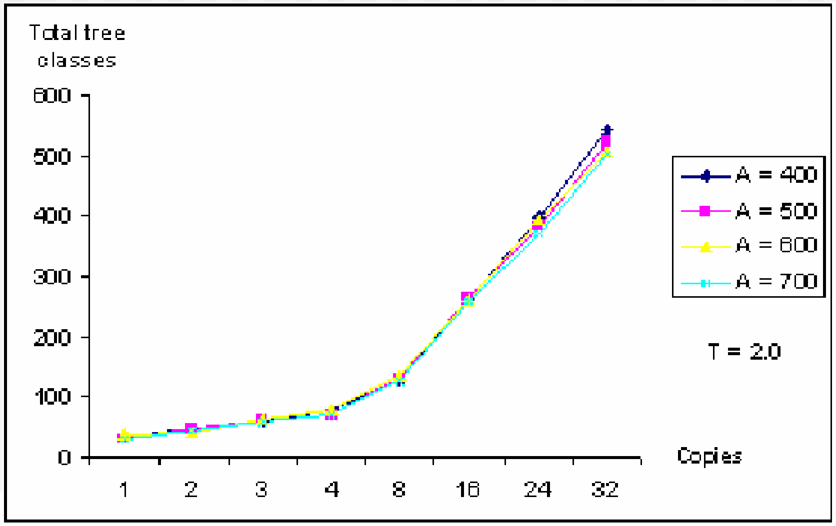


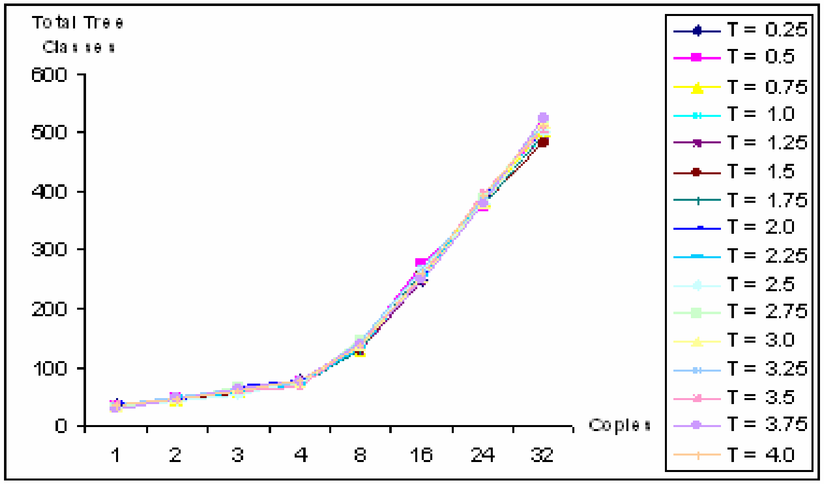
Experiment results (bubble sort)



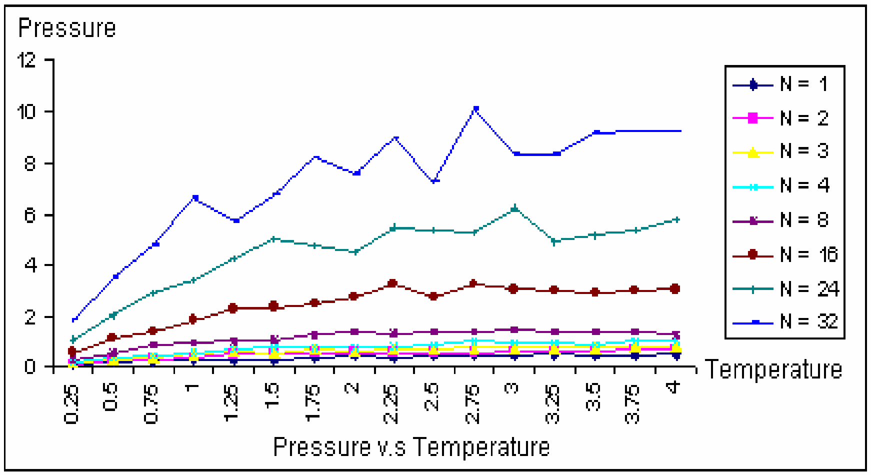


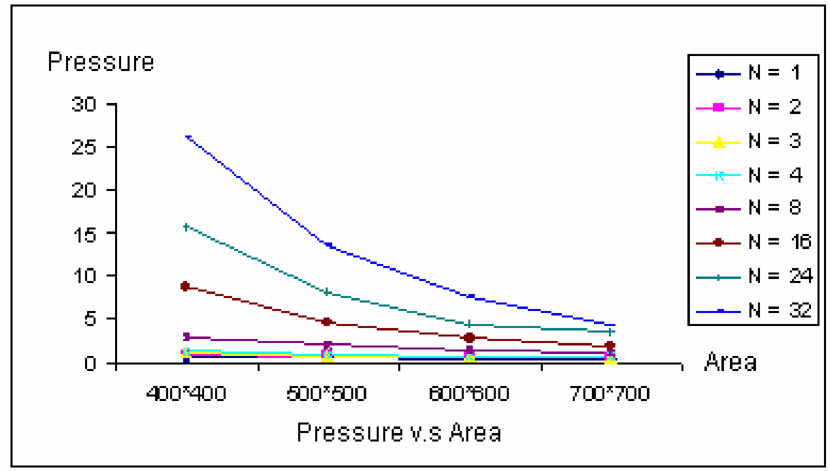
Experiment results (bubble sort)

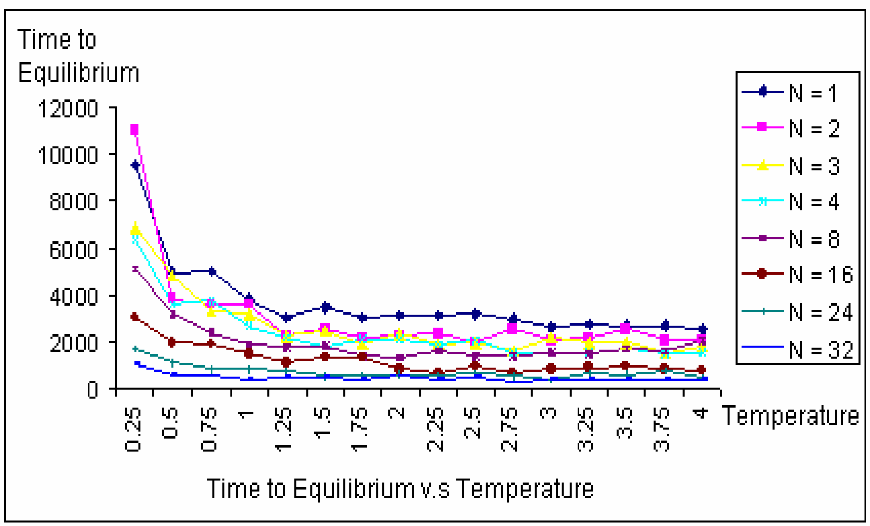


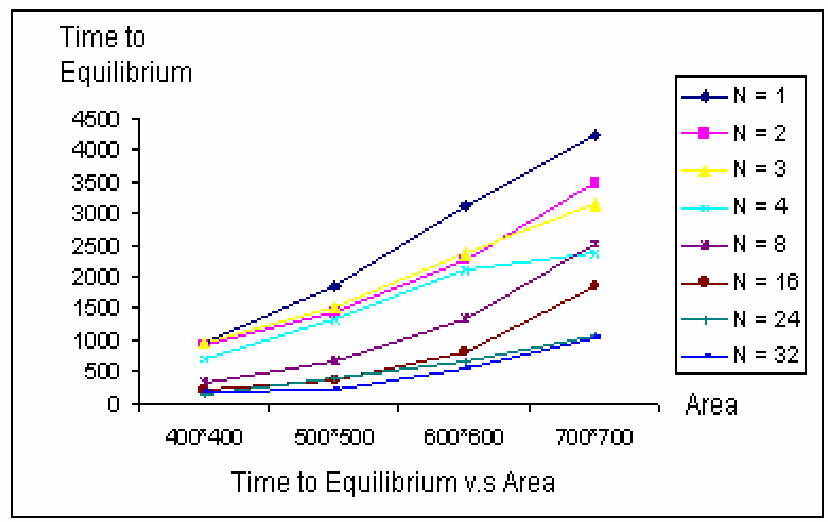


Experiment results (insertion sort)

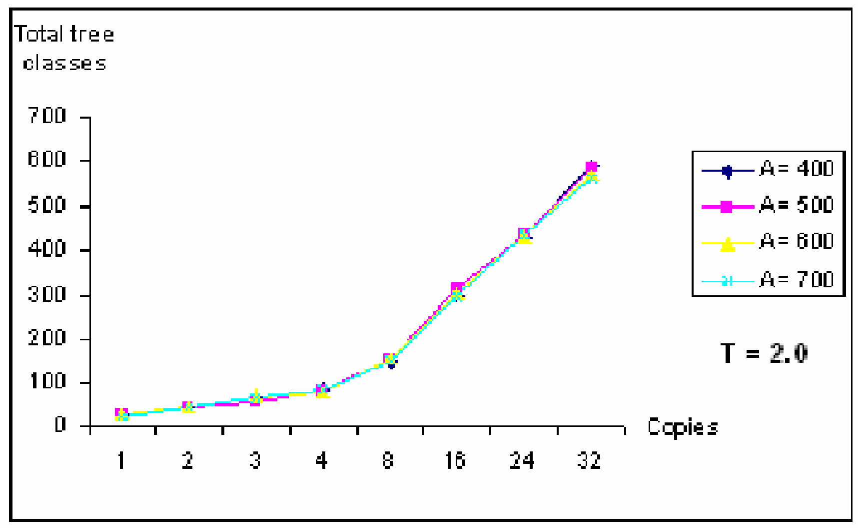


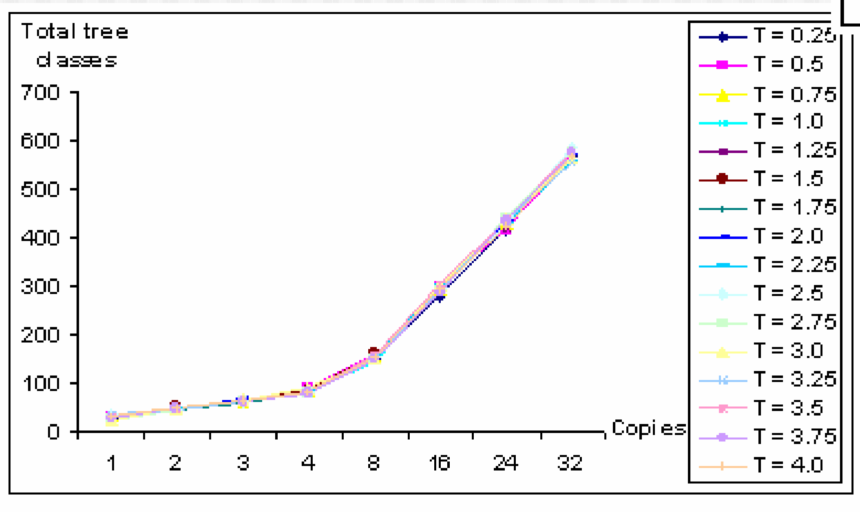




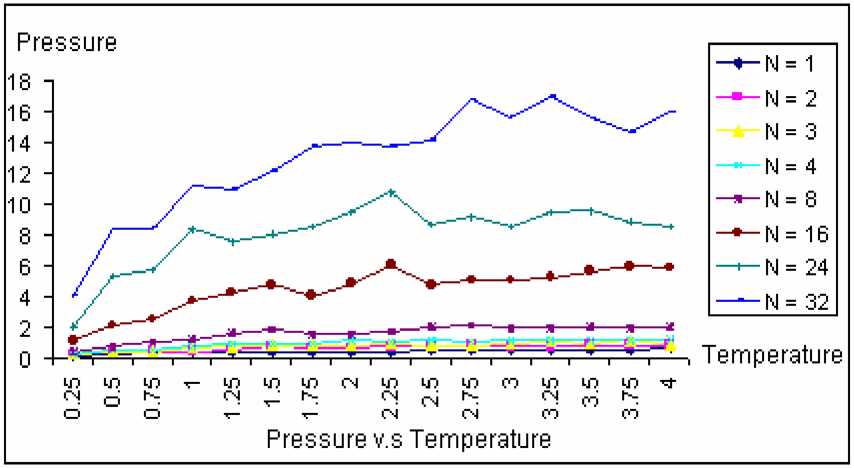


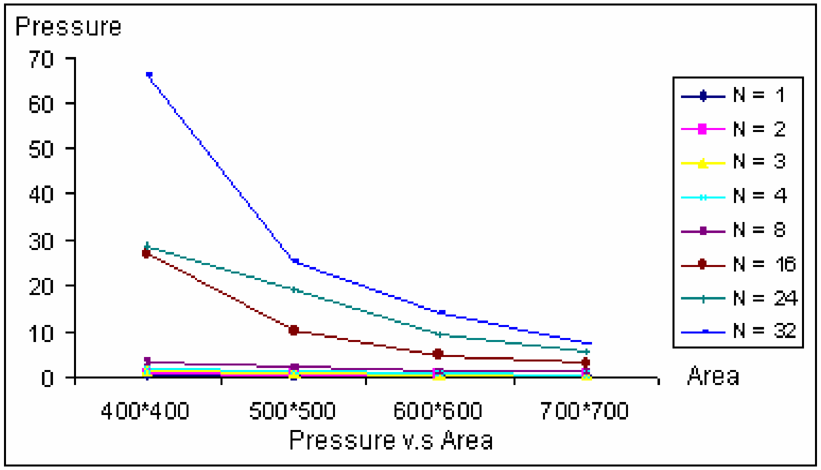
Experiment results (insertion sort)

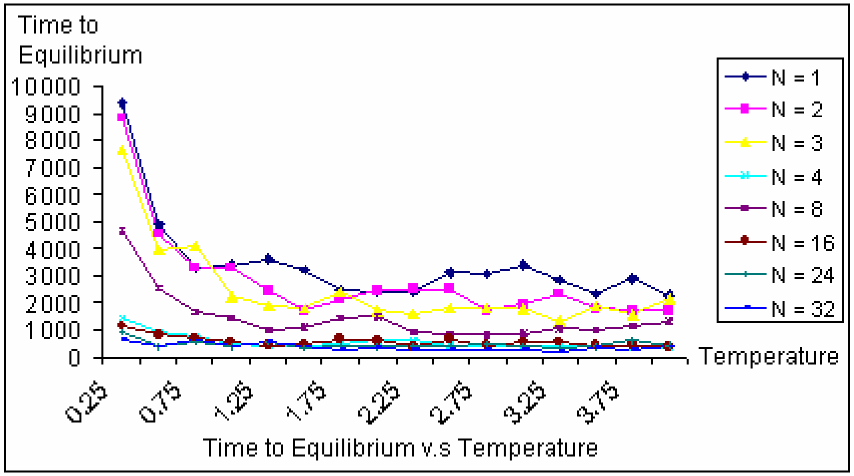


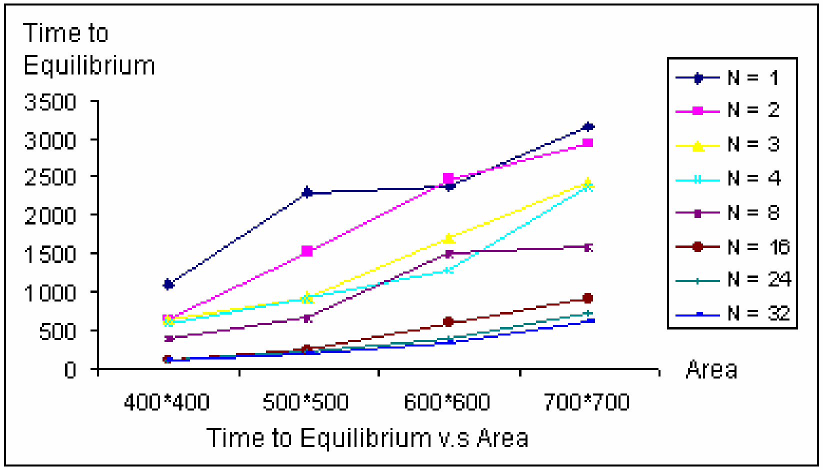


Experiment results (selection sort)

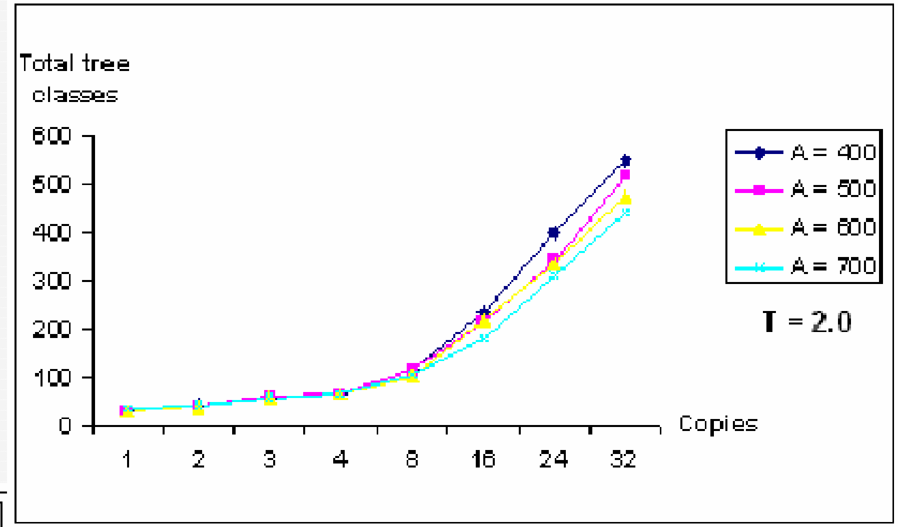


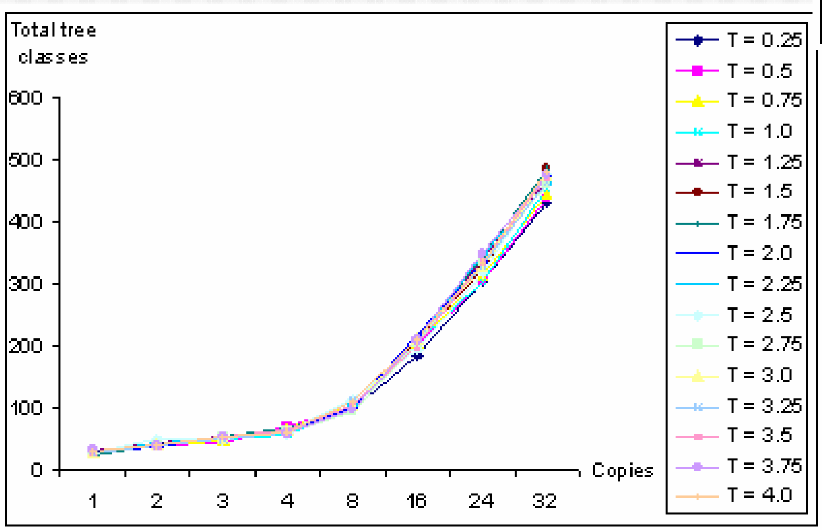






Experiment results (selection sort)



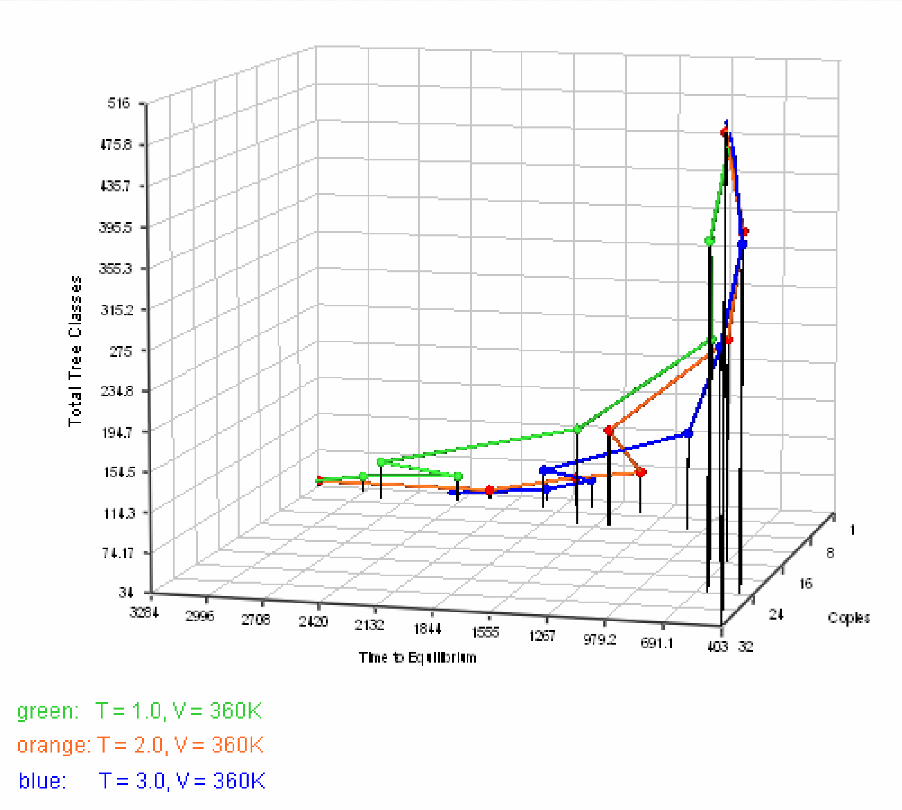


Experiment analysis

- As the equation of ideal gases suggests:
  - Pressure increases when there is a rise in temperature or number of components placed into the pool.
  - Pressure decreases when size of the pool becomes larger.
- Self-assembly can be made more efficient with:
  - Greater number of components placed into the pool.
  - Smaller size of the pool.
  - High temperature.

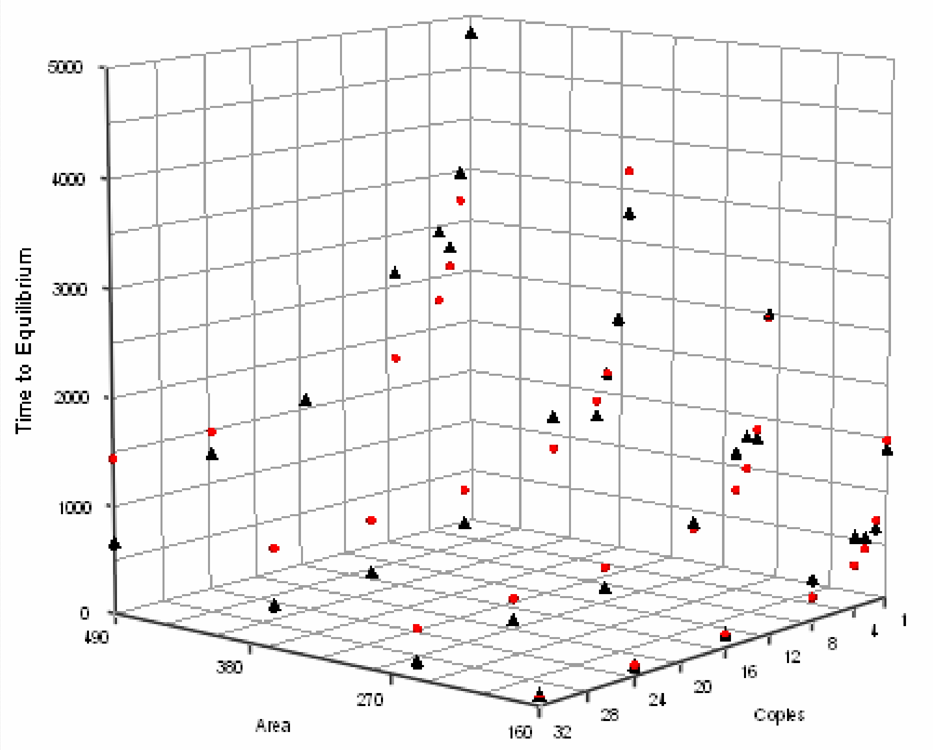
Experiment analysis

- Diversity of the generated programs are mainly affected by number of components placed into the pool.



Predictive formulae

- $t_\varepsilon(A,N) = (((7.05 * A) + 321.2) / N) +(3.91 * A) – 593.71$

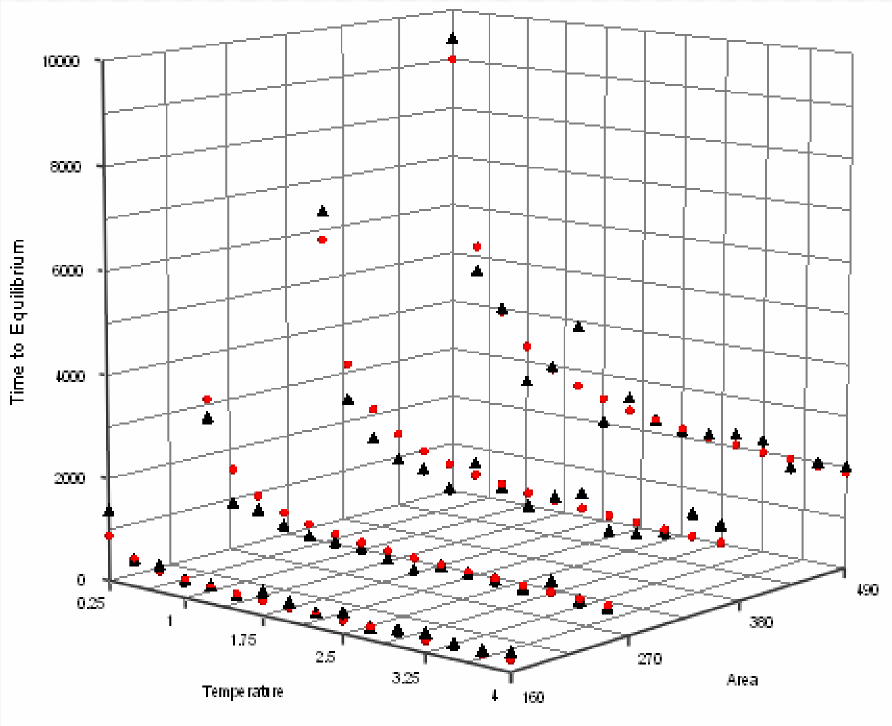


black triangles: experimental data;

red circles: corresponding data obtained using predictive model.

Predictive formulae

- $t_{\varepsilon}(A,T) = (((5.22 * A) – 659.02) / T) +(3.73 * A) – 416.11$

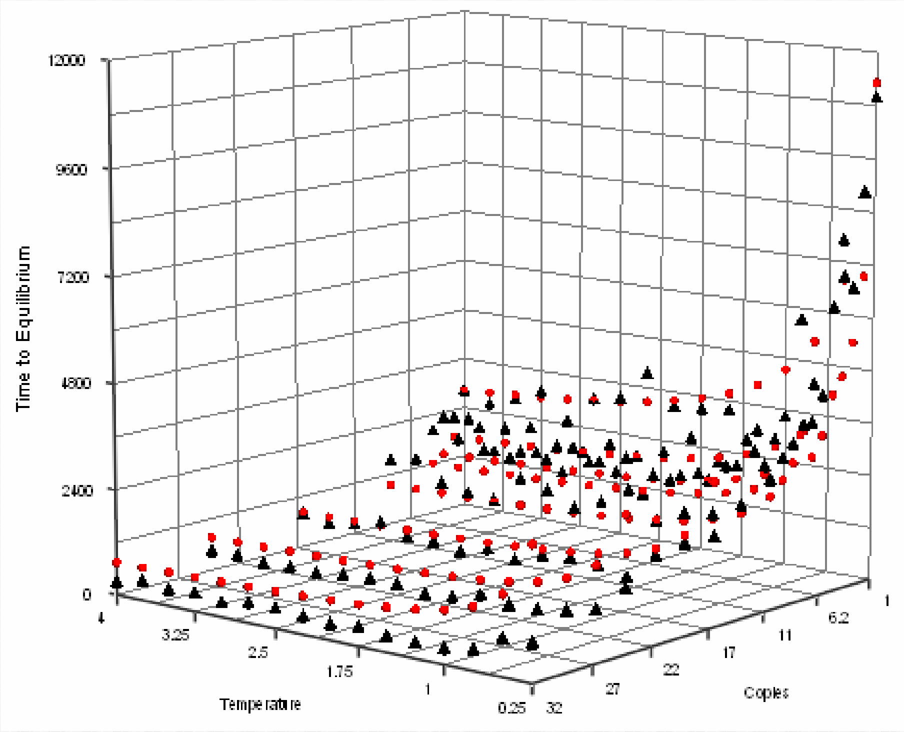


black triangles: experimental data;

red circles: corresponding data obtained using predictive model.

Predictive formulae

- $t_{\varepsilon}(T, N) = ((1/T) + 1) * ((1726/N) + 637.33)$

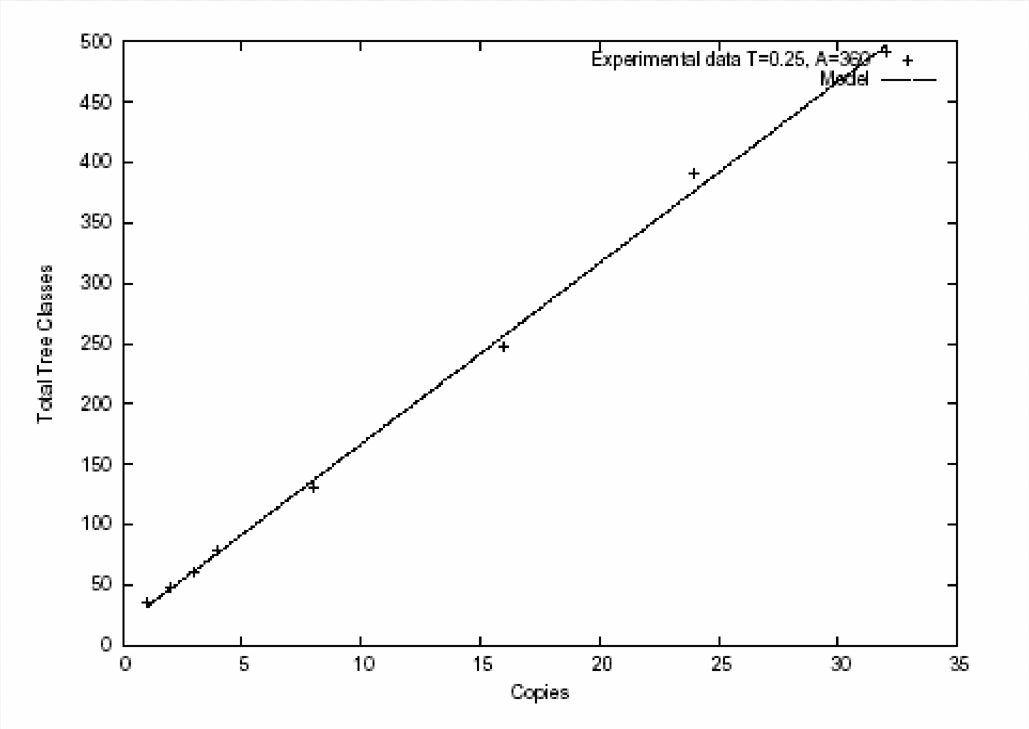


black triangles: experimental data;

red circles: corresponding data obtained using predictive model.

Predictive formulae

- $D_\varepsilon = 15*N + 16.6$



Prediction model assessment

equation	category	average error rate	standard deviation
Eq. 2	A=160	0.208	0.148
	A=250	0.425	0.525
	A=360	0.389	0.473
	A=490	0.261	0.374
Eq. 3	A=160	0.242	0.145
	A=250	0.152	0.146
	A=360	0.192	0.098
	A=490	0.097	0.090
Eq. 4	N=1	0.106	0.106
	N=2	0.129	0.089
	N=3	0.181	0.108
	N=4	0.263	0.095
	N=8	0.224	0.144
	N=16	0.219	0.189
	N=24	0.733	0.207
	N=32	1.190	0.521
Eq. 5	N/A	0.039	0.010

Conclusion

- In this talk, we have:
  - introduced an unguided software self-assembly system.
  - used kinetic theory on perfect gases as a metaphor for embodying $ASAP^2$.
  - from the experiments, we measured efficiency of software self-assembly and diversity of the generated programs.
  - obtained a suitable range of environment settings.
  - produced predictive equations for T_ε and D_ε
  - this prediction has not been done yet for GP.

Thank you!

Questions?