Distributed Genetic Programming Framework

The DGPF-Project
A presentation for interested students
Index

- What is the DGPF-project?
- Task Distribution System
- Genetic Algorithms
- Genetic Algorithm Layer
- Suggested strategy to avoid getting stuck: Hysteresis
- Automaton Simulation
- Network Simulation Layer
- Fitness Functions
- Examples of tasks open for students
- Things that I expect
What is the DGPF-project?

- “One Thousand Monkey Framework”
- Genetic Programming Framework
- Target: Engineer Programs for small, interlinked devices like Sensor Nodes genetically
- The DGPF is a distributed application itself
- Open Source, LGPL-Licensed, SourceForge-Project
- CVS-Repository: ext:user-name@cvs.sourceforge.net:/cvsroot/dgpf
- http://dgpf.sourceforge.net/
- http://sourceforge.net/projects/dgpf/
What is the DGPF-project?

- The DGPF-Framework employs a layered approach

<table>
<thead>
<tr>
<th>Genetic Programming Layer</th>
<th>Genetic Algorithms Layer</th>
<th>Task Distribution System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Simulation</td>
<td>Automaton Simulation</td>
<td>Basic Operations</td>
</tr>
<tr>
<td>Communication Operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

additional operations for communication
provides a virtual network

the basic operations of an automaton
provides the virtual hardware to simulate an automaton

supports arbitrary Genetic Algorithms
provides multiple selection algorithms

gathers statistical data and controls the evolution

distributes arbitrary tasks over a heterogeneous network
provides load balancing
provides error recovery
Task Distribution System

- Tasks are distributed from one client to several task-processing servers.
- Load Balancing: Servers may differ in speed, links may differ in bandwidth, tasks may differ in complexity.
- Fault-Tolerance, Error Recovery
Genetic Algorithms

- Genetic Programming bases on Genetic Algorithms
- Genetic Algorithms = heuristic search in (infinite) large spaces

**Initial Population**
create an initial population of random individuals

**Evaluation**
compute the fitness values of the individuals

**Control**
gather statistical data and check if we found a good enough solution to stop

**Reproduction**
create new individuals from the selected ones by crossover and mutation

**Selection**
select the fittest individuals for reproduction
Genetic Algorithm Layer

- Distribute Evaluation and Reproduction using the Task Distribution System
- GA-Engine of DGPF is multi-purpose: just need to define reproduction operators and fitness functions
Genetic Algorithm Layer

- Allows multi-objective Genetic Algorithms (*in principle*)
- Automatically gathers rich (and extensible) statistical information on the fitness of:
  - Individuals resulting from a mutation
  - Individuals resulting from a crossover
  - The whole population
  - The modified fitness
  - Minimum, median, average, maximum, variance
  - On each single fitness function and on the composed fitness
Genetic Algorithm Layer

- To control the evolution, you can change at runtime:
  - Selection algorithm
  - Parameters of the selection algorithm
  - Mutation rate
  - An enforced zone of death
  - A replication limit for this zone of death
  - Should the best-ever individual be injected in the zone of death?
  - Hysteresis algorithm

- Either set these parameters automatically or use the static informations for automatic adjustments.
Suggested strategy to avoid getting stuck: Hysteresis

- Evolution might get stuck at local optimum

```c
while(59)
{
  if(HAS_WORD)
  {
    mem[0]=RECEIVE_WORD
  }
  SEND_WORD (mem[0]+1)
  SEND_MESSAGE
}
```

```
while(59)
{
  if(HAS_WORD)
  {
    mem[1]=RECEIVE_WORD
  }
  SEND_WORD (mem[0]+1)
  SEND_MESSAGE
}
```

```
while(59)
{
  if(HAS_WORD)
  {
    mem[1]=RECEIVE_WORD
  }
  SEND_WORD mem[0]
  SEND_MESSAGE
}
```

```
while(59)
{
  if(HAS_WORD)
  {
    mem[1]=RECEIVE_WORD
    if(mem[1]==mem[0])
    {
      mem[0]=(mem[1]+1)
    }
  }
  SEND_WORD mem[0]
  SEND_MESSAGE
}
```
Suggested strategy to avoid getting stuck: Hysteresis

- Hysteresis might bridge the Zone of Death
Automaton Simulation

- Evaluation \( \equiv \) simulate program code and compare results with expectations
- Instantiate programs in automata \( (\equiv \) virtual hardware\)
- Simulate hardware-ticks, perform introspections periodically

Program

\[
\begin{align*}
@0: & \quad \text{SendWord mem[0]} \\
& \quad \text{SendMessage} \\
@1: & \quad \text{mem[1]=ReceiveWord} \\
& \quad \text{IfJump mem[1]<=mem[0], @1} \\
@2: & \quad \text{mem[0]=mem[1]} \\
& \quad \text{Goto @0}
\end{align*}
\]

Data Memory

<table>
<thead>
<tr>
<th>IP</th>
<th>SegP</th>
<th>Ticks</th>
<th>Slept Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td></td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>-234</td>
<td></td>
</tr>
<tr>
<td>5532</td>
<td></td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

Execution Status

- Automaton stepwise execution
Automaton Simulation

- Program-code evolved is Turing-complete
- Allows all possible algorithms to be evolved (in theory...)
- Reproduction uses optimization-cycle
- Reproduction caches results to reduce the amount of similar programs evolved
- Reproduction uses knowledge about instructions and operator semantics
- Code is exchanged in a direct executable format
Automaton Simulation

@0:
mem[0]=mem[0]
mem[1]=(mem[1] mod mem[0])
mem[0]=mem[0]-(mem[1] mod mem[0])
IfJump mem[1], @0
Sleep (mem[0]/mem[0])
IfJump mem[2], @0
mem[2]=(mem[0]/(0/mem[0]))
Goto @0

@0:
mem[0]=mem[0]
mem[1]=(mem[1] mod mem[0])
mem[0]=mem[0]-(mem[1])
Goto @0

@0:
mem[1]=811
Goto @0
Automaton Simulation

![Graph showing individuals processed over time for different servers and overall performance.](attachment:image.png)
Automaton Simulation

- Normal GA and simple GP: continuous fitness functions
- Software Engineering: A program’s functionality might change dramatically if only one piece of code is modified slightly.
- Our GP: fitness function is not continuous, contains jump discontinuity
- Evolution is very complicated, often might not work at all
Network Simulation Layer

- One Program used by multiple instantiated automata
- Network extension for virtual hardware
- State complemented by global network state
- Reusable Network simulator

Network Status
- MsgCount
- Collisions
- Lost
- Time

Network of Automata

Execution Status
- IP
- SegP
- Ticks
- Slept
- Net Extension
  - Lost
  - Received
  - Sent

Data Memory
- 231
- 0
- 39343
- 234
- 120
Network Simulation Layer

- The automata run at approximately the same speed (which cannot be regarded as constant)
- The system of automata runs asynchronously
- The automata will be started at different times
- The links between the nodes are randomly created (but no network partitions exist)
- Messages are simple sequences of memory words
- Messages are sent like radio broadcasts and will be received by every node in a specific distance
- Messages can get lost without special cause
- Transmissions may take a random time
- Collisions lead to message losses
Network Simulation Layer

@0:
    SendWord mem[0]
    SendMessage
@1:
    mem[1]=ReceiveWord
    IfJump mem[1]<=mem[0], @1
@2:
    mem[0]=mem[1]
    Goto @0
Fitness Functions

- Multiple fitness functions per evolution process are possible (multi-objective GP), but currently not used.

Fitness function

1. Create a random network.
2. Simulate the generated code on that network.
3. Compute the results the specified behavior would yield.
4. Compare both results.
5. Repeat this $n$ times and calculate the average/minimum fitness.
Examples of tasks open for students

- Create a GUI
  - Control the Genetic Engine
  - Gather and Visualize the Genetic Engine’s statistical data
  - compose/execute evolutions

- Develop and test strategies to speed up evolution or to improve its qualities (see Slide 9). Such strategies will be application-dependent and not work for all GA the same.
Examples of tasks open for students

- Help me to improve the GA or even the GP layer
  - By analyzing as many existing GP implementations as possible and compare their approach to ours
  - Analyze my source and suggest improvements
  - Find out how good the client/server approach scales

- Help me to improve network simulation layer
  - By analyzing as many existing wireless-LAN/transmission simulators implementations as possible and compare their approach to ours
  - Analyze my source and suggest improvements
Things that I expect

- Team work, Team meetings on a regular basis (every two weeks)
- Students need SourceForge Accounts
- All sources are documented fully (in English language)
- Everything you find/do/develop during your work should become part of the project and thus be LGPL licensed and also put on our website (but you’re the author, of course)
- All documents/results worked: in project’s CVS