Hybrid models as cooperating computational agents

Roman Neruda

Institute of Computer Science,
Academy of Sciences of the Czech Republic,
Prague, Czech Republic

roman@cs.cas.cz

UTIA, 31.10.2006
Computational intelligence

- Soft computing (L. Zadeh): creative fusion of artificial neural networks, evolutionary algorithms, fuzzy logic controllers, ...
- Benefits over individual methods.
- No one underlying theory.
- Importance of heuristics, experiments, practical skills.
- Combination with 'hard' (statistics, numerical analysis) and formal methods.
Goals

- To describe various computational methods as sets of several cooperating agents.
- **MAS scheme** is a concept for describing the relations within such a set of agents.
- It should be easy to ‘connect’ a particular computational method (implemented as an agent) into hybrid methods, using schemes description.
- The scheme description should be strong enough to describe all the necessary relations within a set of agents that need to communicate one with another in a general manner.
Bang as a middleware

- **support for agents life-cycle**: creation, migration, persistence,
- **communication**: message encoding, delivery
- **resource allocation**: memory, processor, disk
- **complexity analysis**: parallelization profiling
- **airport** on each computer, TCP/IP
- **agent granularity**: monolithic system / 1 or more threads per agent / processes
- **user interface**
Agents in Bang

- **computational agents**: neural nets (MLP, RBF), GA suite, Kohonen maps, vector quantization, decision tree
- **computational helpers**: linear system solver, gradient descent optimization
- **task-related**: data source, task manager, file system wrapper
- **system**: launcher, yellow pages, ontology services, debugger, profiler
- **other**: MASman, console, GUI
RBF as MAS

- GA
- VQ + GRAD + LS
- Training set
- RBF
- VQ
- GRAD
- Least Squares Solver
GA as MAS

Chromosom dependent

Chromosom independent

Required blocks

Optional blocks

Fitness

Operators

Genetics

Selection

Tuner

Shaper
GAs in action

Hybrid models as cooperating computational agents
Definitions: Communication

- **Message type:** identifies a category of messages that can be send to an agent in order to fulfill a specific task.

- **Interface:** the set of message types understood by a class of agents.

- **Gate:** a tuple consisting of a message type and a named link.

- **Connection:** a triple consisting of a sending agent, the sending agent’s gate, and a receiving agent.
Definitions: Agents and MAS

- **Agent class**: defined by an interface, a set of message types, a set of gates, and a set of types.
- **Agent**: an instance of an agent class. It is defined by its name and its class.
- **Multi-Agent Systems (MAS)**: consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.
## Concepts and roles

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mas(C)</td>
<td>C is a Multi-Agent System</td>
</tr>
<tr>
<td>class(C)</td>
<td>C is the name of an agent class</td>
</tr>
<tr>
<td>gate(C)</td>
<td>C is a gate</td>
</tr>
<tr>
<td>m_type(C)</td>
<td>C is a message type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type(X,Y)</td>
<td>Class X is of type Y</td>
</tr>
<tr>
<td>has_gate(X,Y)</td>
<td>Class X has gate Y</td>
</tr>
<tr>
<td>gate_type(X,Y)</td>
<td>Gate X accepts messages of type Y</td>
</tr>
<tr>
<td>interface(X,Y)</td>
<td>Class X understands mess. of type Y</td>
</tr>
<tr>
<td>instance(X,Y)</td>
<td>Agent X is an instance of class Y</td>
</tr>
<tr>
<td>has_agent(X,Y)</td>
<td>Agent Y is part of MAS X</td>
</tr>
</tbody>
</table>
Computational agent

class(decision_tree)
type(decision_tree, computational_agent)
has_gate(decision_tree, data_in)
gate_type(data_in, training_data)
interface(decision_tree, control_messages)
...
Computational MAS

\[
\text{comp\_MAS(MAS)} \leftarrow \\
\text{type(CAC, computational\_agent)} \land \\
\text{instance(CA, CAC)} \land \\
\text{has\_agent(MAS, CA)} \land \\
\text{type(DSC, data\_source)} \land \\
\text{instance(DS, DSC)} \land \\
\text{has\_agent(MAS, DS)} \land \\
\text{connection(CA, DS, G)} \land \\
\text{type(TMC, task\_manager)} \land \\
\text{instance(TMC, TM)} \land \\
\text{has\_agent(MAS, TM)} \land \\
\text{connection(TM, CA, GC)} \land \\
\text{connection(TM, GC, GD)}
\]
Trusted MAS

trusted_MAS(MAS) ←
    findall(X, has_agent(MAS,X, A)) ∧
    all_trusted(A)
all_trusted([]) ← true
all_trusted([F|R]) ←
    instance(F,FC) ∧
    type(FC, trusted) ∧
    all_trusted([R])

MAS is trusted if all of its agents are instances of a “trusted” class. Prolog predicates `findall` (returns a list of all instances of a variable for which a predicate is true) and `all_trusted`.
Description of Agents

(implies iAgentStdIface (and (some messagetype agentLifeManagement) (all messagetype agentLifeManagement)))

(implies igToYellowPages (and (some messagetype yellowPageRequest) (all messagetype yellowPageRequest)))

(implies Father (and (some interface iAgentStdIface) (all interface iAgentStdIface) (some gate igToYellowPages) (all gate igToYellowPages)))

...
Description of Agents

;;;;Decision Tree
(implies aDecisionTree (and Classifier
IterativeComputation
Father
classInBang))

;;;;Neural Networks
(implies NeuralNetwork Approximator)

;;;;RBF Network
(implies RBFNetworkAI (and NeuralNetwork
IterativeComputation
classInBang
SimpleTaskManager
Father
(some gate igSolveRepresentatives)
(some hide igCommonCompControl)
(all hide igCommonCompControl)
(some gate igSolveLinEqSystem)

(all gate (or igSolveRepresentatives igSolveLinEqSystem)))
BOA agent generates a MAS configuration description and...

Roman Neruda

Hybrid models as cooperating computational agents
Introduction

Description of Computational Agents

Evolutionary algorithm

Autonomous Behaviour Support

Conclusions

Outline

1. Introduction
2. Description of Computational Agents
   - Description of Agent and MAS
   - Implementation
3. Evolutionary algorithm
   - Description
   - Experiments
4. Autonomous Behaviour Support
   - Architecture description
   - Experiments
5. Conclusions
What it does

- EA operates on schemes definitions in order to find a suitable scheme solving a specified problem.

- Inputs:
  - number and the types of inputs and outputs of the scheme;
  - training set used to compute the fitness of a particular solution;
  - list of types of agents available for being used in the scheme.

- EA uses the agents logical description and reasoning component (described above) in order to produce only such schemes that satisfy given constrains.
How it works

- Random creation of population feasible schemes
- Evaluation of scheme:
  - Creating the MAS
  - Running it on the training set
  - Computing the fitness
- Creating new population by means of:
  - roulette-wheel selection
  - crossover of 2 schemes
  - 2 mutations of schemes (link swap, random node change)
Crossover and Mutation
Symbolic regression

- Setup similar to J. Koza Genetic programming task
- Training set: 100 samples of a polynomial $x^3 - 2x^2 - 3$
- Agents: two families of agents working on INT and FLOAT values
- MUL, ADD, COPY, ROUND, FLOATIZE, ...
- 1 generation evaluation takes seconds on a 2GHz machine
- Several hundred/thousands generations needed to find a solution
Convergence

Roman Neruda

Hybrid models as cooperating computational agents
The best scheme from generation 3000
Outline

1. Introduction

2. Description of Computational Agents
   - Description of Agent and MAS
   - Implementation

3. Evolutionary algorithm
   - Description
   - Experiments

4. Autonomous Behaviour Support
   - Architecture description
   - Experiments

5. Conclusions
Adaptive Computational Agent

In order to act autonomously, an agent should be able to cope with three different kinds of problems:

- cooperation of agents,
- computation processing support,
- optimization of the partner choice.

The architecture supports

- reasoning,
- descriptions of agents and tasks (ontologies),
- monitoring and evaluation of various parameters,
- learning.
Cooperation of agents

An intelligent agent should be able to answer the questions about its willingness to participate with particular agent or on a particular task. The following subproblems follow:

- deciding whether two agents are able to cooperate,
- evaluating the agents (according to reliability, speed, availability, etc.),
- reasoning about its own state of affairs (state of an agent, load, etc.),
- reasoning about tasks (identification of a task, distinguishing task types, etc.).
Computations processing

The agent should be able to recognize what it can solve and whether it is good at it, to decide whether it should persist in the started task, and whether it should wait for the result of task assigned to another agent. This implies the following new subproblems:

- learning (remembering) tasks the agent has computed in the past (we use the principles of case-based learning and reasoning to remember task cases),
- monitoring and evaluation of task parameters (duration, progress, count, etc.),
- evaluating tasks according to different criteria (duration, error, etc.).
Optimization of the partner choice

An intelligent agent should be able to distinguish good partners from unsuitable ones. The resulting subproblems follow:

- recognizing a suitable (admissible) partner for a particular task,
- increasing the quality of an evaluation with growing experience.
Layer architecture
Modeling State of an Agent.

- **Computation State**
- **Tiredness**
- **Load**
- **Stress**

**Monitors**
- **Tasks monitors**
  - Computation Completed %
  - Idle Time
  - Tasks Count
  - Tasks Duration
  - Computation Time
  - Rejected Requests #

**Experiments**

Hybrid models as cooperating computational agents
Measuring Quality of Services of Partners.

![Diagram of Service Quality and Monitors](image-url)

- Satisfaction
- Services Quality
- Accuracy
- Reliability
- Speed
- Availability

Monitors:

- Task monitors:
  - Tasks Error
  - Interrupted Tasks #
  - Tasks Duration
  - Accepted Cooperation #
  - Reply Time
  - Ping Time

- Hybrid models as cooperating computational agents
Support for Cooperation.

Hybrid models as cooperating computational agents

Roman Neruda
Computations Processing Support.

- Recent Computations
- Can Compute Task
- Computation Persist
- Wait For Result

Task monitors:
- Tasks repository
- Computation Completed %
- Computation Time
- Assigned Computation Time

Preferences:
- Reconsidering Approach

Stress
Tiredness

Hybrid models as cooperating computational agents
Optimization of Partner Choice.

- Best Accuracy
- Best Speed
- Best Services
- Best Availability
- Best Reliability

Evaluator of the agents’ ranking in:
- Services
- Quality
- Accuracy
- Speed
- Availability
- Reliability

Roman Neruda
Hybrid models as cooperating computational agents
BDI architecture within the Network of Concepts.

- Cooperation
  - Task Solving Decision
- Computations
  - Computation Persist
  - Recent Computations
  - Wait For Result
  - Computation State
- Preferences
  - Proactivity Regime
- Optimization
  - Best Services
- Task monitors
  - Tasks repository
- Options
  - BDI Simple Task Manager Options
  - BDI Task Manager Options
- BDI model

Roman Neruda
Hybrid models as cooperating computational agents
Experiments summary

- Testing on perceptron and local-unit neural networks, with two task managers, sets of (repeating) tasks.
- The architecture has (only) about 10% overhead.
- Optimization enhances efficiency of computational agents (w.r.t. different criteria).
## Overhead of the architecture

<table>
<thead>
<tr>
<th></th>
<th>Without the arch.</th>
<th>With the arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent creation time</td>
<td>3604 $\mu$s</td>
<td>9890 $\mu$s</td>
</tr>
<tr>
<td>Message delivery time</td>
<td>2056 $\mu$s</td>
<td>2672 $\mu$s</td>
</tr>
<tr>
<td>Total computation time</td>
<td>8994681 $\mu$s</td>
<td>9820032 $\mu$s</td>
</tr>
</tbody>
</table>
## Optimization of partner choice

<table>
<thead>
<tr>
<th></th>
<th>Error</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random choice</td>
<td>11.70</td>
<td>208710ms</td>
</tr>
<tr>
<td>Best speed</td>
<td>1.35</td>
<td>123259ms</td>
</tr>
<tr>
<td>Best Accuracy</td>
<td>1.08</td>
<td>274482ms</td>
</tr>
<tr>
<td>Best services</td>
<td>1.17</td>
<td>102247ms</td>
</tr>
</tbody>
</table>
Optimization by Reusing

<table>
<thead>
<tr>
<th>Repeated tasks</th>
<th>Optimized</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 %</td>
<td>135777097</td>
<td>121712748</td>
</tr>
<tr>
<td>20%</td>
<td>94151838</td>
<td>90964553</td>
</tr>
<tr>
<td>40%</td>
<td>50704363</td>
<td>91406591</td>
</tr>
<tr>
<td>60%</td>
<td>47682940</td>
<td>90804052</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Description of Computational Agents
   - Description of Agent and MAS
   - Implementation
3. Evolutionary algorithm
   - Description
   - Experiments
4. Autonomous Behaviour Support
   - Architecture description
   - Experiments
5. Conclusions
Conclusions

- Formal Description of computational agents and MAS ...
- ... by means of Description Logics and Prolog-like rules
- Application to automatic MAS scheme generation for simple problems
- Integration with Evolutionary algorithms on schemes
- Autonomous behavior support for computational agents
To do

- Dynamical aspects:
  - tasks description,
  - agents performance.
- User assistance, model verification.
- Use of First-Order Logics reasoning engine (KR-HYPER).
- Hybrid approaches, better combination with evolutionary search.
- Using the information from autonomous support in scheme evolution.