TTC 2018 CASE PRESENTATION

Quality-based Software-Selection and Hardware-Mapping as a Model Transformation Problem

Sebastian Götz, Johannes Mey, Rene Schöne and Uwe Aßmann
The TTC Case

Optimally combine heterogeneous hardware and adaptive software by deriving a solution model from a problem model.
Our History of the Case

In the beginning, there was a PhD in 2013:

- [Götz 2013] *Multi-Quality Auto-Tuning by Contract Negotiation*
Our History of the Case

In the beginning, there was a PhD in 2013:

- [Götz 2013] *Multi-Quality Auto-Tuning by Contract Negotiation*

which was improved by faster intermediate model generation in 2016:

- [Schöne et al. 2016] *Incremental Runtime-Generation of Optimisation Problems Using RAG-Controlled Rewriting*
In the beginning, there was a PhD in 2013:

- [Götz 2013] *Multi-Quality Auto-Tuning by Contract Negotiation*

which was improved by faster intermediate model generation in 2016:

- [Schöne et al. 2016] *Incremental Runtime-Generation of Optimisation Problems Using RAG-Controlled Rewriting*

which was still a bit slow, so now there is

- TTC 2018
Problem 1: “Software Selection”

Software model:
- Software component specifications:
  - functionality
- Implementations of component specs:
  - provide non-functional properties
  - require other components

Selection Task
- Fulfill requests
  - chose implementations
  - ensure non-functional requirements
- Solution Part 1: Trees of assignments
Problem 2: “Hardware Mapping”

- **Hardware model**
  - Resources with sub-resources and properties

- **Contracts**
  - Implementations specify resource requirements

**Resource Allocation Task**

- Map assignments to hardware
  - ensure resource requirements

**Solution Part 2**: Resource mapping
Problem 3: “Quality-Based”

- **Contracts**
  - Implementations provide non-functional properties depending on hardware

**Optimization task**

- Optimize aggregated non-functional property of system
  - *Here*: minimize energy
- **Solution Part 3**: Assignments + mapping with minimal energy
The Models in Detail

• Model: two grammars with overlay edges and connecting references
  - Problem model:
    • software and hardware part
  - Solution model:
    • tree of dependent assignments

• Grammar?
  - Reference Attribute Grammar: efficient analysis
  - Parser available
  - Simple solution within model
The Models in Detail

Components

Request

Target

Software-Component A

Impl A-0

Requires

Software-Component B

Impl B-0

Impl A-1

Requires

Software-Component C

Impl C-0

HardwareResource 1

HardwareResource 2

HardwareResource 4

HardwareResource 5
The Models in Detail

Solution Part 1: Implementation Selection

Request

Software-Component A

Impl A-0

Impl A-1

Software-Component B

Impl B-0

Software-Component C

Impl C-0

HardwareResource 1

HardwareResource 2

HardwareResource 4

HardwareResource 5
Solution Part 2: Hardware Mapping

- Request
- Software-Component A
- Impl A-0
- Impl A-1
- Software-Component B
- Impl B-0
- Software-Component C
- Impl C-0

HardwareResource 1
HardwareResource 2
HardwareResource 4
HardwareResource 5

(target)
Solution Part 3: Optimization

- Request
  - target
  - Software-Component A
    - Impl A-0
      - requires
      - Software-Component B
        - Impl B-0
          - requires
    - Impl A-1
      - requires
      - Software-Component C
        - Impl C-0
          - requires

Valid: ✓
Optimal: ✗
Valid: ✔
Optimal: 😐
Task and Solutions
Case Scenarios

• Five **sizes:**
  - minimal, small, medium, large, huge

• Three **types:**
  - standard
  - more hardware components
  - more (complex) software components

• Flexible scenario **generator:**
  - 10 parameters for software/hardware config
  - Fixed hardware types, and software properties
  - Flexible shape of software model and solution tree
## Case Scenarios

<table>
<thead>
<tr>
<th>ID</th>
<th>Requests</th>
<th>Impl’s</th>
<th>Resources</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>minimal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>small</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>15</td>
<td>small-hw</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>62</td>
<td>47</td>
<td>small-sw</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>30</td>
<td>68</td>
<td>medium</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>30</td>
<td>225</td>
<td>medium-hw</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>155</td>
<td>465</td>
<td>medium-sw</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>60</td>
<td>90</td>
<td>large</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>60</td>
<td>300</td>
<td>large-hw</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>310</td>
<td>930</td>
<td>large-sw</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>150</td>
<td>225</td>
<td>huge</td>
</tr>
<tr>
<td>11</td>
<td>50</td>
<td>150</td>
<td>750</td>
<td>huge-hw</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>620</td>
<td>2325</td>
<td>huge-sw</td>
</tr>
</tbody>
</table>
A Simple Attribute Grammar Reference Solution

- Simple reference implementation
  - Based on reference attribute grammar
  - Iterator over model
  - Some pruning

- Performance:
  - Almost full state space exploration
  - Encouraging for TTC participants
  - Always finds optimal solution... eventually
## Evaluation criteria

<table>
<thead>
<tr>
<th><strong>Solution time</strong></th>
<th>Time to compute a valid solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solution quality:</strong></td>
<td>Validity of solution + Quality of found objective value</td>
</tr>
<tr>
<td><strong>Scalability:</strong></td>
<td>Largest scenario for which a valid solution can be found</td>
</tr>
</tbody>
</table>
Measurement results

- ✔️ = valid and in time
- ✗ = valid, but timeout
- ✗ = invalid
- 🔥 = optimal (if known from ILP solver)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>ACO</th>
<th>EMFeR</th>
<th>ILP (direct/ext)</th>
<th>Simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 trivial</td>
<td>6</td>
<td>194</td>
<td>24 / 21</td>
<td>1</td>
</tr>
<tr>
<td>1 small</td>
<td>8</td>
<td>212</td>
<td>37 / 40</td>
<td>6</td>
</tr>
<tr>
<td>2 small-hw</td>
<td>11</td>
<td>240</td>
<td>44 / 61</td>
<td>8</td>
</tr>
<tr>
<td>3 small-sw</td>
<td>451</td>
<td>7min52s</td>
<td>377 / 572</td>
<td>15min</td>
</tr>
<tr>
<td>4 medium</td>
<td>1min33</td>
<td>8min22s</td>
<td>8min28s</td>
<td>15min</td>
</tr>
<tr>
<td>5 medium-hw</td>
<td>4min48s</td>
<td>11min15s</td>
<td>twig</td>
<td>15min</td>
</tr>
<tr>
<td>6 medium-sw</td>
<td>15min</td>
<td>x</td>
<td>15min</td>
<td>15min</td>
</tr>
</tbody>
</table>
Some Observations

- ACO sometimes returns invalid solutions
- ILP direct much better than ILP external
- EMFeR for scenarios 3-6 aborts search before timeout
- Simple either is fastest and optimal, or runs into timeout
References


Backup
Questions to the Audience

- Accessibility of the benchmark?
- Explanation of the case clear enough?
- How complex was the problem (compared to previous years)?
- Anything missing or improvable in the benchmark framework?
Grammar Hardware

![Diagram of Grammar Hardware](image)
Grammar Expression

Exp

0..1
General::Instance

1
General::Property

1
SW::MetaParameter

SoftwareDesignator

LiteralExpression
value: double

Expression

BinaryExpression
left
right

PropertyResourceDesignator

MetaParameterDesignator

AddExpression
SubExpression
MultExpression
DivExpression
PowExpression
Grammar General

```
Objective
  agg: PropertyAggregation

Property
  unit: String

Model
  HW::HardwareModel

Request
  name: String

MetaParameter
  SW::SoftwareModel

Assignment
  SW::Component

SW::Clause

enum
  PropertyAggregation
  SUM, MAX

enum
  ClauseComparator
  LT, LE, EQ, NE, GE, GT

enum
  ClauseType
  REQUIRING, PROVIDING

Solution
General::Model
```
Grammar Solution

```
Solution

  General::Model

  SW::Implementation

  Solution

  Assignment

    topLevel: boolean

  ComponentMapping

  Instance

  ResourceMapping

  HW::Resource

  General::Request
```

TTC 2018 Case Presentation

Slide 29 of 30