Hyper-Heuristics

Andreas Neustifter, e0325716@student.tuwien.ac.at

Seminar aus Algorithmik, LVA 186.182, SS 2008
Outline

Motivation

Hyper-Heuristics

Using Hyper-Heuristics for 2D Bin Packing

Conclusions
What are Hyper-Heuristics?

“...using (meta-)heuristics to choose (meta-)heuristics...” [2]
Problems with “Traditional” Heuristics

- simple but inefficient OR
- complex and hard to parametrise
- very problem specific
- require deep problem knowledge
- may not work on only slightly different problems
Hyper-Heuristics Basics

Hyper-Heuristics Prerequisites

- a set of easy, non parametrized low level heuristics
- that work on and return partially solved problems \(^1\)
- a measure for solution quality

Hyper-Heuristic chooses a sequence of heuristics that improves the solution most.

\(^1\)One does not need know what the problem is or what the heuristics do!
Naive Approach

If \( (\text{problemType} ( P ) == \text{p1} ) \) apply \( (\text{heuristic1}, P) \);

\text{ElseIf} ( \text{problemType} ( P ) == \text{p2} ) \) apply \( (\text{heuristic2}, P) \);

\text{ElseIf} ( \text{problemType} ( P ) == \text{p3} ) \) apply \( (\text{heuristic3}, P) \);

\text{ElseIf} ( \text{problemType} ( P ) == \text{p4} ) \) apply \( (\text{heuristic4}, P) \);

....

Well, yes, but

- usually it is not possible to determine problem type efficiently
- the best heuristic for each problem type must be known
- theoretically each problem could be of a different type
Outline

Motivation

Hyper-Heuristics

Using Hyper-Heuristics for 2D Bin Packing

Conclusions
Key Ingredients

▶ a set of low level heuristics that
  ▶ work on partly solved problems
  ▶ solve the problem a “little more”
▶ a measure of efficiency for either the heuristics OR
▶ a measure of solution quality for a partly solved problem
▶ a Hyper-Heuristic that
  ▶ selects the next heuristic to use
  ▶ and determines when to stop
Hyper-Heuristic: Algorithm

The Hyper-Heuristic iteratively

1. uses a heuristic to choose which heuristic to apply next
2. applies heuristic and records the results (processing time, solution improvement,...)
3. checks if the abortion criterion is meet (solution is good enough,...)
4. goes to step 1.
Domain Barrier

- is an important part in a Hyper-Heuristic
- the knowledge about the problem (about the domain) is only in
  - the low level heuristics
  - the solution quality function
- this enables the Hyper-Heuristic to solve different problems by switching only the low level heuristics
Domain Barrier: Interface

A proper interface between hyper and low level heuristics

- ensures the domain barrier
- eases addition of low level heuristics
- makes no assumptions about used heuristics

It usually contains

- the problem instance given to and returned from the heuristic
- solution improvement measure calculated by low level heuristic
Hyper-Heuristics Main Features

- uses a set of easy (implementation, parameters, ...) heuristics
- work on other problem by only switching the set of heuristics
- fine tuning of Hyper-Heuristic is done only once and is not so time consuming
- good average performance [7]
- often faster to implement and fine tune
Outline

Motivation

Hyper-Heuristics

Using Hyper-Heuristics for 2D Bin Packing

Conclusions
Problem Description

Place a set of rectangular items into rectangular bins such that

- the items do not overlap
- the items are rotated only by 0 or 90 degrees
- the number of used bins is minimised
Solution Approach [4]

Use Ant Colony Optimisation (ACO) to select low level heuristics. ACO consists of:

- Pheromones (for guiding ants through solution space)
- Pheromone Update (to encode good solutions in the pheromones)
- Pheromone Evaporation (to make bad solutions less probable)
Used Low Level Heuristics

Low Level Heuristics are a combination of five values:

- Quantity (how many items to place)
- Rotation (is rotation allowed)
- Item Order (in which order are items selected)
- Bin Selection (into which bin does the item go)
- Item Placement (where is the item placed in the bin)

The squared bin utilization was used as solution quality measure:

\[
Q_R = \frac{1}{N} \sum_{B=1}^{N} \left( \frac{1}{A_B} \sum_{i=1}^{n} A_{I_i} \right)^2
\]
Using 5 variables per heuristic and executing (at most) 5 heuristics, gives 25 pheromone matrices plus one start matrix:

**Figure:** ACO Matrices
Coding to ACO: Pheromone Matrices

Each matrix $Ph_V$ has

- number of rows $\leftrightarrow$ number of possible values for current variable
- number of columns $\leftrightarrow$ number of possible values for next variable

A entry in row $i$ and column $j$ encodes:

Provided value $i$ of current variable is selected, select value $j$ of next variable with probability $Ph_V(i,j)$. 
Coding to ACO: Update and Evaporation

After an iteration the path quality is assessed and normalised and squared for each path:

$$Q(path) = \left( \frac{Q_R(path) - \min(Q_R)}{\max(Q_R)} \right)^2$$

The $Q(path)$ is added in the cells of the pheromone matrices where the path passed.

The evaporation is done by normalizing the sum of each matrix row to 1.
Coding of ACO: Termination

- use a total of 30000 ants
- 50 ants form an iteration
- when two successive iterations did not improve quality a cycle is finished
- the final solution is that of the best cycle
### Comparison to other 2D Bin Packing Systems

<table>
<thead>
<tr>
<th>criterion</th>
<th>package quality</th>
<th>used bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>best single heuristic per instance</td>
<td>100 %</td>
<td>122 %</td>
</tr>
<tr>
<td>ACO Hyper-Heuristic</td>
<td>104,23 %</td>
<td>119 %</td>
</tr>
<tr>
<td>Genetic Hyper-Heuristic [1]</td>
<td>-</td>
<td>118 %</td>
</tr>
<tr>
<td>Classifier Based Approach [6]</td>
<td>-</td>
<td>118 %</td>
</tr>
<tr>
<td>theoretical</td>
<td>-</td>
<td>100 %</td>
</tr>
</tbody>
</table>

(Using cgcut, gcut, ngcut [3, 5] as problem instances.)
Outline

Motivation

Hyper-Heuristics

Using Hyper-Heuristics for 2D Bin Packing

Conclusions
Hyper-Heuristics Are

- easier to implement (than traditional high performance heuristics)
- more stable (better average performance)
- reusable
- a viable alternative to single heuristic approaches
Appendix

Morán-Saavedra A.
Optimización de corte de material en dos dimensiones mediante hiperheurísticas construidas con un algoritmo genético, 2004.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg.

Nicos Christofides and Charles Whitlock.
An algorithm for two-dimensional cutting problems.

Alberto Cuesta-Cañada, Leonardo Garrido, and Hugo Terashima-Marín.

Beasley J. E.
An exact two-dimensional non-guillotine cutting tree search procedure.
Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems.  
pages 637–643, Washington DC, USA, 2005. ACM.

No free lunch theorems for optimization.  