Automated Heuristic Design

Gabriela Ochoa, Matthew Hyde & Edmund Burke
Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, The University of Nottingham
{gxo, mvh}@cs.nott.ac.uk

Copyright is held by the author/owner(s).
GECCO’11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.
Agenda

❖ First Section: Introduction
• General introduction and motivation
• What is a hyper-heuristic?
• Classification of hyper-heuristic approaches

❖ Second Section: Heuristic Selection Methodologies
• A Constructive hyper-heuristic: Graph-based hyper-heuristic
• A Perturbative hyper-heuristic: Tabu-search hyper-heuristic
• HyFlex and the Cross-domain Heuristic Search Challenge
• Conclusion and Future Work

❖ Third Section: Heuristic Generation Methodologies
• Introduction
  – Hyper-heuristic Definition
  – What’s the Point?
• Case Study 1: Max-SAT
• Case Study 2: Bin Packing
• Conclusion
Automated Heuristic Design

- Search and optimisation problems are everywhere, and search algorithms are getting increasingly powerful
- They are also getting increasingly complex
- Only autonomous self-managed systems that provide high-level abstractions can turn search algorithms into widely used methodologies

**Research Goals:**
- Reduce the role of the human expert in the process of designing optimisation algorithms and search heuristics
- Software systems able to automatically tune, configure, generate and design optimisation algorithms and search heuristics.
- Self-tuning, self-configuring and self-generating search heuristics
Automated Heuristic Design: Several Approaches

- **Online approaches**
  - Self-tuning and self-adapting heuristics on the fly, effectively learning by doing until a solution is found
  - **Examples:** adaptive memetic algorithms, adaptive operator selection, parameter control in evolutionary algorithms, adaptive and self-adaptive search algorithms, reactive search

- **Offline approaches**
  - Learn, from a set of training instances, a method that would generalise to unseen instances
  - **Examples:** automated algorithm configuration, meta-learning, performance prediction, experimental methods, SPO

- **Hyper-heuristics (offline and online)**
Motivation

The “Up the Wall” game

- We have a problem (e.g. exam timetabling) and a set of benchmark instances
- We develop new methodologies (ever more sophisticated)
- Apply methodologies to benchmarks
- Compare with other “players”
- The goal is to “get further up the wall” than the other players

Consequence: Made to measure (handcrafted) Rolls-Royce systems

e.g. Exam Timetabling
Motivation

The “Many Walls” game

- Can we develop the ability to automatically work well on different problems?
- Raising the level of generality
- Still want to get as high up the wall as possible ... BUT...
- We want to be able to operate on as many different walls as possible

- Consequence: Off the peg, *Ford* model

One method that operates on several problems
Motivation

- Develop decision support systems that are \textit{off the peg}
- Develop the ability to automatically work well on different problems

Research challenges

- Automate heuristic design
  - Now made by human experts
  - Not cheap!
- How general we could make hyper-heuristics
  - No free lunch theorem
Motivation

The General Solver

Doesn’t exist....

Significant scope for future research

More General

These situations exist

Problem Specific Solvers
What is a Hyper-heuristic?

'standard' search heuristic

Operates upon

potential Solutions
Hyper-heuristics:
“Operate on a search space of heuristics”

- 'standard' search heuristic
  - Operates upon
  - potential Solutions

- hyper-heuristic
  - Operates upon
  - heuristics
  - Operates upon
  - potential Solutions
What is a hyper-heuristic?

Recent research trend in hyper-heuristics

- Automatically *generate* new heuristics suited to a given problem or class of problems
- Combining, i.e. by GP, *components* or *building-blocks* of human designed heuristics

New definition:

A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems

Origins and early approaches

Term *hyper-heuristics*
- First used 1997 (Dezinger et. al): a protocol for combining several AI methods in automated theorem proving
- Independently used in 2000 (Colwing et. al): ‘heuristic to choose heuristics’ in combinatorial optimisation
- First journal paper (Burke et. al, 2003)

The ideas can be traced back to the 60s and 70s
- Automated heuristic sequencing (early 60s and 90s)
- Automated planning systems (90s)
- Automated parameter control in evolutionary algorithms (70s)
- Automated learning of heuristic methods (90s)
Classification of hyper-heuristics

Search paradigms

Perturbation
- **Search space:** complete candidate solutions
- **Search step:** modification of one or more solution components
- **TSP:** 2-opt exchanges

Construction
- **Search space:** partial candidate solutions
- **Search step:** extension with one or more solution components
- **TSP:** Next-neighbour
Classification of hyper-heuristics (nature of the search space)

Hyper-heuristics

Heuristic Selection
- Construction heuristics
- Perturbation heuristics

Heuristic generation
- Construction heuristics
- Perturbation heuristics

Fixed, human-designed low level heuristics

Heuristic components
Classification of hyper-heuristics (source of feedback during learning)

Online
- Learning while solving a single instance
- Adapt
- **Examples**: reinforcement learning, meta-heuristics

Offline
- Gather knowledge from a set of training instances
- Generalise
- **Examples**: classifier systems, case-based, GP
Section 2: Heuristic Selection Methodologies

A constructive Hyper-heuristic
Graph-based hyper-heuristics

- A general framework (GHH) employing a set of low level constructive graph colouring heuristics

- Low level heuristics: sequential methods that order events by the difficulties of assigning them
  - 5 graph colouring heuristics
  - Random ordering strategy

- Applied to exam and course timetabling problem

Examination timetabling

A number of exams \((e_1, e_2, e_3, \ldots)\), taken by different students \((s_1, s_2, s_3, \ldots)\), need to be scheduled to a limited time periods \((t_1, t_2, t_3, \ldots)\) and certain rooms \((r_1, r_2, r_3, \ldots)\).

**Hard Constraints**
- Exams taken by common students can’t be assigned to the same time period
- Room capacity can’t be exceeded

**Soft Constraints**
- Separation between exams
- Large exams scheduled early
How can we model this problem?

- There are 7 exams, e1 ~ e7
- 5 students taking different exams
  - s1: e1, e2, e4
  - s2: e2, e3, e4
  - s3: e3, e4, e5
  - s4: e4, e5, e6
  - s5: e7

Objective: assign colours (time periods) to nodes (exams), adjacent nodes with different colour, minimising time periods used
Low-level heuristics

Order events by how difficult to schedule them

<table>
<thead>
<tr>
<th>Graph Heuristics</th>
<th>Ordering strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest degree (LD)</td>
<td>Number of clashed events</td>
</tr>
<tr>
<td>Largest weighted degree (LW)</td>
<td>LD with number of common students</td>
</tr>
<tr>
<td>Saturation degree (SD)</td>
<td>Number of valid remaining time periods</td>
</tr>
<tr>
<td>Largest enrolment (LE)</td>
<td>Number of students</td>
</tr>
<tr>
<td>Colour degree (CD)</td>
<td>Number of clashed event that are scheduled</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Random ordering (RO)</td>
<td>Randomly</td>
</tr>
</tbody>
</table>

Automated Heuristic Design
Graph-based hyper-heuristics

Automated Heuristic Design
Graph-based hyper-heuristics

Automated Heuristic Design
Graph-based hyper-heuristics

events

| e2 | e4 | e5 | e7 | e8 | e11 | e12 | ...

heuristic list

SD  SD  LD  CD  LE  SD  SD  LW  SD  LD  CD  RO  ...

order of events

| e5 | e32 | e19 | e22 | e13 | e31 | e12 | e7 | e2 | e15 | e27 | e12 | ...

slots

| e1 | e3 | e6 | e19 | e26 | e25 | e28 | e17 | e10 | e5 | e13 | e32 | e19 | e13 |
**Graph-based hyper-heuristics**

- Tabu Search and other meta-heuristics (VNS, ILS) used to search the heuristic search space
- **Objective function**: quality of solutions (timetables) built by the corresponding heuristic list
Heuristic Selection Methodologies

- The domain barrier
- A perturbative hyper-heuristic: Tabu-search hyper-heuristic
- HyFlex and the Cross-domain Heuristic Search Challenge
Decide which heuristic, $i$, to apply to which solution, $j$, and where to store it in the list of solutions, $k$. Based only on past history of heuristics applied and objective function values returned.

**Problem Domain**

- Problem representation
- Problem instances
- Evaluation function $f(s_k)$
- **List of solutions**
- Others…

**Hyper-heuristic**

- $H_1$, $H_n$
- $H_2$, …

**Domain Barrier**

$f(s_k)$

**HH framework:** (Cowling P., Kendall G. and Soubeiga, 2000, 2001), (E. K. Burke et al., 2003)


Automated Heuristic Design
Tabu-search hyper-heuristics

- Heuristics selected according to learned ranks (using reinforcement learning)
- Dynamic tabu list of heuristics that are temporarily excluded from the selection pool

Later combined with SA acceptance

Each heuristic $k$ is assigned a rank $r_k$ initialised to 0 and allowed to increase and decrease within interval $[r_{\text{min}}, r_{\text{max}}]$. 

**Tabu search hyper-heuristics**

- Select highest-ranking non-tabu heuristic and apply it once
  - Improvement in objective function $> 0$: Increase rank
  - Improvement in objective function $= 0$: Decrease rank
  - Improvement in objective function $< 0$: Decrease rank
  - Empty tabu list
    - Include highest-ranking heuristic in tabu list on a 1st in, 1st out basis
    - Check stopping condition
      - Does not hold: STOP & output best solution (s)
      - Holds: Continue

Automated Heuristic Design
**HyFlex (Hyper-heuristics Flexible framework)**

- **Question**: Can we produce a benchmark to test the generality of heuristic search algorithms?

- A software framework (problem library) for designing and evaluating general-purpose search algorithms

- Provides the *problem-specific* components

- Efforts focused on designing high-level strategies

HyFlex: a benchmark for cross-domain heuristic search

- Six different domains, hard combinatorial problems, interesting and varied set of operators and instances
- Implemented using the same software framework (common software interface)
- A single high-level strategy can operate and solve all the domains
- What are the principles and design strategies of successful cross-domain search heuristics?

http://www.asap.cs.nott.ac.uk/chesc2011/
Overview of the problem domain modules

1. A routine to initialise (randomised) solutions
2. A population or list of solutions
3. A set of heuristics to modify solutions
   a. Mutational: makes a random modification
   b. Ruin-recreate: partially destroy a solution and rebuild it using a constructive procedure
   c. Local-search (hill-climbing): iterative procedures searching on the neighbourhood of solutions for non-worsening solutions
   d. Crossover: takes parent solutions and produce offspring solution
4. A set of interesting instances, that can be easily loaded
“Civilization advances by extending the number of important operations which we can perform without thinking about them.”

Alfred North Whitehead, *Introduction to Mathematics (1911)*

**Crowdsourcing:** “the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call”.

*Jeff Howe, Wired Magazine, 2006*
Conclusions of 1st Section

A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems.

- **Main feature:** search in a space of heuristics
- **Term used for** ‘heuristics to choose heuristics’ in 2000
- **Ideas can be traced back to** the 60s and 70s
- **Two main type of approaches**
  - Heuristic selection
  - Heuristic generation
- **Ideas from online and offline machine learning are relevant,** as are ideas of meta-level search
Future work

- **Generalisation**: By far the biggest challenge is to develop methodologies that work well across several domains.

- **Foundational studies**: Thus far, little progress has been made to enhance our understanding of hyper-heuristic approaches.

- **Distributed, agent-based and cooperative approaches**: Since different low-level heuristics have different strengths and weakness, cooperation can allow synergies between them.

- **Multi-criteria, multi-objective and dynamic problems**: So far, hyper-heuristics have been mainly applied to single objective and static problems.
References: Hyper-heuristics

References: Automated Heuristic Design

This a small sample of books, survey papers, and other journal papers:

Section 3
Heuristic Generation Methodologies
Outline

- Introduction to this section
  - Hyper-Heuristic Definition
  - What’s the Point?
- Case Study 1: SAT
- Case Study 2: Bin Packing
- Conclusion
“A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems”
Two Types of Hyper-Heuristic?

A Hyper Heuristic Model:

Hyper Heuristic to Generate Heuristics

- Heuristics
  - Problem
  - Hyper-Heuristic
    - Heuristic Defined by the
      - User
    - Domain-Specific Heuristic Defined by the
      - Hyper-Heuristic

Heuristics

Heuristic

Heuristic

Heuristic

Heuristic

Heuristic

Problem
What’s the Point?

- We spend a lot of time testing, and fine tuning, solution methods.
- They are usually specialised to a particular problem instance set, with certain characteristics.
- Automating this creative process can potentially save time and/or effort.
- Humans still have a creative role in heuristic generation, but the idea is that more of the process is automated.
What’s the Point?
Heuristic Generation Methodologies

Case Study 1
Evolving Heuristics for SAT
Bader-el-Den and Poli, 2007
Based on Fukunaga, 2004, 2008
SAT local search heuristics can be evolved from a set of components, obtained by analysing existing heuristics from the literature
Evolving Heuristics for SAT

- Make a boolean expression true
- \((\neg A \lor B \lor C) \land (B \lor \neg C \lor E) \land (\neg B \lor A \lor \neg D) \land (\ldots) \land (\ldots) \ldots\)
- Hundreds/thousands of variables and clauses
- Local search heuristics iteratively choose a variable to flip.
Existing Heuristics for SAT

- **GSAT**
  - Flip variable which removes the most broken clauses (highest ‘net gain’)

- **HSAT**
  - Same as GSAT, but break ties by choosing the variable that has remained ‘unflipped’ for the longest

- **HARMONY**
  - Pick random broken clause BC. Select the variable V in BC with highest net gain, unless V has been flipped most recently in BC. If so, select V with probability p. Otherwise, flip variable with 2nd highest net gain
**Existing Heuristics for SAT**

- **GWSAT**
  - With probability 0.5, apply GSAT
  - Otherwise flip a random variable in a random broken clause.
They define a grammar, which can represent many heuristics from the literature, and new heuristics.
Evolving New SAT Heuristics

- **Flip**
- Maximum Net Gain
  - IF
    - 20% Broken Clause
    - All Clauses
  - Tie:
    - Age
    - All Clauses

<table>
<thead>
<tr>
<th>start</th>
<th>FLIP v</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>RANDOM l</td>
</tr>
<tr>
<td></td>
<td>MAX_SCR l</td>
</tr>
<tr>
<td></td>
<td>IFV prob, v, v</td>
</tr>
<tr>
<td></td>
<td>MIN_SCR l</td>
</tr>
<tr>
<td></td>
<td>MAX_AGE l</td>
</tr>
<tr>
<td>l</td>
<td>ALL</td>
</tr>
<tr>
<td></td>
<td>RAND_USC</td>
</tr>
<tr>
<td></td>
<td>IFL prob, l, l</td>
</tr>
<tr>
<td></td>
<td>SCR_Z l</td>
</tr>
<tr>
<td>op</td>
<td>TIE_RAND</td>
</tr>
<tr>
<td></td>
<td>TIE_SCR</td>
</tr>
<tr>
<td>prob</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>70</td>
</tr>
</tbody>
</table>
Lessons – Case Study 1

- Existing local search heuristics were broken down into components
- These heuristics return a variable to flip, not a value or ‘score’
- Local search heuristics evolved here, rather than constructive heuristics
Heuristic Generation Methodologies

Case Study 2
CASE STUDY 2

- One Dimensional Bin Packing
- Burke, Hyde, Kendall, and Woodward 2007
- Heuristics can be evolved that are specialised to different types of problems
- Extended to two dimensional packing heuristics in Burke, Hyde, Kendall, and Woodward 2010
The Bin Packing Problem

- Pack all the pieces into as few bins as possible
The Bin Packing Problem Set

- Online
- 7 problem classes
- Bin Capacity 150
- 120 items

7 Training sets  
7 Validation sets
GP Parameters Outline

- 50 generations
- 90% crossover
- 10% reproduction

- Functions and terminals:
  - Bin Capacity
  - Bin Fullness
  - Piece Size
  - +, -, *, %, ≤

- 1000 population
- Fitness proportional selection
Evolving Bin Packing Heuristics
Illegal Heuristics

- Permitted
- High penalty
- The system evolves an understanding of the rules
Results - Specialisation of Heuristics

![Diagram showing specialisation of heuristics]

- super-super-class
- super-class
- class
- Class $C_{30-49}$
- Class $C_{70-89}$
Results - Specialisation of Heuristics

![Diagram showing the specialisation of heuristics with classes and super-classes. The diagram has a top section labeled 'super-super-class', with a middle section divided into 'super-class' and 'class' categories, and a bottom section also divided into 'class' categories. Arrows indicate the flow or relationship between these classes.](image-url)
Results - Robustness of Heuristics

- all legal results
- some illegal results
Example of an evolved heuristic

- Heuristic evolved on instances with the widest distribution
- Tested on instances with piece sizes between 10-29

The heuristic performs very badly, by putting just one piece into each bin.
Example of an evolved heuristic

- The heuristic always scores the empty bin as the best

\[
\frac{2S + F}{S + F} + \frac{C}{\left(\left(\frac{F}{C}\right) \leq (2C - F)\right)} + (C - S - F)
\]
Lessons – Case Study 2

- Heuristics can be **specialised** to specific types of sub problem
- Heuristics may not work at all on new instances if they contain different distributions of pieces
- The **training set must be carefully chosen** to ensure it represents every type of problem that the heuristic must solve in the future
Conclusion

- Presented three case studies which highlight different research issues
- Humans will (always?) still have a role in heuristic generation
- The hyper-heuristic automates the process of combining elements that have been chosen by humans
- Our role moves from designing heuristics to designing the search space in which the best heuristic is likely to exist
References


- Alex S. Fukunaga. 2008. Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1) p31-61

References

- Hyper-heuristic bibliography online
  - http://www.cs.nott.ac.uk/~gxo/hhbibliography.html

- The Cross-domain Heuristic Search Challenge (CHeSC)
  - http://www.asap.cs.nott.ac.uk/chesc2011/