
Metaheuristic Design Patterns
3rd ECADA Workshop
GECCO
Genetic and Evolutionary Computation Conference
2013
Amsterdam, The Netherlands
July 06-10, 2013
acm

Metaheuristic Design Patterns

jerry.swan@cs.stir.ac.uk
krzysztof.krawiec@cs.put.poznan.pl
john.woodward@cs.stir.ac.uk

July 07, 2013

Premature Optimization

Claim: metaheuristic *research* is trapped in a local minimum.
Why?

1. Conceptual Parsimony - a historical preference for metaheuristics to be 'top-down, easily-described'.
2. Still in the software 'stone age' - everyone working in their own 'silo'; no metaheuristic analog of 'programming in the large'.

Consequences of Conceptual Parsimony

- **Claim**: we need to move towards *much* more sophisticated problem descriptions and solution representations, rather than the black boxes of the traditional hyper-heuristic 'domain-barrier'.
- In particular, we need to do a much better job of *knowledge engineering*, e.g. incorporating 'domain-independant domain knowledge'.
- An elementary example is the incorporation of nontrivial algebraic relationships between states and/or operators (see 'Representative' pattern later on).

The Software Stone Age - Consequences

- Parameter-tuning and algorithm selection are a manual processes, rather than being an integrated part of the researcher's toolset.
- Metaheuristic hybridization is traditionally done by hand, rather than via e.g. meta-learning or systems self-assembly.
- Automated Design of Algorithms (ADoA) researchers attempt to address this issue at the 'hyper-heuristic' level
- . . . but to some extent suffer the same problem 'one level higher-up'.

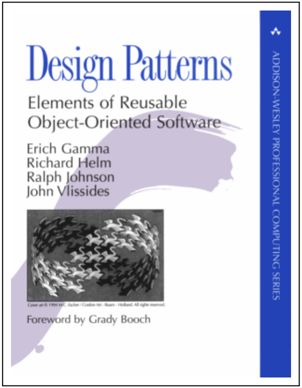
Software Development has already been here ...

- In the 1940s, a 'large' program might have 500 assembly language instructions. Modern techniques mean that codebases with $> 10^6$ LOC can be successfully maintained by distributed development teams.
- This is largely due to **modularity**. A modular system can be factored into parts that are *loosely-coupled*, i.e. their external connections are few relative to their internal complexity.
- Modularity is obviously a key concern in 'automatic programming'.
- Most interestingly, the average software engineer *has got quite a lot better at modularity*, quite recently ...

What are the drivers for progress in software development?

- **Education**: increasingly sophisticated design methodologies have become part of the common development culture.
- By contrast, much of our hard-earned knowledge as metaheuristic researchers has been gained in an *ad hoc* fashion, since relatively recent advances (∼5-10 years) are distributed across hundreds of research papers and have not yet been consolodated as ‘common practice’.
- **Componentization**: software is an ideal mechanism for describing concepts in the abstract (e.g. with pure functions or via subtype, parametric or *ad hoc* polymorphism).
- Claim we should do this with metaheuristic concepts on a much wider scale than we are currently doing.

The ‘Design Patterns’ Revolution



- One of the biggest step-changes in the overall quality of software engineering happened in 1994.
- ‘Design Patterns’ revolutionised software design and implementation.
- Over 500,000 copies sold.

Example - The Observer Pattern - 1

Intent
Define a one-to-many relationship between a subject and its observers, so that observers are updated automatically when the subject changes state.

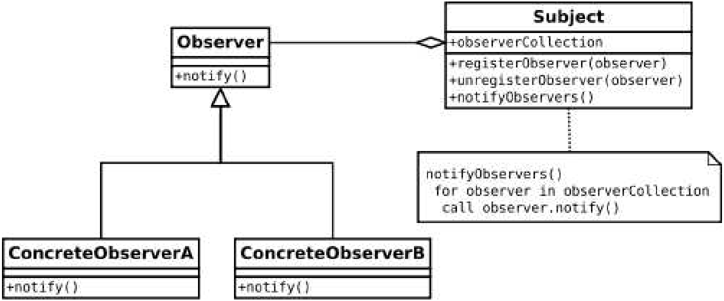
Motivation
To maintain consistency between co-operating entities.

Applicability
When a change to a subject is of interest to distributed and/or heterogenous observers.

Examples
A Model-View-Controller architecture can be greatly simplified by using the Observer pattern.

Consequences
Modularizes the update mechanism and reduces domain-specific coupling between subject and observers.

Example - The Observer Pattern - 2



More formally, a Design Pattern . . .

1. Addresses some repeatedly-observed design problem.
2. Describes the problem; the solution; when to apply the solution; consequences.
3. Gives implementation hints and examples.
4. The solution is an abstract component (or more generally a network of components) for solving the problem.
5. The generalized solution is customized to the context at hand.

Those steps seem rather familiar . . .

- As metaheuristic researchers, we use our own expert knowledge to perform these steps all the time.
- **Proposal**: Techniques of ubiquitous value should be explicitly documented as Design Patterns.
- The least we might aspire to is to codify repeated patterns (including more recent advances) as a kind of 'Metaheuristics 102': a cookbook for researchers in the field.

“That’s Impossible/Trivial . . .”

- This has of course already been done on a number of (less overtly Design Pattern-centric) levels.
- Standard EC or LS framework is clearly a kind of design pattern, customised (mainly by hand) to problem-specific needs.
- Krasnogor used a pattern-language to define families of memetic algorithms [4].
- PMML - A generic language for describing classifiers [3].
- PDDL - Generic planning DSL [2].

Example - Selection

Intent
Choose members of a collection according to some context-specific criterion.

Motivation
Original motivation comes from EC, where the context is defined as a quantitative fitness measure.

Applicability
When it is desirable to promote certain features of a population.

Collaborations
May benefit from access to an archive of previous items.

Consequences
Generally acts as an intensification mechanism.

The Pattern Catalog

The original GOF categories were:

- Creational – concerned with generation.
- Structural – concerned with composition.
- Behavioural – concerned with interaction/division of responsibility.

Some additional categories for metaheuristics ...

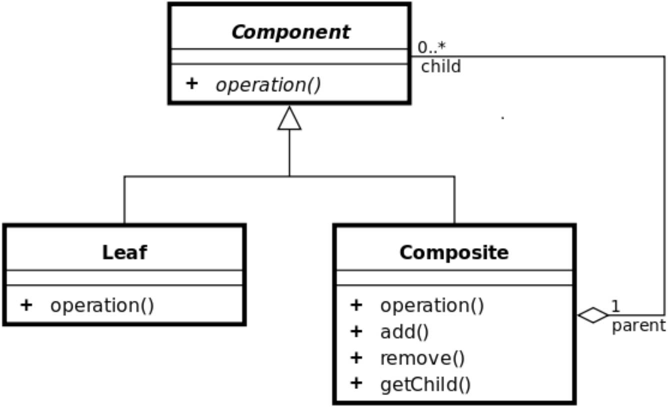
- Semantic – Representative, Repair, Locality.
- Methodological – Templates for: the design of experiments; performance of statistical tests ...
- Metrics – Euclidian and other norms, Jaquard, Mutual information, MDL ...)

Example metaheuristic patterns

Here are some examples of metaheuristic design patterns:

- **Structural**: Composite Metaheuristics. Relation=Graph=Matrix. Archive. Structured Population (e.g. Spatial and Island Models).
- **Behavioural**: Template method metalearning. Blackboard. Stigmergy (c.f. Visitor). Sampler.
- **Semantic**: Representative. Repair. Locality.

Examples: 1. The ‘Composite’ Pattern



Hyper-heuristics as Composites

- From [8], we can recursively define a hyper-heuristic as:

$$
\begin{aligned}
\mathcal{H}_0 &: ([S] \times [S \to S]) \to ([S] \times [S \to S]) \\
\mathcal{H}_1 &: ([S] \times [\mathcal{H}_0]) \to ([S] \times [\mathcal{H}_0]) \\
&\ldots \\
\mathcal{H}_n &: ([S] \times [\mathcal{H}_{i<n}]) \to ([S] \times [\mathcal{H}_{i<n}])
\end{aligned}
$$

- A metaheuristic such as steepest-ascent hillclimbing can be considered as $\mathcal{H}_0$ with an operator list of length 1.
- Hence, by the usual definition of hyper-heuristic, it can be seen as a composite metaheuristic, i.e. it IS-A metaheuristic that USES-A list of metaheuristic.

Examples: 2. The Template Method Pattern

- The Template Method pattern divides a framework into fixed and variant parts.
- The fixed parts orchestrate the behaviour of the variant parts.
- Simple example: Quicksort with variant partition function.

```
qsort( arr : Array[ T ] ) =
    qsort( lesser ) ++ equal ++ qsort( greater )
where
(lesser,equal,greater) = partition( arr )
```

Template Method Hyper-heuristics

- If we can express an algorithmic framework in template method terms, then we can learn good implementations for the variant parts.
- By ‘good’, we mean ‘biased towards the distribution to which they are exposed’.
- If our algorithms are metaheuristics, such meta-bias is actually a necessity (by NFL) [9].
- Successfully demonstrated this approach for GA selection and mutation operators ([12, 11]).

Examples: 3. Blackboard - Motivation

- Blackboard architectures offer a generic solution mechanism for problems which are:
  - Data-driven\opportunistic.
  - Not easily solved by an *apriori* fixed strategy.
- These are certainly properties of hyper-heuristic search.
- Concurrency support built-in to the architecture.

Blackboard - Implementation and Consequences

- Concurrent agents share search trajectories via a *workspace*.
- Places *minimal* restrictions on agent (metaheuristic) implementation (selective vs generative, constructive vs perturbative etc).
- Facilitates metaheuristic *racing*, interleaving of offline and online activites etc [8].
- Readily supports varying process granularities and transaction costs.
- Has potential as a *very* high-level interoperability mechanism.

Example: 4. Representative

Intent
Transform data (solutions/operators/algorithms) into a canonical (or normal) form.

Motivation
Such a transformation can be used to reduce the size of the search space.

Applicability
When it is possible to define a (relatively efficient) set of transformations.

Examples

- Polynomials can be represented by their roots [10]. See also [5].
- Operator sequences that form a *monoid* can be reduced to normal via the Knuth-Bendix algorithm [7].

The real goal - using Patterns in ADoA

- As ADoA researchers, the real advantages will come if we can obtain a *declarative* description of our patterns, i.e. one that is ammenable to *automated reasoning*.
- What we might want from our formalization?
  - Based on function signatures and contracts.
  - Strongly (even dependantly?) typed

Selection - signature and semantics

```
select( pop : List[T], howMany : ℕ₁ ) : NonEmptyList[ℕ]
require( !pop.isEmpty )
ensure(result.len == howMany && ∀ i∈result, i<pop.len)

selectProportional( pop : List[T], howMany : ℕ₁,
    fitnessFn : T → ℝ⁺, random : Random ) : NonEmptyList[ℕ]

selectTournament( pop : List[T], howMany : ℕ₁,
    fitnessFn : T → ℝ⁺, random : Random,
    tournamentSize : ℕ₁ ) : NonEmptyList[ℕ]
```

1. Strong typing helps enforce semantics.
2. Variant implementations may require extended signatures.

Towards ‘Metaheuristics in the large’ . . .

- Computer scientists don’t design algorithms ‘at random’ . . .
- Good researchers don’t hybridize metaheuristics ‘for no reason’ . . .
- How can we better learn the criteria for good designs?
- One starting point is to think about possible ‘Universal’ features of metaheuristic components . . .

Blum's I&D Frames

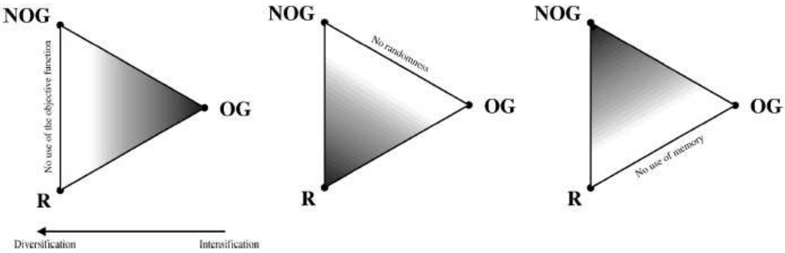


Fig. 18. The *I&D frame* provides a unified view on intensification and diversification in metaheuristics (**OG** = I&D components solely guided by the objective function, **NOG** = I&D components solely guided by one or more function other than the objective function, **R** = I&D components solely guided by randomness).

Declarative Representation

- Static features
  - Signature.
  - Semantics.
  - Fan-in and fan-out.
  - I&D Frame coordinate position.
  - Units and dimensions? (c.f. Mars Climate Orbiter ‘metric mixup’)
- Dynamic features
  - Induced semantics.
  - I&D Frame coordinate position (e.g. as modified by RL).

Other suggestions . . . ?

Assembling larger systems

- Given declarative descriptions of components, we can combine these descriptions to indicate how assemblies of these components might behave.
- For example, an Iterative Local Search procedure that uses SA as an operator will have higher diversity and randomness components than one that uses a hillclimber.
- This process applies recursively: for example, we can consider the construction of an SA metaheuristic to be parameterized by its source of 'thermal noise' values.
- By using a Markov chain c of order 2 as the source, then an instance of SA that uses c would have a lower I&D frame value for randomness than one that used a Random Number Generator for which samples are independant.

An ADoA Component Framework

- As per Pappa et al. [6], the time is right for principled integration of ML and ADoA.
- Having a common repertoire of features to describe components facilitates the creation of learning algorithms that can operate at heirarchical levels of modularity.
- Modules and learning algorithms can be shared across the community, moving metaheuristic design closer to electronic and software component design [1].

Thanks for listening

Questions?

References I

Amnon H. Eden, Amiram Yehudai, and Joseph (Yossi) Gil.
Precise specification and automatic application of design patterns.
pages 143–152. Society Press, 1997.

Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld.
PDDL - The Planning Domain Definition Language.
Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Alex Guazzelli, Michael Zeller, Wen-Ching Lin, and Graham Williams.
PMML: An open standard for sharing models.
The R Journal, 1(1):60–65, 2009.

References II

N. Krasnogor.
Handbook of Natural Computation, chapter Memetic Algorithms.
Natural Computing. Springer Berlin / Heidelberg, 2009.

Trent McConaghy and Georges Gielen.
Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions.
In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 855–862, New York, NY, USA, 2006. ACM.

References III

GiseleL. Pappa, Gabriela Ochoa, MatthewR. Hyde, AlexA. Freitas, John Woodward, and Jerry Swan.
Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms.
Genetic Programming and Evolvable Machines, pages 1–33, 2013.

Jerry Swan, Martin Edjvet, and Ender Özcan.
Augmenting metaheuristics with rewriting systems.
Technical Report CSM-197, Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, Scotland, January 2014.

References IV

Jerry Swan, John Woodward, Ender Özcan, Graham Kendall, and Edmund Burke.
Searching the hyper-heuristic design space.
Cognitive Computation, 2013.

John Woodward and Jerry Swan.
Why classifying search algorithms is essential.
In *2010 International Conference on Progress in Informatics and Computing. (PIC-2010)*, 2010.

John R. Woodward and Ruibin Bai.
Canonical representation genetic programming.
In *GEC Summit*, pages 585–592, 2009.

References V

John R. Woodward and Jerry Swan.
The automatic generation of mutation operators for genetic algorithms.
In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, pages 67–74, New York, NY, USA, 2012. ACM.

John Robert Woodward and Jerry Swan.
Automatically designing selection heuristics.
In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 583–590, New York, NY, USA, 2011. ACM.