1. **Template Method Hyper-heuristics**
2. The Composite Design Pattern
GECCO -

John Woodward jrw@cs.stir.ac.uk

Jerry Swan jsw@cs.stir.ac.uk

Simon Martin spm@cs.stir.ac.uk

Outline

Template method hyper-heuristics

– Sets of algorithms

– Type signatures

– Example Genetic Algorithm mutation operator

– Consequences

Composite design pattern

– hyper-heuristic

– Ensembles.

Template Method Hyper-heuristics

- **Template Method** is a design pattern.
- Some methods of a class have specified type signatures but no implementation (body) i.e. **abstract class**.
- The abstract class(es) can be **supplied later** by another programmer.
- OR can be supplied by Automatic Programming Technique such as **Genetic Programming**.

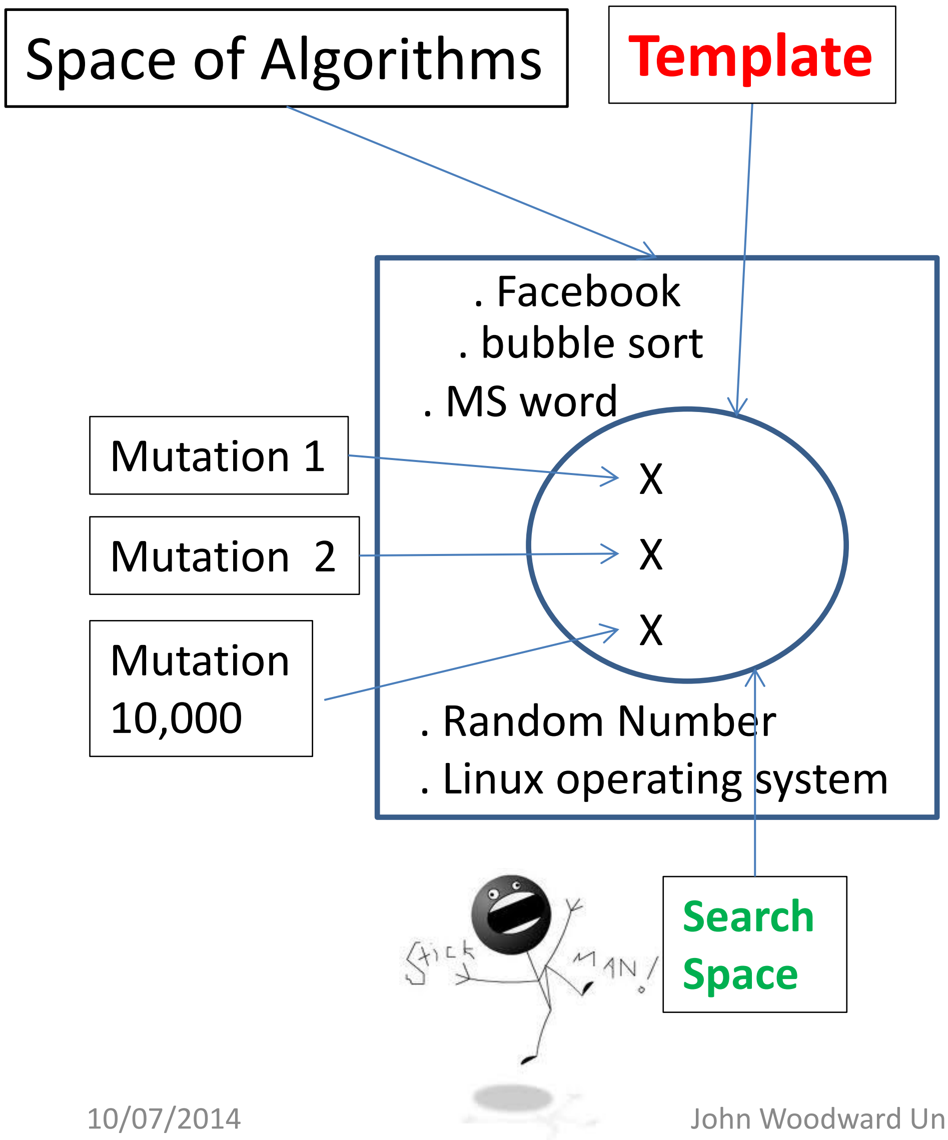
Proposing Sets of Algorithms

In the *template method*, one or more algorithm steps can be **overridden** by subclasses to allow **differing** behaviours while ensuring that the **overarching** algorithm is still followed.

- **Concrete** methods/classes **constrain** the **behaviour** of the program.
- **Abstract** methods/classes allow variation.

A template is a **skeleton**.

One Man ….Many Algorithms



1. Challenge is defining an algorithmic framework (**set**) that **includes** useful algorithms, and **excludes** others.

2. Let Genetic Programming **select the best algorithm for the problem class at hand**. **Context!!!** Let the data speak for itself without imposing our assumptions.

Type Signatures

- H = history, P = population

$$
\begin{aligned}
initialization &: Void \to P \\
selection &: P \times H \to P \\
variation &: P \times H \to P \\
succession &: P \times H \to P \\
termination &: H \to Bool
\end{aligned}
$$

Evolutionary Algorithm Template

```
procedure evolve
begin
  pop = initialization()
  history = []
  repeat
    parents = selection( pop, history )
    offspring = variation( parents, history )
    pop = succession( offspring, history )
    history = history.append( pop )
  until termination( history )
end
```

Example – mutation for GA.

- **Examples:** one point and uniform mutation.
- **Behaviour:** Given a bit string of length n, return a bit string of length n.
- We could write another mutation operator.
- **NO NO NO** – lets let Genetic Programing DO ALL THE HARD (and boring) WORK.
- **Generate-and-test a Generate-and-test method**

Building a Space of Mutation Operators

Inc	0
Dec	1
Add	1,2,3
If	4,5,6
Inc	-1
Dec	-2

Program counter pc	2

WORKING REGISTERS

110	-1	+1	43	…	

INPUT-OUTPUT REGISTERS

-20	-1	+1	20	…	

A **program** is a list of instructions and arguments.

A **register** is set of addressable memory (R0,..,R4).

Negative register addresses means **indirection**.

A program can only **affect IO registers indirectly.**

+1 (TRUE) -1 (FALSE) +/- sign on output register.

Insert bit-string on IO register, and extract from IO register

Expressing Mutation Operators

Line	UNIFORM	ONE POINT MUTATION
0	**Rpt, 33, 18**	**Rpt, 33, 18**
1	Nop	Nop
2	Nop	Nop
3	Nop	Nop
4	**Inc, 3**	**Inc, 3**
5	Nop	Nop
6	Nop	Nop
7	Nop	Nop
8	**IfRand, 3, 6**	**IfRand, 3, 6**
9	Nop	Nop
10	Nop	Nop
11	Nop	Nop
12	**Ivt,−3**	**Ivt,−3**
13	Nop	**Stp**
14	Nop	Nop
15	Nop	Nop
16	Nop	Nop

- **Uniform mutation**

Flips all bits with a fixed probability.

4 instructions

- **One point** mutation

flips a single bit.

6 instructions

Why insert NOP?

We let GP start with these programs and mutate them.

In a Nutshell

- **Humans** design the **structure** of the program e.g. the for-loops (GP is bad at that) (INVARIANT)
- Let **GP** build the **body** of the for-loop (VARIANT).
- The final program is part **man made** and part **machine made**.
- We used the **Object Oriented** approach but could be expressed in terms of e.g. **Functional programming** (pass in a mutation operator).

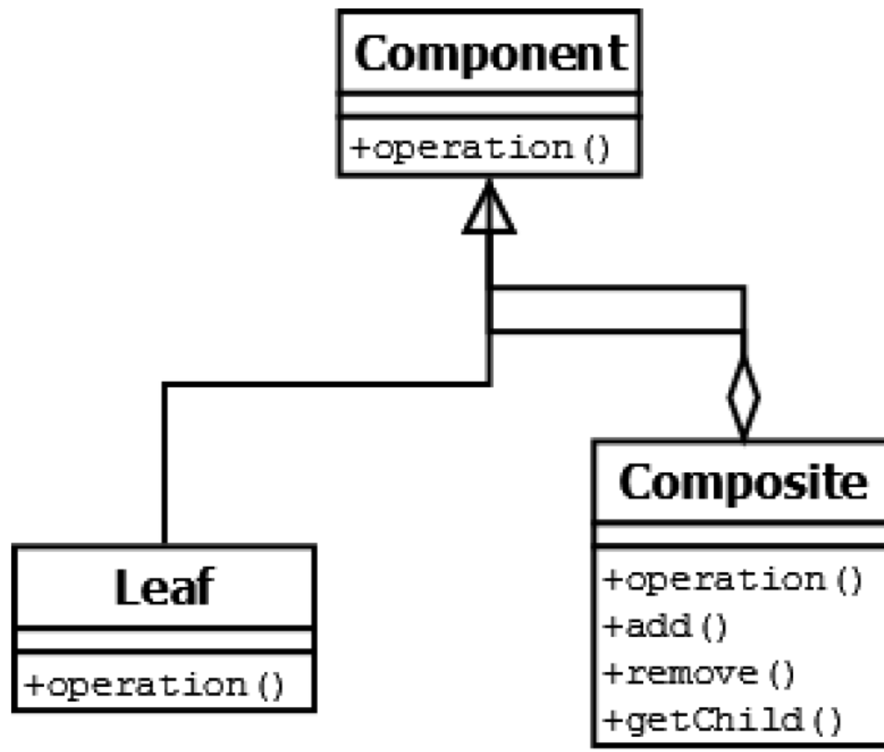
Consequences

1. Instead of proposing a **single algorithm**, “In this paper we propose a novel algorithm”…
2. We can now propose a set of algorithms, “In this paper we propose **10,000 algorithms**”
3. The resulting algorithm is **typically better** than a human designed algorithm.
4. If the problem changes, **we can instantly call** on Genetic Programming **again**.

The Composite Design Pattern

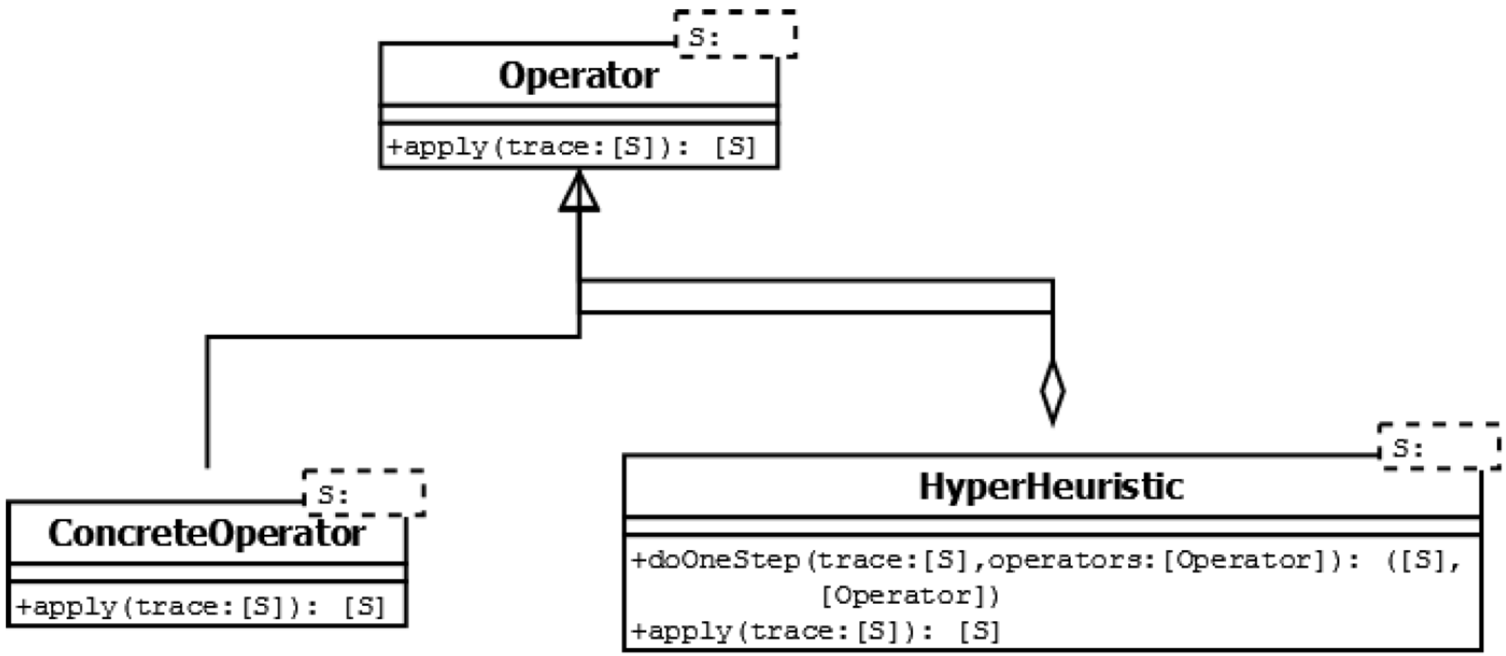
The composite pattern describes that a **group of objects** is to be treated in the same way as a single instance of an object.

- **Hyper heuristics**
- **Ensembles**



Hyper-Heuristics

- Heuristics to **choose** heuristics **H:[S]->[S]**
- Heuristics to **generate** heuristics **H:[O]->[O]**



Composite Hyper-heuristic

- Operator maps state to state. **O:[S]->[S]**
- Where [S] is a list (trace or history of states)
- Hyper-heuristic **H:[S]X[O]->[S]X[O]**
- Selective hyper-heuristics update former **[S]**
- Generative hyper-heuristics update latter **[O]**

Ensembles of Classifiers

1. How to **combine** the classier outputs to compute an overall classification?
2. How to **generate** multiple diverse classifiers to produce a well-performing ensemble?
3. How to **set the parameters** of machine learning algorithms?
4. How can we build high quality classifiers more efficiently in the new era of **big data** and **parallel processing**?

Combining Classifier Outputs

- **Majority vote**: The entire set of classifiers vote on a class, and the class which receives the most votes is taken.
- **Averaging**: If the outputs of each classier are a real number then the outputs can be averaged.
- **Weighted average**: Each classier is assigned a weight according to its `expertise'. When the averaging is done more emphasis is placed on the classifiers with a higher weight.
- **Algebraic combiners**: real-valued outputs of classifiers are combined through statistical expressions such as sum, mean, product, median, minimum, maximum.

Generating Diverse Classifiers

- **Bagging** (bootstrap aggregation)random samples (usually with replacement) taken from the original dataset
- **Boosting** adjusts the probability of sampling misclassified data. Thus, misclassified data is more likely to be considered in the training of subsequent classifiers.
- **Stacked Generalization** trains multiple levels of classifiers.
- **Mixture of Experts** generates several classifiers whose outputs are combined through a rule which typically trained using the expectation maximization (EM) algorithm.

Consequences

1. Generation of **Diverse** Classifiers
2. Statistically better **behaviour**
3. Integration of different types of classifier
4. Learning Classifier Systems
5. **Confidence**
6. Levels of Measurement
7. **Statistics and Machine Learning**
8. Classifier Outputs as Features
9. Ensembles of Linear Classifiers
10. **Big Data and Parallelism**

Surrogate Fitness Function

- We may **substitute** an objective function (supplied by the domain expert) with a surrogate fitness function.
  1. It is expensive to execute
  2. It is not known explicitly
  3. It is rugged/multimodal.

Closing Statement

- A **catalogue of design patterns** (with motives and consequence) could stop us **reinventing the wheel.**
- **Definition** – do we need one? Even **informal**?
- Metaheuristics are very ad hoc – why?
- **Machine learning – training and testing phase**.
- **Standardise** terminology? **(Re)-educate**?
- Thank you – questions? ☺