Benchmarks that Matter for Genetic Programming

John R. Woodward jrw@cs.stir.ac.uk

Simon Martin spm@cs.stir.ac.uk

Jerry Swan jsw@cs.stir.ac.uk

Outline

- Need for **non-arbitrary** benchmarks
- Metaheuristics and **problem classes**
- Recent **Theorem** about performance.
- **Base** and **meta** level (sampling) learning
- **Type signatures**
- **Matching** metaheuristics to problem classes.
- Automatic design of algorithms is a **natural solution.**

Argument for Better Benchmarks

1. Machine learning has become **disconnected** from the **communities of domain experts**.
2. GP lacks "**standardized**" **benchmarks**.
3. **Theoretical results** link with problem classes.
4. Automatic design of algorithms (**meta-learning and hyper-heuristics**) can bridge this gap.

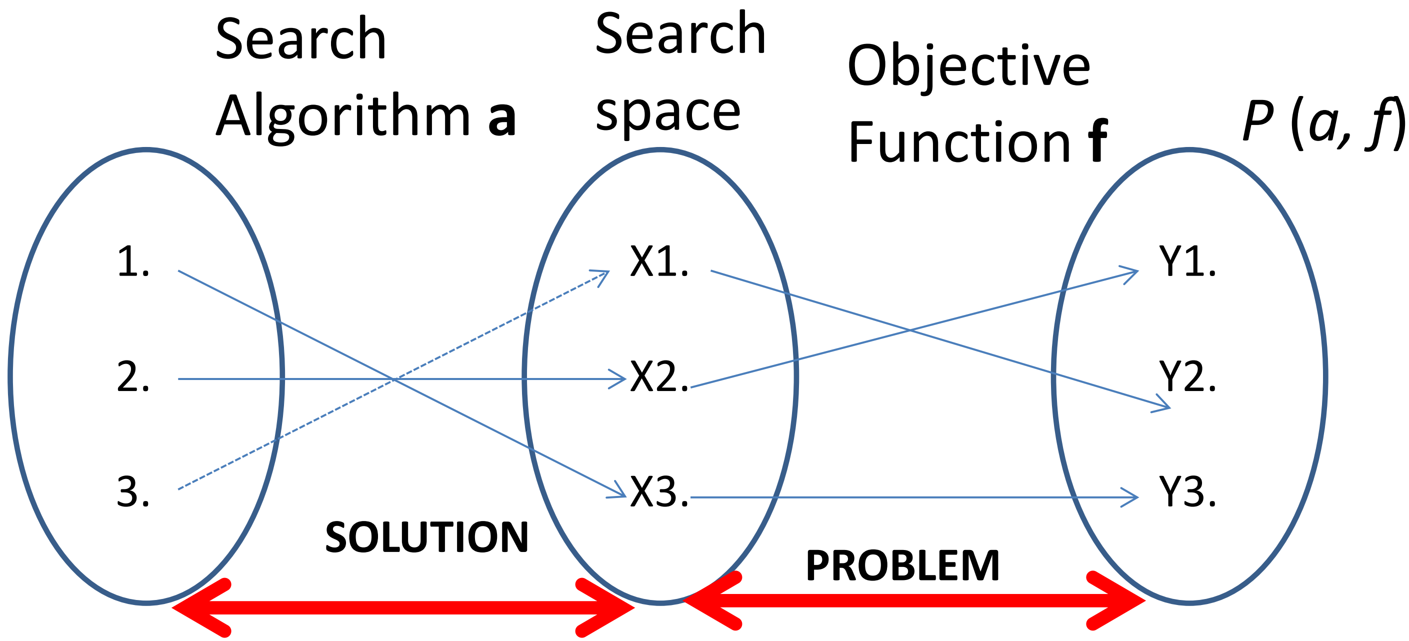
Metaheuristics and Problem Classes

- A metaheuristics **samples** (stochastically) a **search space** of possible solutions.
- A metaheuristics is a conditional **probability distribution** over the search space.
- A **set of problem instances** come from a probability distribution.
- There is therefore a **link** between a metaheuristic and a problem class

Base and Meta/Hyper Level.

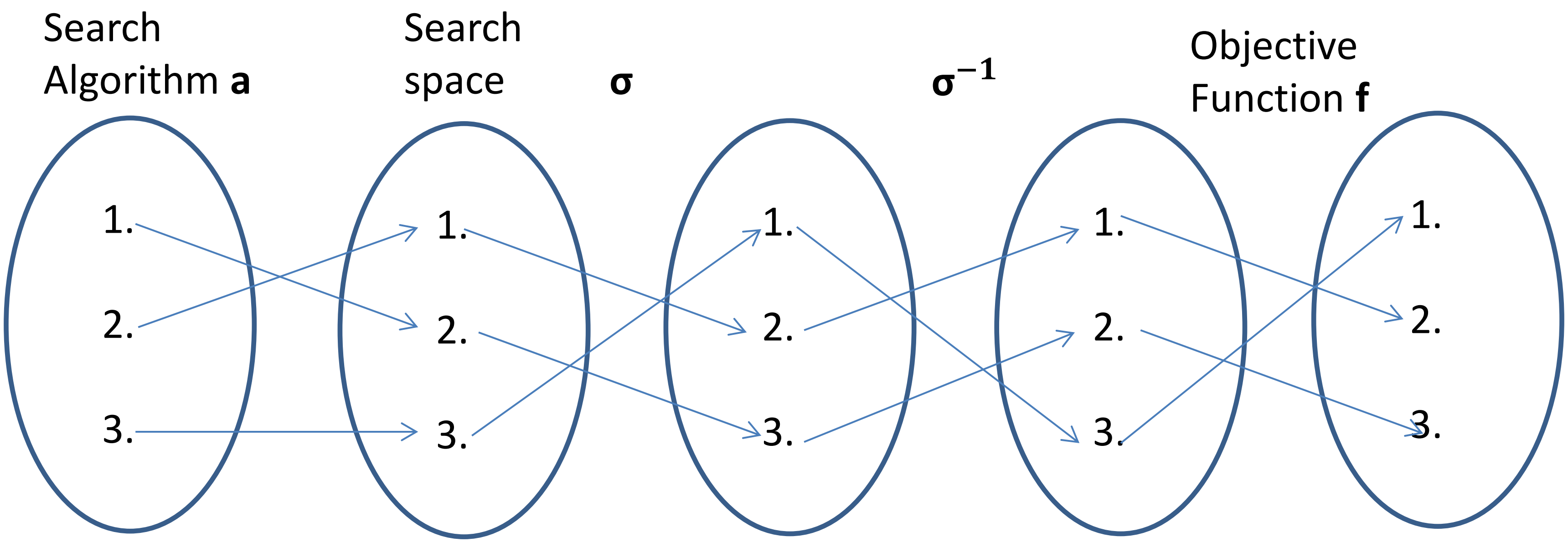
- At the **base level** we are learning about a function.
- At the **meta level** we are learning about the probability distribution of functions.

Theoretical Motivation 1



1. A **search space** contains the <u>set of all possible solutions</u>.
2. An **objective function** determines the <u>quality of solution</u>.
3. A **search algorithm** determines the <u>sampling order</u> (i.e. enumerates i.e. without replacement). It is a (approximate) permutation.
4. **Performance measure** *P* (*a*, *f*) depend only on y1, y2, y3
5. **<u>Aim find a solution with a near-optimal objective value using a search algorithm.</u>** ANY QUESTIONS BEFORE NEXT SLIDE?

Theoretical Motivation 2



$P\,(a, f) = P\,(a\,\boldsymbol{\sigma}, \boldsymbol{\sigma}^{-1}\,f)$ $\qquad$ $P\,(A, F) = P\,(A\boldsymbol{\sigma}, \boldsymbol{\sigma}^{-1}F)$

P is a **performance measure**, (<u>based only on output values</u>).

$\boldsymbol{\sigma}, \boldsymbol{\sigma}^{-1}$ are a permutation and inverse permuation.

A and F are probability distributions over algorithms and functions).

F is a problem class. ASSUMPTIONS IMPLICATIONS

1. Algorithm **a** applied to function $\boldsymbol{\sigma\sigma}^{-1}\boldsymbol{f}$ (that is $\boldsymbol{f}$)
2. Algorithm **aσ** applied to function $\boldsymbol{\sigma}^{-1}\boldsymbol{f}$ precisely identical.

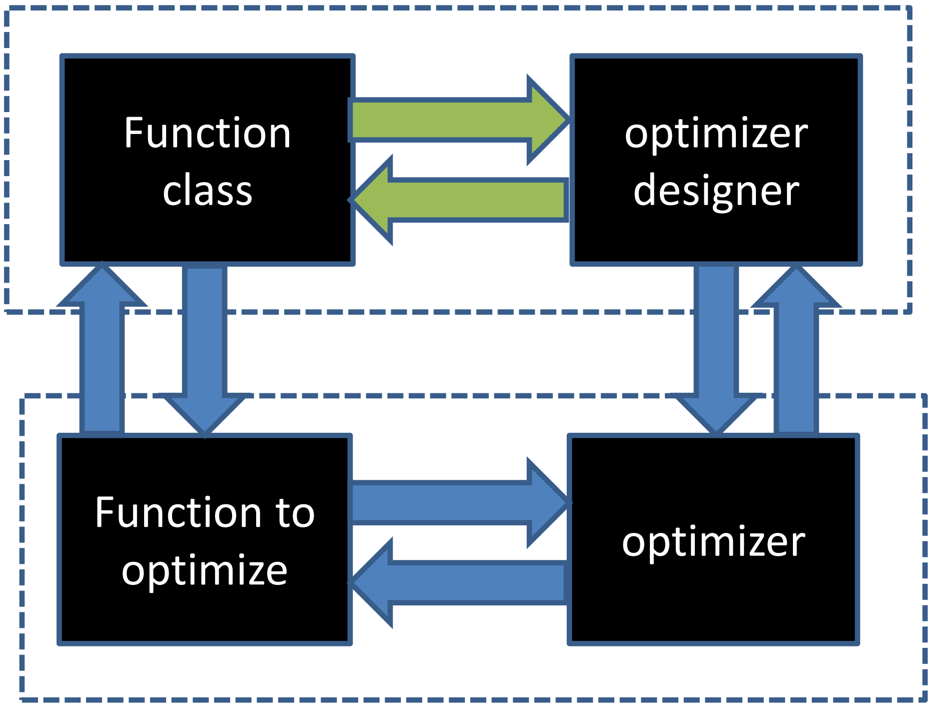
No Free Lunch Theorems (NFL)

- NFL (informally) states “two metaheuristics perform equally **over all problems**”
- What NLF really says is over a problem class, **some metaheuristics can perform better** than others (we are just talking probability distributions).
- A uniform distribution is a **special case** (and unrealistic?).

Meta and Base Learning

1. At the **base** level we are learning about a **specific** function.
2. At the **meta** level we are learning about the problem **class**.
3. We are just doing **"generate and test"** on **"generate and test"**
4. What is being passed with each **blue arrow**?
5. Training/Testing and Validation

Meta level



base level

Conventional optimizer

Compare Signatures (Input-Output)

Optimizer

- (B^n -> R) -> B^n

Input is an objective function mapping bit-strings of length n to a real-value.

Output is a (near optimal) bit-string

i.e. the solution to the problem instance

Optimizer Designer

- [(B^n -> R)] -> ((B^n -> R) -> B^n)

Input is a *list of* functions mapping bit-strings of length n to a real-value (i.e. sample problem instances from the problem class).

Output is a (near optimal) mutation operator for a GA

i.e. the solution method (algorithm) to the problem class

We are **raising the level of generality** at which we operate.
Give a man a fish and he will eat for a day, teach a man to fish and…

Black-Box Sampling

- If we **sample a function f1**
- f1(x1)= y1, f1(x2)=y2, what can we say f1(x3)=?
- If we **sample a function f2**
- f2(x1)= y1, f2(x2)=y2, what can we say f2(x3)=?
- If we experience a number of functions, the best we can do is make probabilistic inferences.
- The best we can do is give a **probability p(f)** that we think we are sampling function f.

Matching Metaheuristics to Problem Classes

- **TRAINING:** Given a set of algorithms to choose from, select the best (near optimal) for a set of problem instances (drawn from a probability distribution).
- **TESTING:** The resulting algorithm should perform well on a set of problem instances (drawn from the same probability distribution).
- We are using a machine learning algorithm (GP) to build an optimization algorithm.

New Benchmarks for GP

- Typically Genetic Programming is applied to problems requiring **synthesis of a function** e.g. a controller for a robot or function regression.
- Now we have a new set of problems (e.g. **optimization, combinatorial problems, TSP**)
- This is because we are operating indirectly on the search space using a hyper-heuristics methodology.

Generating Timetabling Problems

- A standard timetabling problem consists of a number of locations (rooms) time slots, examiners (lecturers) and students.
- Given one problem instance we can generate more similar instance.
  - Number of room should not vary.
  - Number of teachers may vary a little
  - Number of student will vary more

  By sensibly perturbing the given problem instance you can generate a set of similar problem instances.

Conclusions

- It is **difficult to infer** which optimization algorithm we should pick, given performance on arbitrary benchmarks.
- **Match** an optimizer to a probability distribution of problem instances.
- Meta-learning/hyper-heuristics **learn about the probability distribution** of problem instances.
- **New unseen benchmark instances** can be generated for **fair comparison**.