Hyper-heuristics generate heuristics for problem classes

Edmund Burke, Jerry Swan, John Woodward
http://www.cs.stir.ac.uk/~jrw/
jrw@cs.stir.ac.uk
University of Stirling.

Short Abstract.

Meta-heuristics sample a search space, with quality dictated by an objective function. **For any pair of metaheuristics, there is a pair of objective functions with identical performance**.

A similar statement can be made about **problem classes** (probability distributions over problem instances). The intuition behind this result is that a meta-heuristic can be viewed as a conditional probability over the search space and therefore this result can be considered as a **conservation law**.

The contribution is an implication that **meta-heuristics should be designed for a problem class**. A natural solution to the ``metaheuristic design problem'' is to employ **generative hyper-heuristics** to yield heuristics tailored to the problem class.

Full Length Abstract.

Meta-heuristics operate by sampling a search space of candidate solutions, with quality dictated by an objective function. By permuting a function to create a new function, and using the inverse permutation to create a new meta-heuristics, we can state that for any pairing of meta-heuristic and function instance, there exists a distinct pairing with strictly identical performance. A similar statement can be made problem classes (probability distributions over problem instances). An alternative intuition behind this result is that a meta-heuristic can be viewed as a conditional probability over the search space and therefore (since the probabilities over the search space sum exactly to one) can be considered as a conservation law. The contribution of this paper is an implication of this theoretical result which is that meta-heuristics should be designed for a problem class (probability distribution over problem instances). A natural solution to the ``metaheuristic design problem'' is then to employ generative hyper-heuristics to yield heuristics tailored to the problem class.

Talk In a Sound bite…

- A **problem class is a probability distribution over problem instances**. Your problem instances must come from some probability distribution (sounds like machine learning!!!).
- A **metaheuristics generate solutions with some probability distribution** over the search space of solutions. We want this to match the problem class
- **Hyper-heuristics** are a method to generate (meta)heuristics for your problem class (i.e. machine learning with training and test set)

Outline of talk

1. How should we sample a search space?
2. Thought experiment – can we shift bias?
3. Bias of problem class = bias of metaheuristic
4. This is (general) No Free Lunch!!!
5. Simple proof
6. Hyperheuristics can generate
7. Convergence at hyper/meta level.

Which cup is the pea under?

1. Endlessly fascinating for a young child.
2. But what about researchers?
3. **(see recommended paper later – metaphor exposed)**

Question How do we sample a search space?

- Randomly?
- Enumeration?
- Simulated Annealing?
- Bio-inspired?

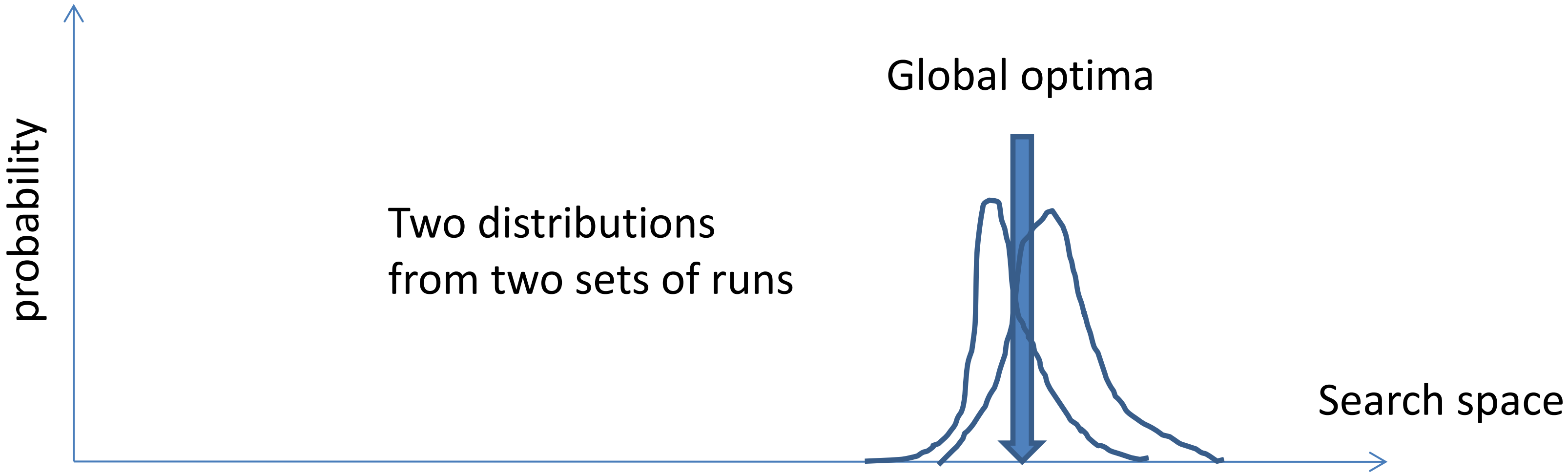
How do we sample a search space?

1. Randomly?
2. Enumeration?
3. Simulated Annealing?
4. Bio-inspired?

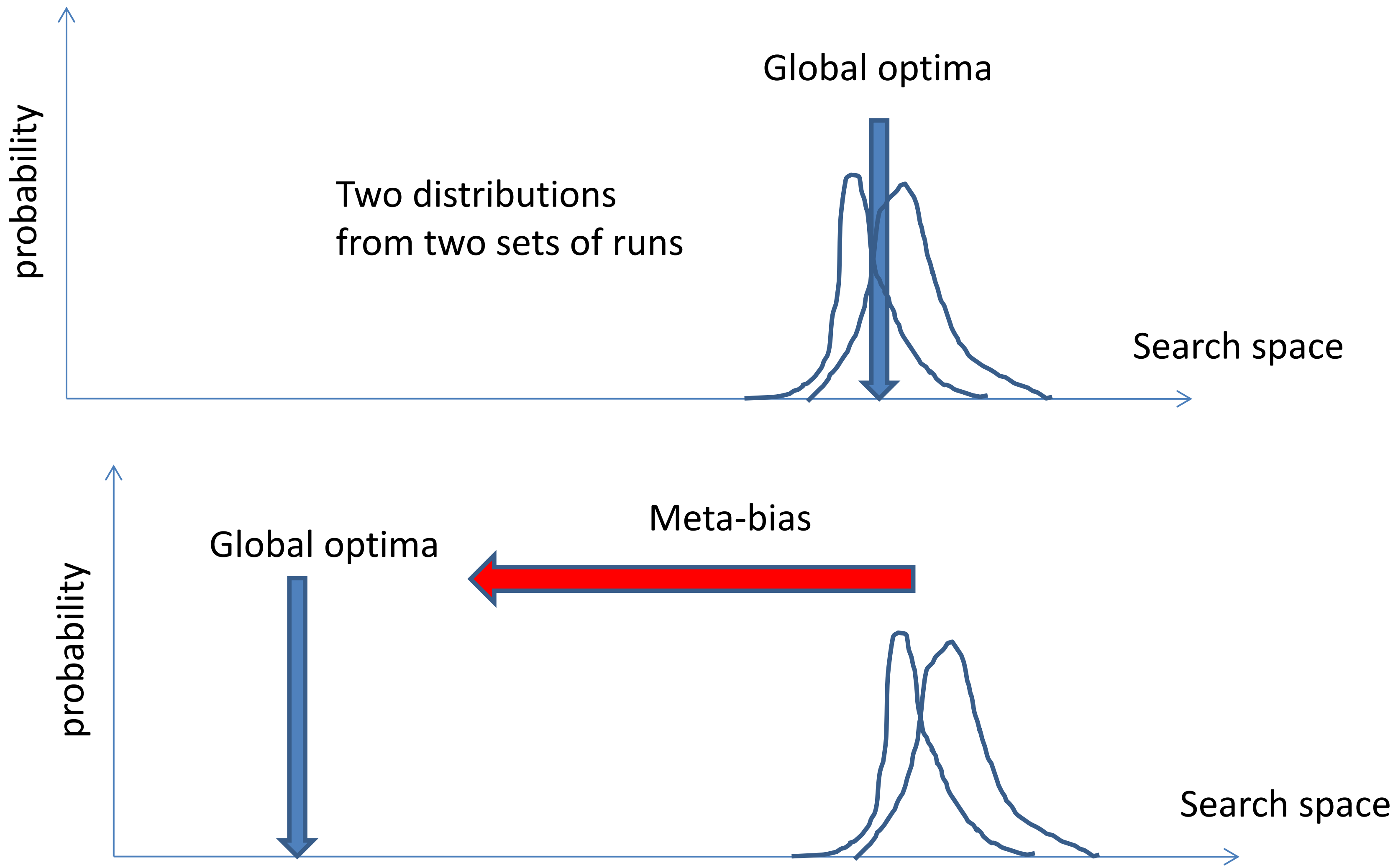
It **depends**, if the space has a high probability of

1. Random (incompressible)?
2. Has a known property?
3. Is unimodal?
4. ???

A Simple Thought Experiment 1



A Simple Thought Experiment 2



Bias

1. Bias is just a probability distribution over the search space. A metaheuristic is just a conditional probability (different implementations).
2. Bias come from choices e.g. mutation rate, cooling schedule, any parameters.
3. Tom Mitchell ("**The Need for Biases in Learning Generalizations**") – bias is necessary for learning.
4. By the same argument, meta-bias is necessary if we are to apply our optimization/machine learning algorithms to more than a single problem instance.

From The Original NFL Paper

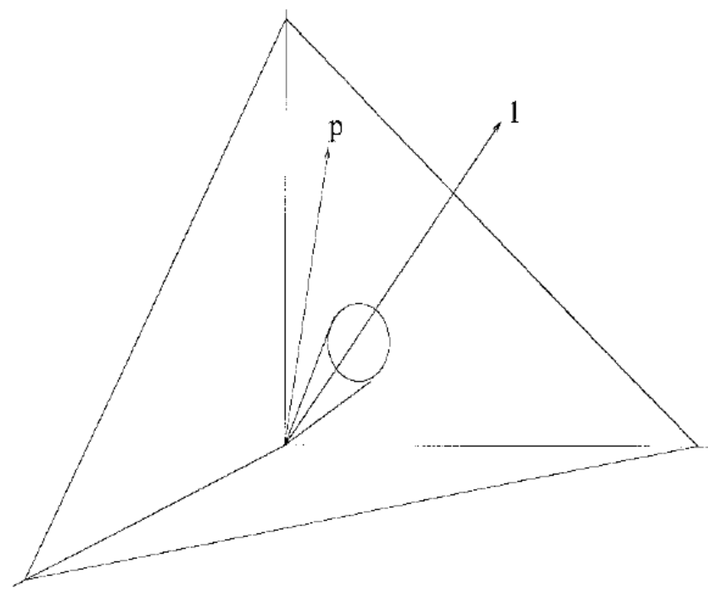
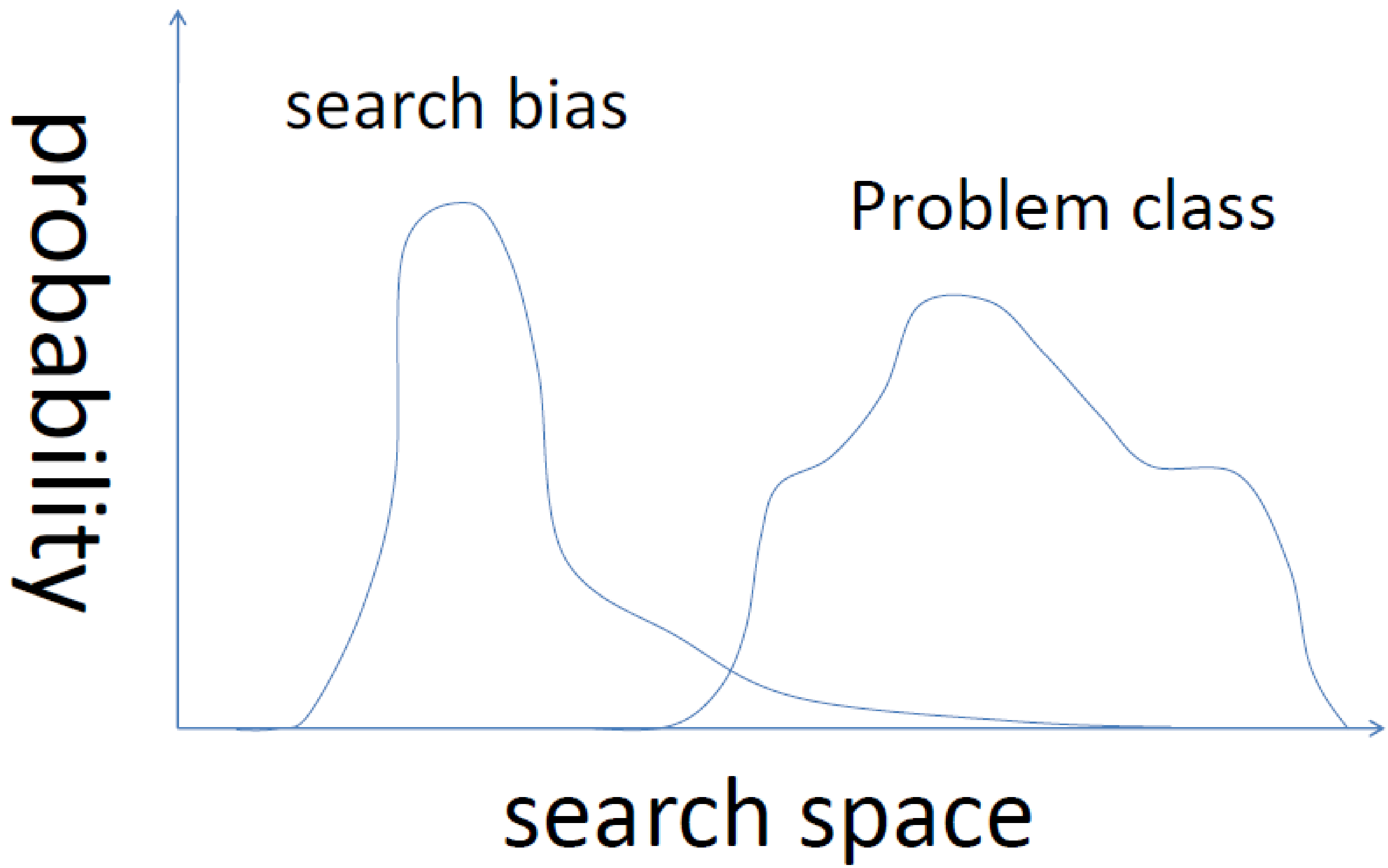
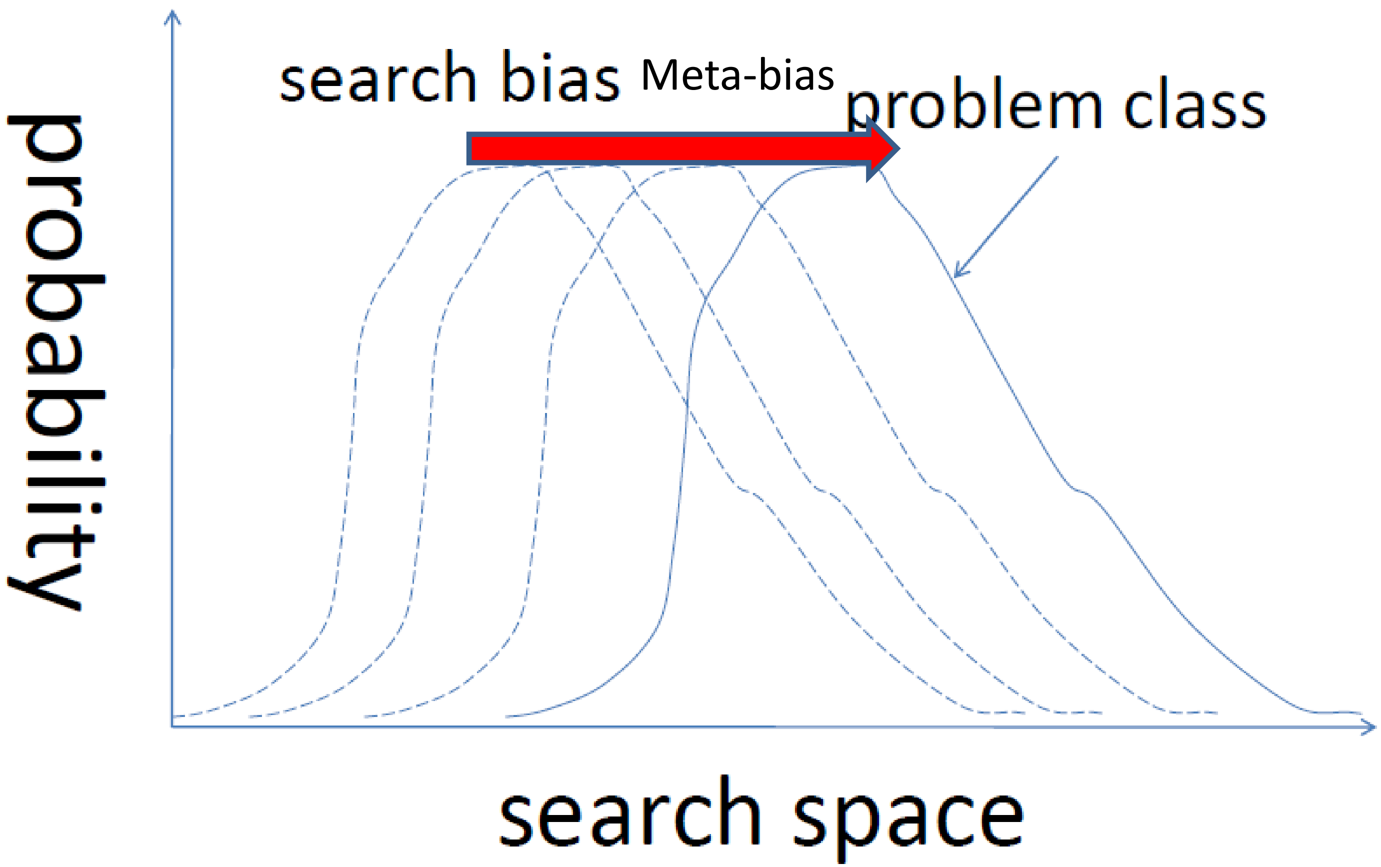


Fig. 1. Schematic view of the situation in which function space $\mathcal{F}$ is three dimensional. The uniform prior over this space, $\vec{\mathbf{1}}$, lies along the diagonal. Different algorithms a give different vectors v lying in the cone surrounding the diagonal. A particular problem is represented by its prior $\vec{\boldsymbol{p}}$ lying on the simplex. The algorithm that will perform best will be the algorithm in the cone having the largest inner product with $\vec{\boldsymbol{p}}$.

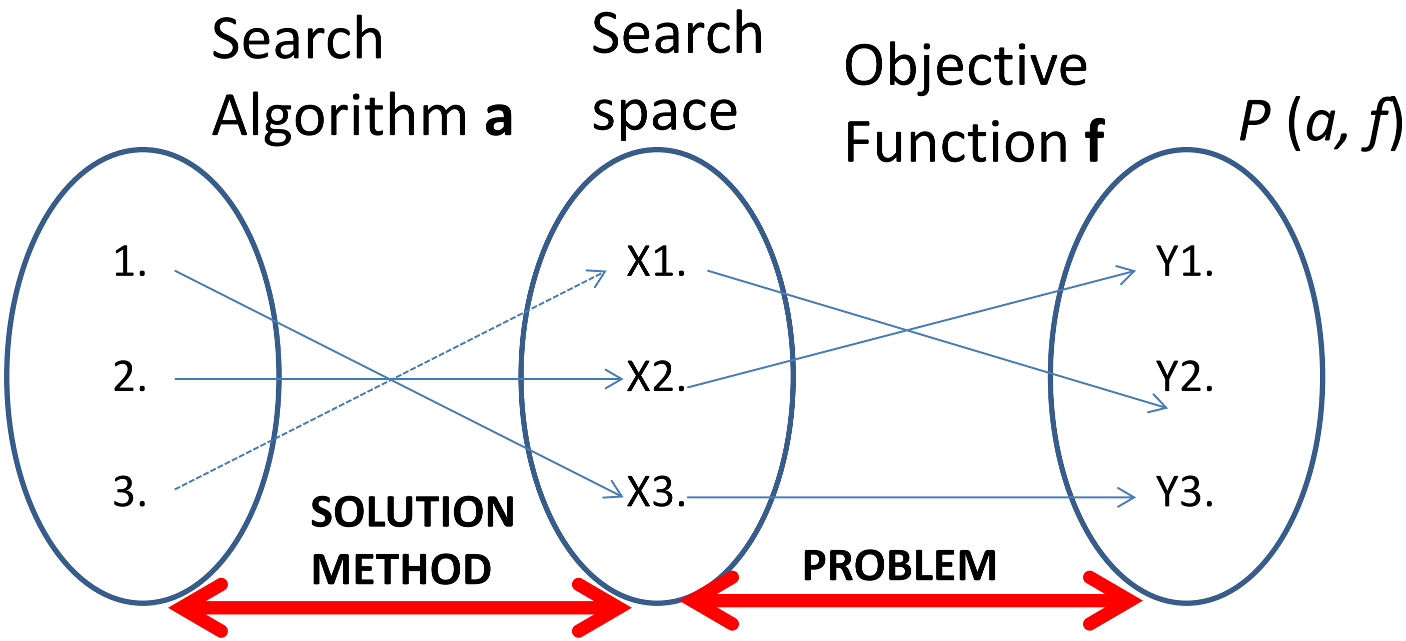
The Bias does not match the problem class –> poor performance



Meta-bias shifts search bias to match that of the problem class –> improving

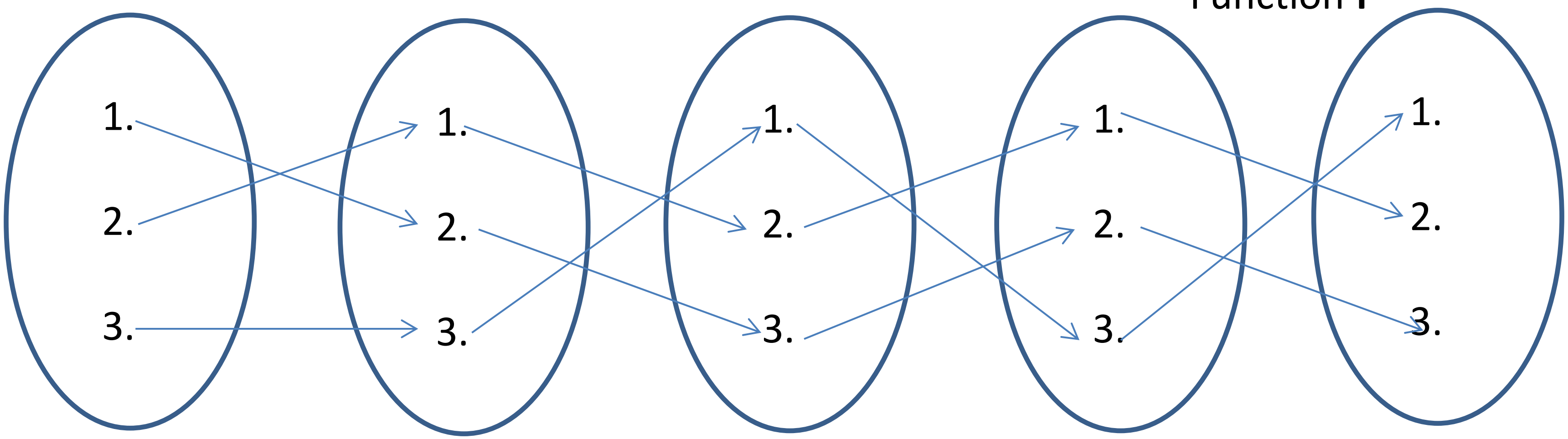


Theoretical Motivation 1



1. A **search space** contains the set of all possible solutions.
2. An **objective function** determines the quality of solution.
3. A **search algorithm** determines the sampling order (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. **Aim find a solution with a near-optimal objective value using a search algorithm.** 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>).

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 **a$\boldsymbol{\sigma}$** applied to function $\boldsymbol{\sigma}^{-1}\boldsymbol{f}$ precisely identical.

Is the No Free Lunch Theorem a Show-stopper?

- Lemma. Knowing **p(f)**, the probability of encountering an arbitrary function f, is equivalent to knowing **p(c|e)**, the probability of class membership c for an arbitrary example e.
- Do a thought experiment to confirm this for yourself (or ask me to go through example).
- In other words, the best we can do with meta-heuristics and black box functions is **align the associated probability vectors**,….but how?

Reinterpret “No Free Lunch”

- Often stated as “over all problems no algorithm does any better than any other”
- Also means, over biased problem class, correctly biased algorithms will perform better!
- -> We should not design algorithms in isolation to problem classes.
- -> Hyperheuristics is one way to generate algorithms tuned to a problem class.

Metaheuristics - the metaphor exposed

1. There are many metaphors for metaheuristics; insects, the flow of water, musicians playing together.
2. Do they offer insights?
3. Or are they more of a hindrance?
4. Automatic design, to some extent solves this problem.

Metaheuristics—the metaphor exposed

Kenneth Sörensen

Woodward, J. & Bai, R. (2009) **Why Evolution is not a Good Paradigm for Program Induction**; **A Critique of Genetic Programming**

Space of {NAND} programs

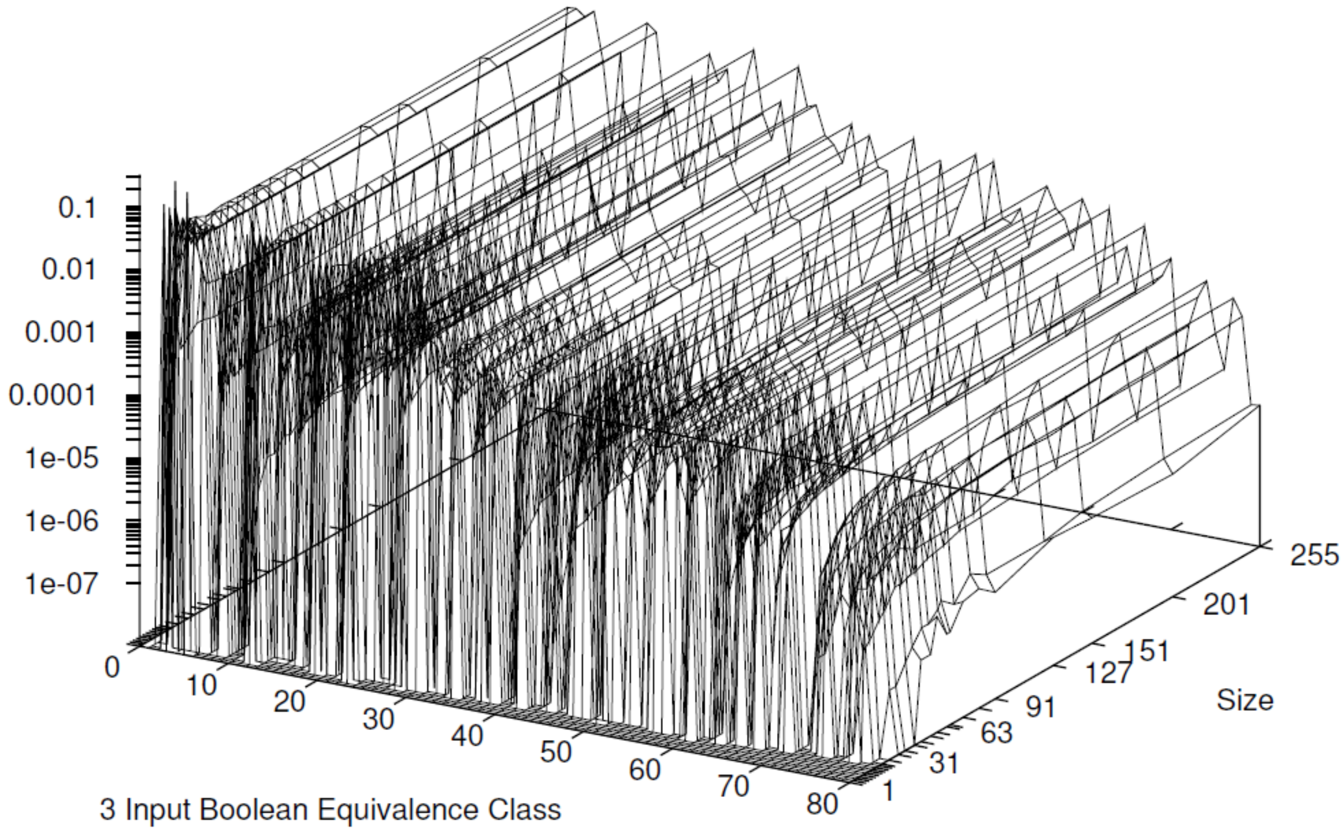


Figure 3: Proportion of NAND trees which yield each 3 input equivalence class.

Space of Programs {AND, OR, NAND, NOR, XOR}

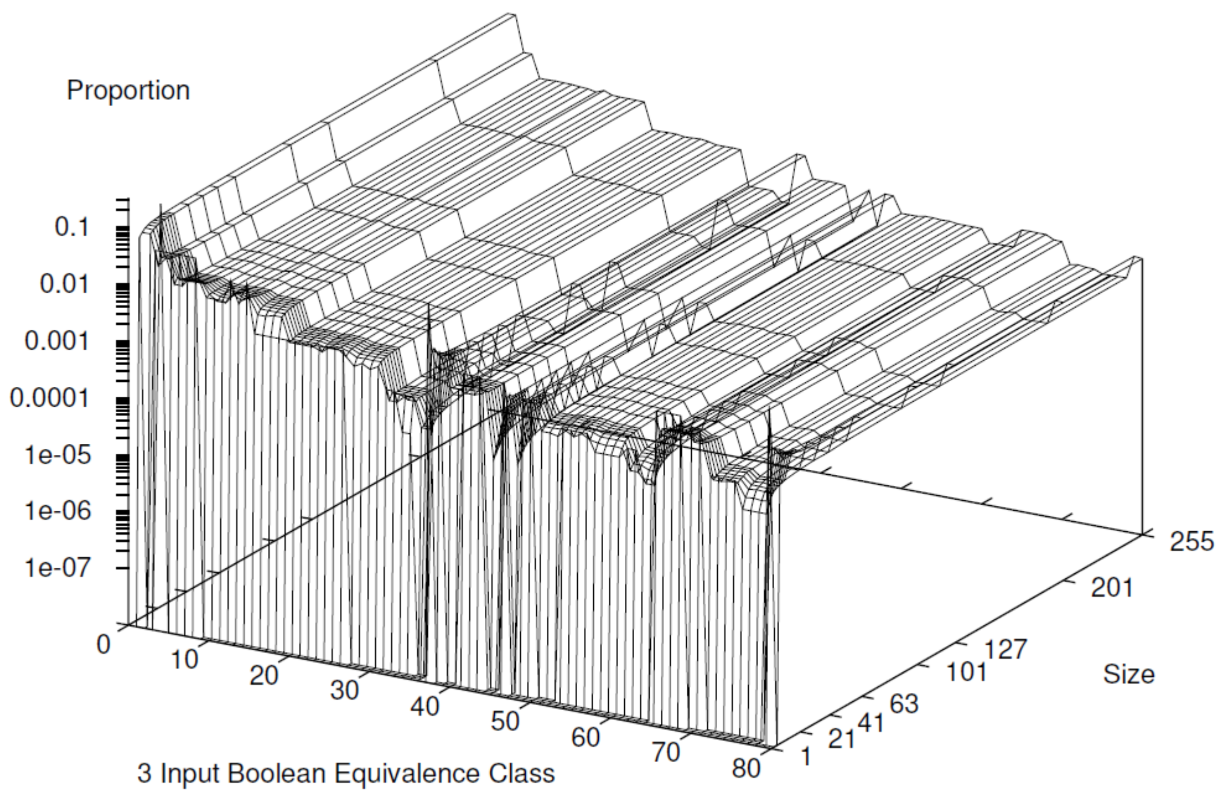


Figure 6: Proportion of functions in each equivalence class {AND, OR, NAND, NOR and XOR}

Space of Programs
{AND, OR, NAND, NOR}

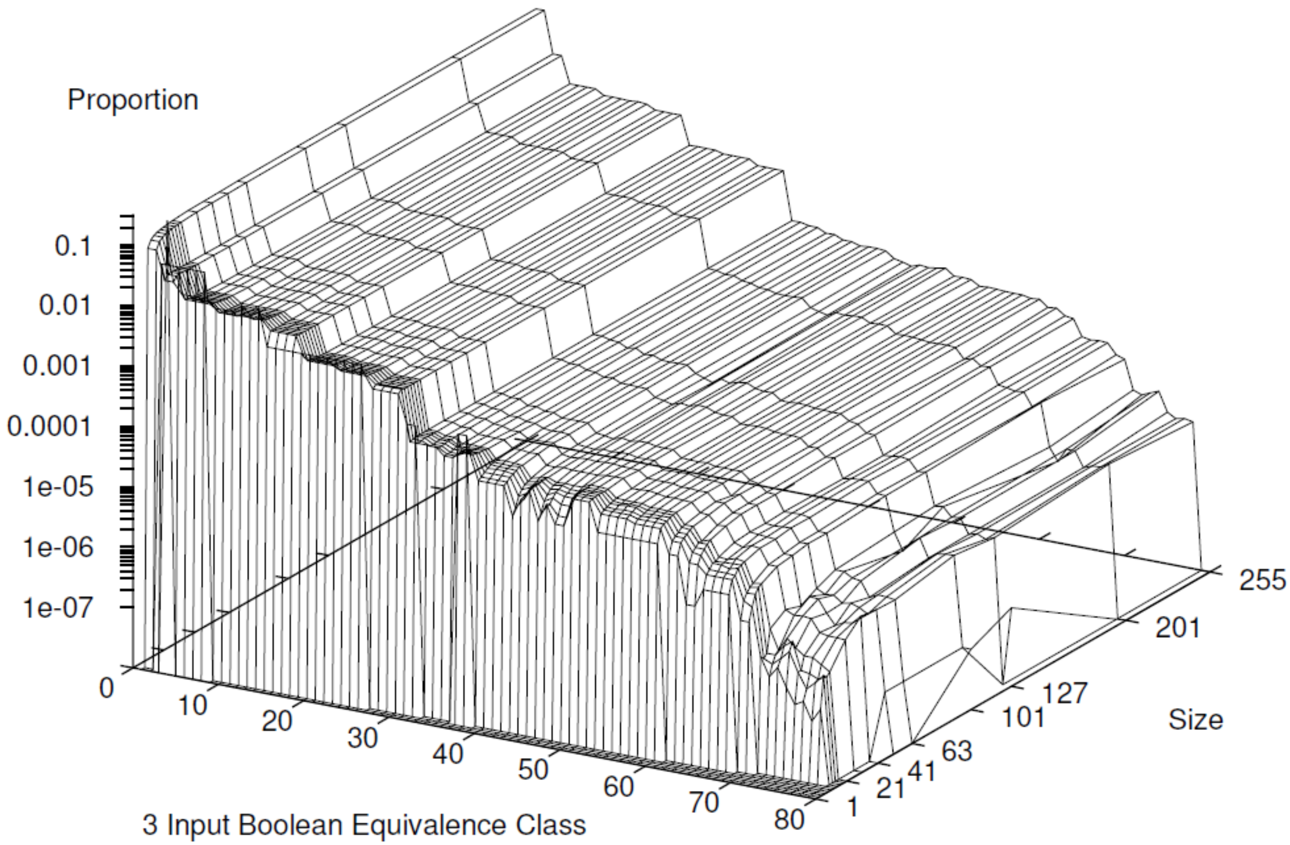
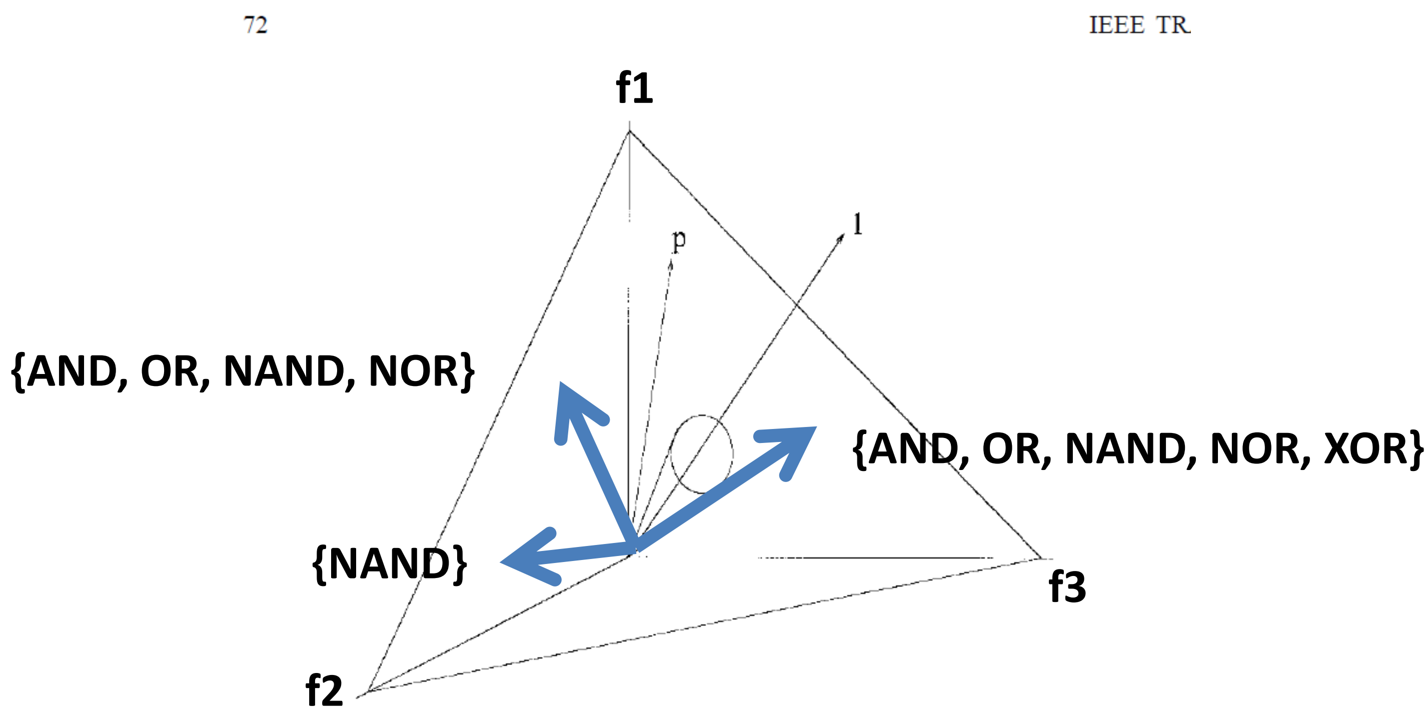


Figure 5: Proportion of functions in each equivalence class {AND, OR, NAND and NOR}

3 vectors of 3 problem classes



Three vectors from three different problem classes (method of generating problems).
What are the consequences of this?
We should qualify what problem classes our algorithms are suited for.

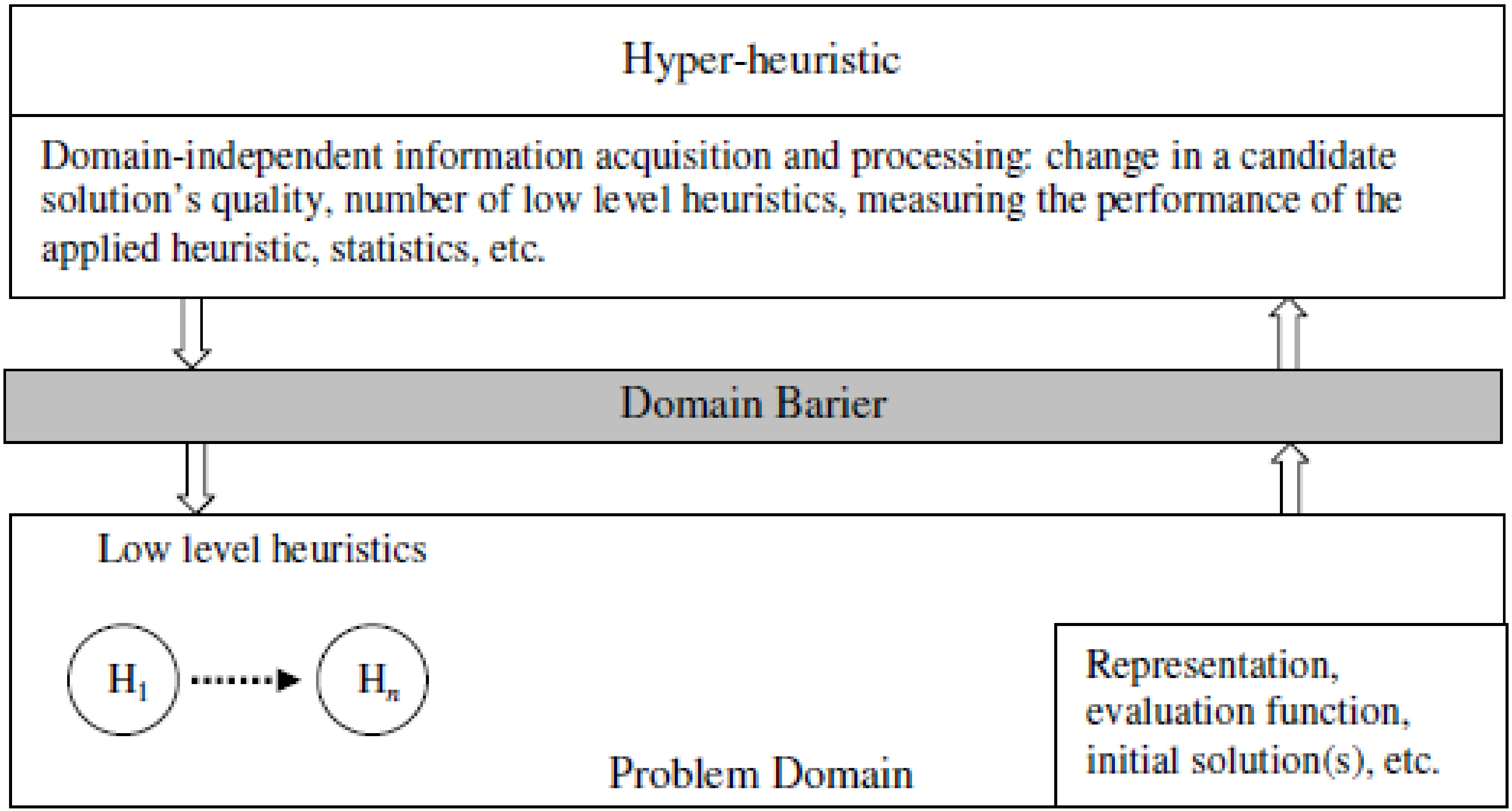
Machine Learning.

We cannot extrapolate/generalize from the training set to the test set (???).

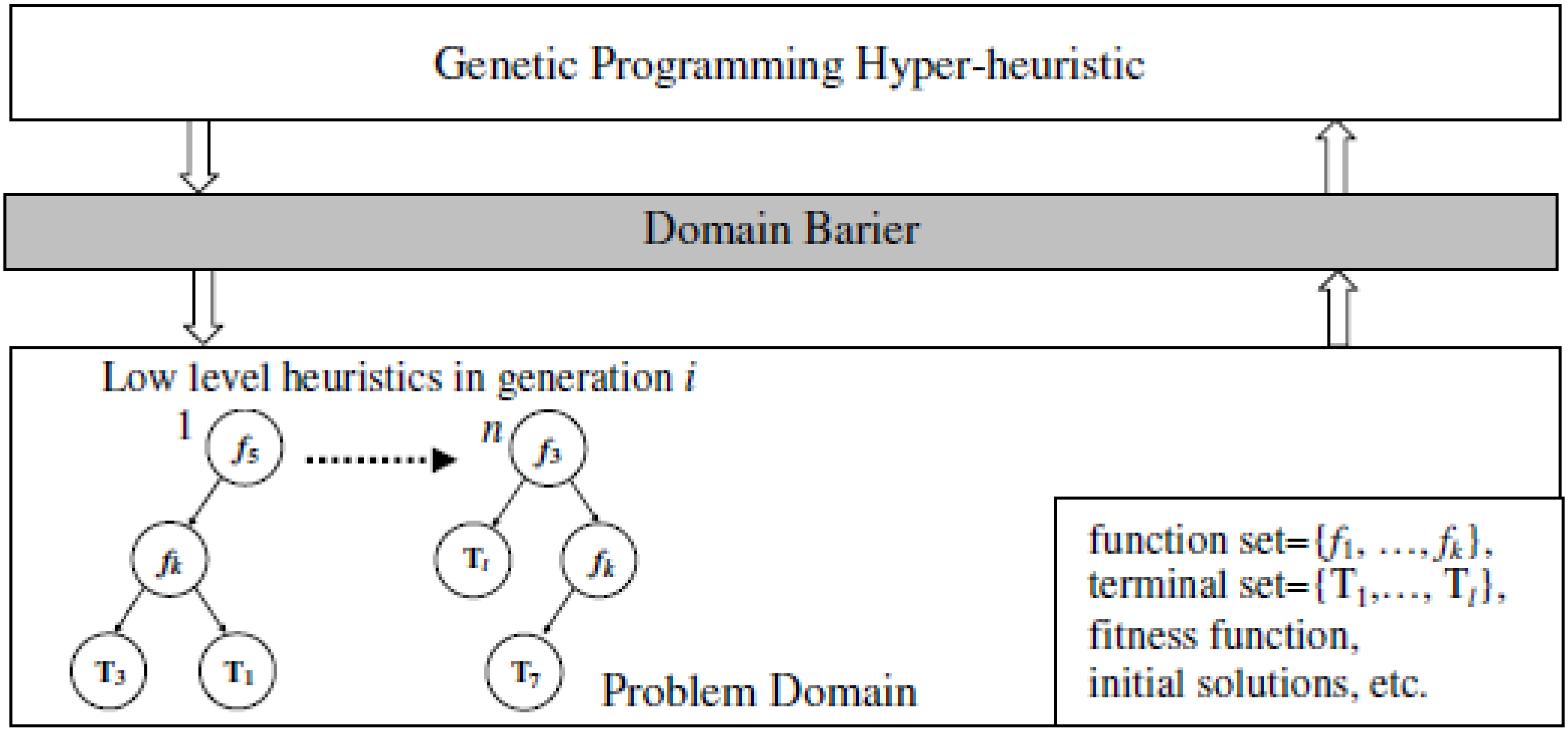
p(f)=**p(c|e)**, given example e, we want to predict which class c it belongs too. This is equivalent to known the distribution over the set of functions.

	Inputs			f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	...
Training Set	0	0	0	0	0	0	0	0	0	0	0	0	0	...
	0	0	1	0	0	0	0	0	0	0	0	0	0	...
	0	1	0	0	0	0	0	0	0	0	0	0	0	...
	0	1	1	0	0	0	0	0	0	0	0	0	0	...
	1	0	0	0	0	0	0	0	0	0	0	1	1	...
	1	0	1	0	0	0	0	1	1	1	1	0	0	...
Test Set	1	1	0	0	0	1	1	0	0	1	1	0	0	...
	1	1	1	0	1	0	1	0	1	0	1	0	1	...

Selective Hyper-heuristics (massaging problem state)



Generative Hyper-heuristics discovering novel heuristics



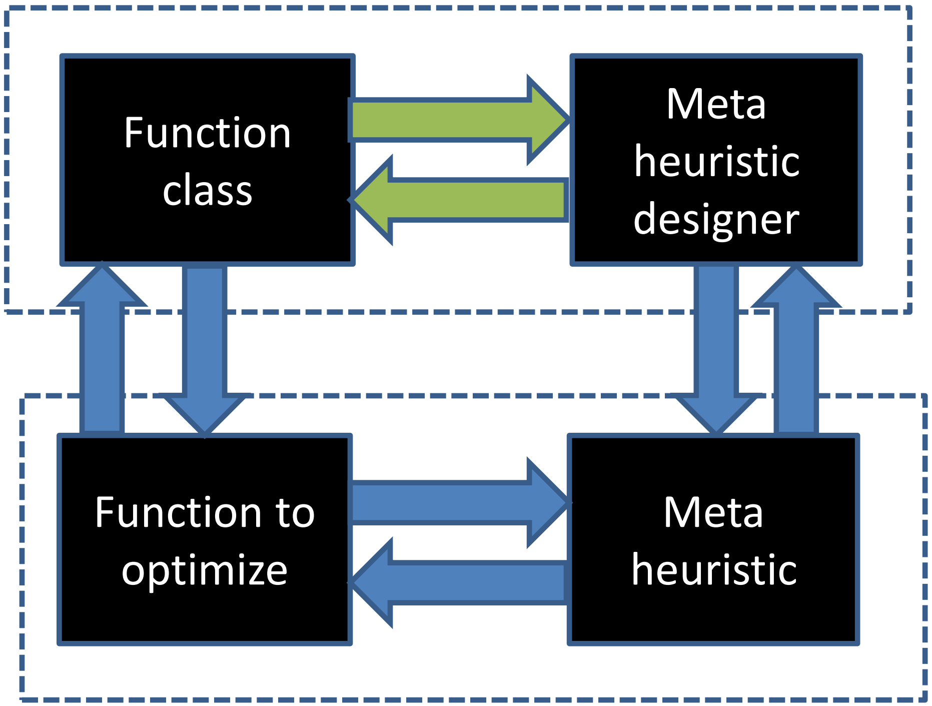
Generative Hyper-heuristics

- Instead of manually designing meta-heuristics, generate them **automatically in a generate-and-test loop**.
- This loop allows **feedback** between the automatic **meta-heuristic designer** and the **problem class**.
- Manual design is effectively taking place without any feedback from the environment in **isolation**.

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** Metaheuristic

Convergence at base/meta level

- A metaheuristic can converge on a global solution if there is a non-zero probability of reaching that point (cf. hill-climbing and simulated annealing).
- Easy to “repair” hill climbing so it converges
- Convergence at the hyper level means we can tune to any probability distribution(problem class). **$p(f)=p(c|e)$ i.e. histograms match,** or probability vectors point in same direction.
- Numerical parameters maybe limited in this respect.
- Do Hyperheuristics guarantee convergence?

Conclusions

- Automatic design avoids metaphors (and awkward terminology – use maths instead).
- Most meta-herusitcs cannot alter their bias over a run (it is fixed from one run to the next)
- Automatic design allows alignment of bias of metaheuristic with bias of problem class.
- Convergence at meta-level is concerned with aligning probability distributions (which may not be achieved with numerical parameters alone).

References

Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper?

Christophe Giraud-Carrier.

Metaheuristics—the metaphor exposed

Kenneth Sörensen

Unbiased Black Box Search Algorithms

Jonathan E. Rowe Michael D. Vose

Edgar A. Duéñez-Guzmán, Michael D. Vose: **No Free Lunch and Benchmarks.** Evolutionary Computation 21(2): 293-312 (2013)

My papers that explicitly mention problem classes…and

- Burke E. K., Hyde M., Kendall G., and Woodward J. **Automatic Heuristic Generation with Genetic Programming: Evolving a Jack-of-all-Trades or a Master of One** Proceedings of Genetic and Evolutionary Computation Conference 2007 London UK.
- Libin Hong and John Woodward and Jingpeng Li and Ender Ozcan. **Automated Design of Probability Distributions as Mutation Operators for Evolutionary Programming Using Genetic Programming**.
- John R. Woodward and Jerry Swan. **The automatic generation of mutation operators for genetic algorithms**.
- John Robert Woodward and Jerry Swan. **Automatically designing selection heuristics.**

The End

- Thank you for you attention.
- Any questions.
- 7 fully funded PhD positions at Stirling !!!
- http://www.cs.stir.ac.uk/~jrw/
- jrw@cs.stir.ac.uk
- Workshop at GECCO on automatic design of algorithms.