Theoretical Foundations of SBSE

Xin Yao
CERCIA, School of Computer Science
University of Birmingham
Some Theoretical Foundations of SBSE

Xin Yao and Many Others
CERCIA, School of Computer Science
University of Birmingham
Motivation for theoretical analysis of EAs

EAs have many attractive features

- ease of implementation
- applicable in a wide range of domains
- results often competitive with traditional techniques, but the understanding of how EAs really work is incomplete
- can be highly sensitive to choice of parameter settings
- experimental parameter tuning expensive
- in most cases, run EA and see what happens
- ...

...
Traditional Investigation of EAs

Run algorithm(s) on “real world” problem instance(s). Analyse results with some statistical methodology.

How representative are the results?
- Can we make any guarantee about performance?
- What happens on other instances?
- What happens for larger instance sizes?
- What happens for other parameter settings?

How can the results be explained?
- Why does/does not the algorithm work?
- Can the algorithm be improved?

⇒ Why not attempt the well established methodology that exists for analysing classical algorithms?
Evolutionary Algorithms are Algorithms

Criteria for evaluating algorithms

1. Correctness.
   - Does the algorithm always give the correct output?

2. Computational Complexity.
   - How much computational resources does the algorithm require to solve the problem?

Same criteria also applicable to search heuristics

1. Correctness.
   - Discover global optimum in finite time?

2. Computational Complexity.
   - Time (number of function evaluations) most relevant computational resource.
Worst Case Computational Complexity

“Real world” runtime: Runtime on “real world” instances
- Are these instances still relevant in 10 years? In 100 years?

Average case runtime: Runtime averaged over instances
- What is an average input (e.g. average FSM)?

Worst case runtime: Runtime on hardest instance
- Strong guarantee about performance of an algorithm.
- Lower bounds obtained by analysing runtime on specific hard problem instance.
Prediction of resources needed for a given instance. Usually *runtime* as function of *instance size*. Number of fitness evaluations before finding optimum.

- **Exponential** runtime $\implies$ Inefficient algorithm.
- **Polynomial** runtime $\implies$ “Efficient” algorithm.

Asymptotic notation hides “unimportant” details about runtime.
Search heuristics depend on **random inputs**

- Runtime differs between runs.

**Expected runtime**

- Runtime averaged over possible random inputs.

**Success probability**

- Probability of finishing within a specified time \( f(n) \).
Research Objectives and Strategy

Runtime analysis of search heuristics on software testing
  ▶ Understand behaviour of algorithm
  ▶ Runtime impact of operators and parameter settings
  ▶ Runtime impact of problem instance characteristics

Research strategy
  ▶ Start by analysing simple problems and algorithms
  ▶ Proceed with more complex scenarios
  ▶ Find appropriate mathematical techniques on the way
Conformance testing involves the *state verification problem*, which can be solved using unique input output (UIO) sequences.

**Definition**

A *unique input output sequence* for a state $s$ is a sequence $x$ st.

- $\forall t \neq s, \lambda(s, x) \neq \lambda(t, x)$,

where

- $\lambda(s, x)$ is output of FSM on input $x$, starting in state $s$.

**Example**

- 1 is a UIO for state $s_3$.
- 1 is not a UIO for state $s_1$. 

![Diagram of FSM with states and transitions]
Previous work

UIOs are fundamental in conformance testing of FSMs.
  - Used to solve the state verification problem.

Theoretical aspects
  - NP-hard to check whether a state has a UIO [Lee and Yannakakis, 1994].
  - Shortest UIOs can be exponentially long (empirical results suggest they are often short).

Experimental comparison between random search and GA [Guo et al., 2004] and [Derderian et al., 2006]
  - Min. length, max. number of different outputs.
  - Similar performance on small FSMs.
  - GA better than random search on larger FSMs, especially when long UIOs are needed
(1+1) Evolutionary Algorithm

(1+1) EA

Choose \( x \) uniformly from \( \{0, 1\}^n \).

Repeat

\[ x' := x. \]

Flip each bit of \( x' \) with probability \( 1/n \).

If \( f(x') \geq f(x) \),

then \( x := x' \).
Hard instance class - FSM Combination Lock

Theorem

On the instance class below

- The prob. that (1+1) EA (or RS) finds the UIO for state \(s_1\) within \(e^{c \cdot n}\) iterations is exponentially small.

Proof idea for (1+1) EA:

- All states “collapse” into \(s_1\) on input 0.
- Problem instance is a “needle in the haystack”.
- Success probability bounded by drift analysis.

[Lehre and Yao, 2007]
Theorem

On the instance class below,

- $(1+1)$ EA finds the UIO for $s_1$ in exp. time $O(n \log n)$.
- The prob. that random search finds a UIO for $s_1$ within $e^{c \cdot n}$ iterations is exponentially small $e^{-\Omega(n)}$.

Proof idea: The problem instance is essentially OneMax. [Lehre and Yao, 2007]
(1+1) EA? Are you kidding?

- What about populations?
- Well, large populations might not help.
Operator Interaction

• We are often concerned about which operators to use. In fact, interactions among operators can be essential. E.g.,

• Even parameter settings.
Insight into Problems

• Search algorithms can help us in gaining insight into a problem, e.g., we can use **EDAs (Estimation of Distribution Algorithms)** to find a near optimum while learning a model of the underlying problem --- a wonderful idea!

• However,
Future Work

Research Questions

- Relationships between problems and heuristics.
- Analysis of other meta-heuristics.
- Analysis of broader problem classes.
- Approximation quality of search heuristics.

Methodology

- Improve mathematical techniques.
More Practical Considerations

• Dynamic optimisation: The objective function may change; Fitness evaluation may be noisy; Variable values may be inaccurate

• Robust optimisation: The optimised solution is robust against minor perturbations of the decision variables

• ROOT (robust optimisation over time)
More Practical Considerations

• Scenario: Given a fixed time budget (say one day), what is the best solution you can generate using whatever algorithms at your disposal?

• Should I select one algorithm and allocate all the time to it? Should I divide the time budget among multiple algorithms? How to allocate the time resources?

More Practical Considerations

- Multi-objective formulation can sometimes solve a problem better, even by measuring a single objective only.