When Is a Meta-heuristic Approach Efficient in Search-Based Software Engineering

Xin Yao

CERCIA, University of Birmingham, UK

The 1st CREST Open Workshop / SEBASE Workshop
November 24th-25th, 2009

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?

$\Longrightarrow$ Why not attempt the well established methodology that exists for analysing classical algorithms?

SEBASE

Outline

Evolutionary Algorithms are Algorithms

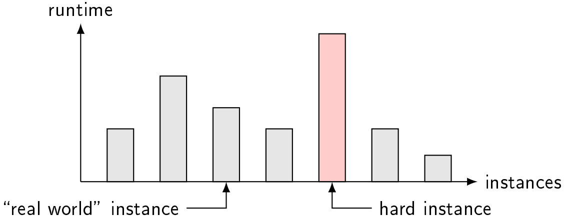
Criteria for evaluating algorithms

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?

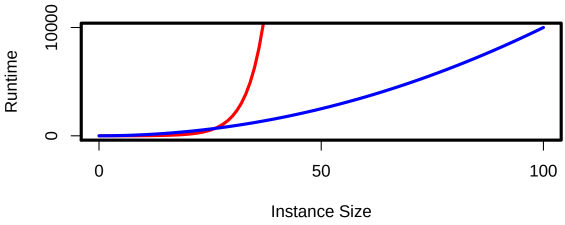
Average case runtime: Runtime averaged over instances

- 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.

Computational Complexity of Search Heuristics



Prediction of resources needed for a given instance.

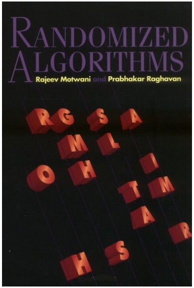
Usually *runtime* as function of *instance size*.

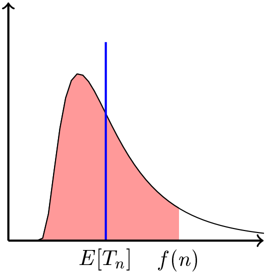
Number of fitness evaluations before finding optimum.

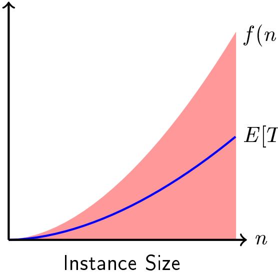
- Exponential runtime $\Longrightarrow$ Inefficient algorithm.
- Polynomial runtime $\Longrightarrow$ "Efficient" algorithm.

Asymptotic notation hides "unimportant" details about runtime.

Search Heuristic are Randomised Algorithms







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

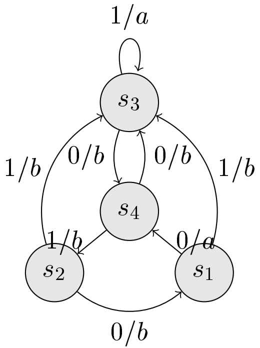
- Understand behaviour of algorithm
- Runtime impact of operators and parameter settings
- Runtime impact of problem instance characteristics

Research strategy

- Start by analysing simple problems and algorithms
- Proceed with more complex scenarios
- Find appropriate mathematical techniques on the way

Conformance testing and UIOs

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.

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

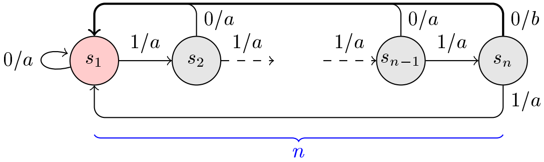
Choose x uniformly from $\{0,1\}^n$.
Repeat
  $x' := x$.
  Flip each bit of x' with probability $1/n$.
  If $f(x') \geq f(x)$,
    then $x := x'$.

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.

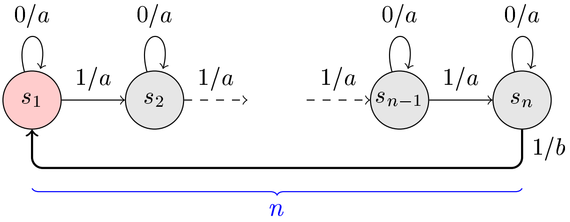
[Lehre and Yao, 2007]

Easy instance class - FSM Counter

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.

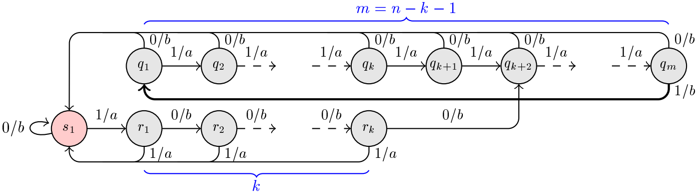
[Lehre and Yao, 2007]

Instances with tunable difficulty

Theorem

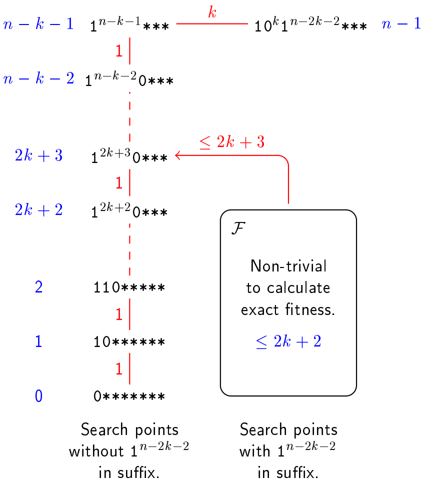
On the instance class below, with $k \geq 2$ any constant,

- *(1+1) EA finds an UIO for s_1 in expected time $\Theta(n^k)$.*



[Lehre and Yao, 2007]

Tunable Difficulty - Proof Idea.



$$\mathbf{Pr}[\mathcal{F}] = e^{-\Omega(n)}$$
$$\mathbf{E}\left[T \mid \overline{\mathcal{F}}\right] = \Theta(n^k)$$
$$\mathbf{E}[T \mid \mathcal{F}] = O(n^{2k+3})$$
$$\mathbf{E}\left[T \mid \overline{\mathcal{F}}\right] = \Omega(n^k)$$

$$\mathbf{E}[T] = (1 - \mathbf{Pr}[\mathcal{F}]) \cdot \mathbf{E}\left[T \mid \overline{\mathcal{F}}\right] + \mathbf{Pr}[\mathcal{F}] \cdot \mathbf{E}[T \mid \mathcal{F}] = \Theta(n^k).$$

Steady State GA with Crossover

(μ+1) SSGA

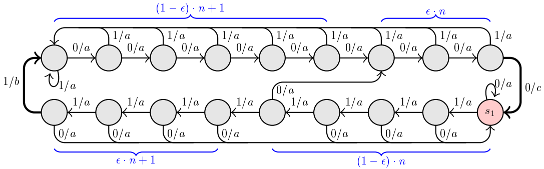
Sample a population P of μ points u.a.r. from $\{0,1\}^n$.
repeat
  with probability $p_c(n)$,
    Sample x and y u.a.r. from P.
    $(x', y') :=$ one point crossover(x, y).
    if $\max\{f(x'), f(y')\} \geq \max\{f(x), f(y)\}$
      then $x := x'$ and $y := y'$.
  otherwise
    Sample x u.a.r. from P.
    $x' :=$ Mutate(x).
    if $f(x') \geq f(x)$
      then $x := x'$.

Effect of Crossover

Theorem

On the instance class below,

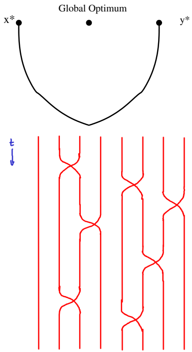
- *(μ+1) SSGA with constant crossover prob.* $p_c > 0$ *finds the UIO for state* s_1 *in* $c\mu^2 n^2$ *generations with probability* $1 - e^{-\Omega(n)} - e^{-\Omega(\mu)}$*.*
- *(μ+1) SSGA without crossover, i.e.* $p_c = 0$*, does not find the UIO for state* s_1 *in time* 2^{cn} *with probability* $1 - e^{-\Omega(n)}$*.*



[Lehre and Yao, 2008]

Proof Idea

$$\textsc{TwoPaths}\,(x) := \begin{cases} 2n & \text{if } x = 1^{(1-\epsilon)\cdot n}0^{\epsilon\cdot n}, \\ \textsc{Lo}(x) + \textsc{Lz}(x) & \text{otherwise.} \end{cases}$$



- Global optimum between two paths.
- Monotonic fitness along lineages.
- Lineages reach a local optimum in

$$O(n^2 \mu \log \mu / (1 - p_c)).$$

- Population divided evenly between paths
- Once on local optima, successful crossover

$$O(n/p_c).$$

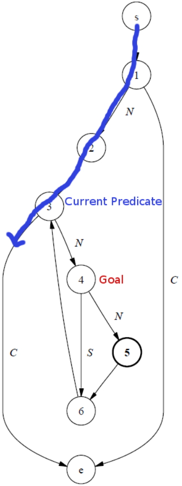
Branch Coverage of Triangle Classification

```
int tri_type(int x, int y, int z) {
  int type;
  int a=x, b=y, c=z;
  if (x > y) {
     int t = a; a = b; b = t;
  }
  if (a > z) { int t = a; a = c; c = t; }
  if (b > c) { int t = b; b = c; c = t; }
  if (a + b <= c) {
      type = NOT_A_TRIANGLE;
  } else {
      type = SCALENE;
      if (a == b && b == c) {
          type = EQUILATERAL;
      } else if (a == b || b == c) {
          type = ISOSCELES;
      }
    }
  return type;
}
```

[McMinn, 2004]

- Testing problem
  - Find x, y, z such that equilateral branch is covered.
- Fitness functions
  - approach level
  - branch distance
- Problem size
  - range of integer variables $x, y, z \in \{-N/2 + 1, ..., N/2\}$.

Fitness Functions (minimisation)



Approach level

- Minimal distance to branch in control flow graph.

Branch Distance

(Approach level, f(curr. predicate))

Predicate	f
`if (a>b)`	$b - a$
`if (a>=b)`	$b - a$
`if (a<b)`	$a - b$
`if (a<=b)`	$a - b$
`if (a==b)`	$\|b - a\|$
`if (a!=b)`	$-\|b - a\|$

McMinn (2004)

Expected runtimes on Equilateral Branch

Algorithms

- RS - Random Search
- HC - Hill Climber (local search)
- AVM - Alternating Variable Method
- (1+1) EA (with unsigned binary integer repr.)

Expected Runtimes

Algorithm	Approach level	Branch distance
RS	$\Theta(N^2)$	$\Theta(N^2)$
HC	$\Theta(N^2)$	$\Theta(N)$
AVM	$\Theta(N^2)$	$\Omega(\log N)$ and $O((\log N)^2)$
(1+1) EA[1]		$\Theta((\log N)^5)$

[Arcuri et al., 2008]

[1] Ongoing work.

Conclusion

Runtime of EAs on **UIO problem**

- (1+1) EA has exponential worst case runtime
- (1+1) EA still efficient on many instances, and outperforms a random search strategy.
- spectrum of increasingly hard instances for (1+1) EA.
- crossover and large population essential on certain instances.

Runtime on **branch coverage** of triangle classification

- AVM $\succ$ (1+1) EA $\succ$ HC $\succ$ RS.
- Theoretically confirmed well known results.

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.

Analysis of Other Meta-Heuristics

- Analysis of (1+1) EA necessary to develop techniques
- Lower bounds for population-based EAs
- Estimation of Distribution Algorithms (EDAs)
- Multi-objective EAs

Analysis of Broader Problem Classes

- Know specific instance classes that are easy and hard.
- Which conditions on the instance are sufficient to guarantee polynomial runtime?

Thank you for your attention!

Arcuri, A., Lehre, P. K., and Yao, X. (2008).
Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem.
In *In Proceedings of the 1st International Workshop on Search-Based Software Testing*.

Lehre, P. K. and Yao, X. (2008).
Crossover can be constructive when computing unique input output sequences.
In *Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL'2008)*.

Lehre, P. K. and Yao, X. (2007).
Runtime analysis of (1+1) EA on computing unique input output sequences.
In *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'07)*, pages 1882–1889.

References I

Arcuri, A., Lehre, P. K., and Yao, X. (2008).

Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem.

In *ICSTW '08: Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 161–169, Washington, DC, USA. IEEE Computer Society.

Derderian, K. A., Hierons, R. M., Harman, M., and Guo, Q. (2006).

Automated unique input output sequence generation for conformance testing of fsms.

The Computer Journal, 49(3):331–344.

Guo, Q., Hierons, R. M., Harman, M., and Derderian, K. A. (2004).

Computing unique input/output sequences using genetic algorithms.

In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES'2003)*, volume 2931 of *LNCS*, pages 164–177.

References II

Lee, D. and Yannakakis, M. (1994).

Testing finite-state machines: state identification and verification.

IEEE Transactions on Computers, 43(3):306–320.

Lehre, P. K. and Yao, X. (2007).

Runtime analysis of (1+1) EA on computing unique input output sequences.

In *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'2007)*, pages 1882–1889. IEEE Press.

Lehre, P. K. and Yao, X. (2008).

Crossover can be constructive when computing unique input output sequences.

In *Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL'2008)*, pages 595–604, Berlin, Heidelberg. Springer-Verlag.

References III

McMinn, P. (2004).
Search-based software test data generation: A survey.
Software Testing, Verification and Reliability, 14(2):105–156.

Oliveto, P. S., He, J., and Yao, X. (2008).
Analysis of population-based evolutionary algorithms for the vertex cover problem.
In *Proceedings of IEEE World Congress on Computational Intelligence (WCCI'2008), Hong Kong, June 1-6, 2008*, pages 1563–1570.