Some Theoretical Results in (Search-Based) Software Testing

Dr. Andrea Arcuri
arcuri@simula.no

Why Theory?

- Counterintuitive example of scalability
  - What works in empirical studies might not scale to larger instances
- Some general results
  - Valid for all instances, all sizes
- Explain *why*

In this talk, focus on the results, not on the math behind them...

A Scalability Example

Runtime Analysis

- Expected (average) number of fitness evaluations before global optimum
- Function of the problem size
    - eg, size of array in sorting algorithm

(Very) Simple Example

```
public void foo(int x1, int x2, int x3, ...){

    if(x1==k1  && x2==k2 && x3==k3 && ...)
       //TARGET
}
```

Runtime

- v integers as input
- Each integer is bounded in [0,n]
- For algorithm A, runtime will be $r(A,v,n)$

Considered Algorithms

- Random Search (RS)
- Two variants of Hill Climbing, with random restarts when local optima
- HCint: +-1 on integer values
- HCbit: bit flipping
- Fitness: branch distance

Example

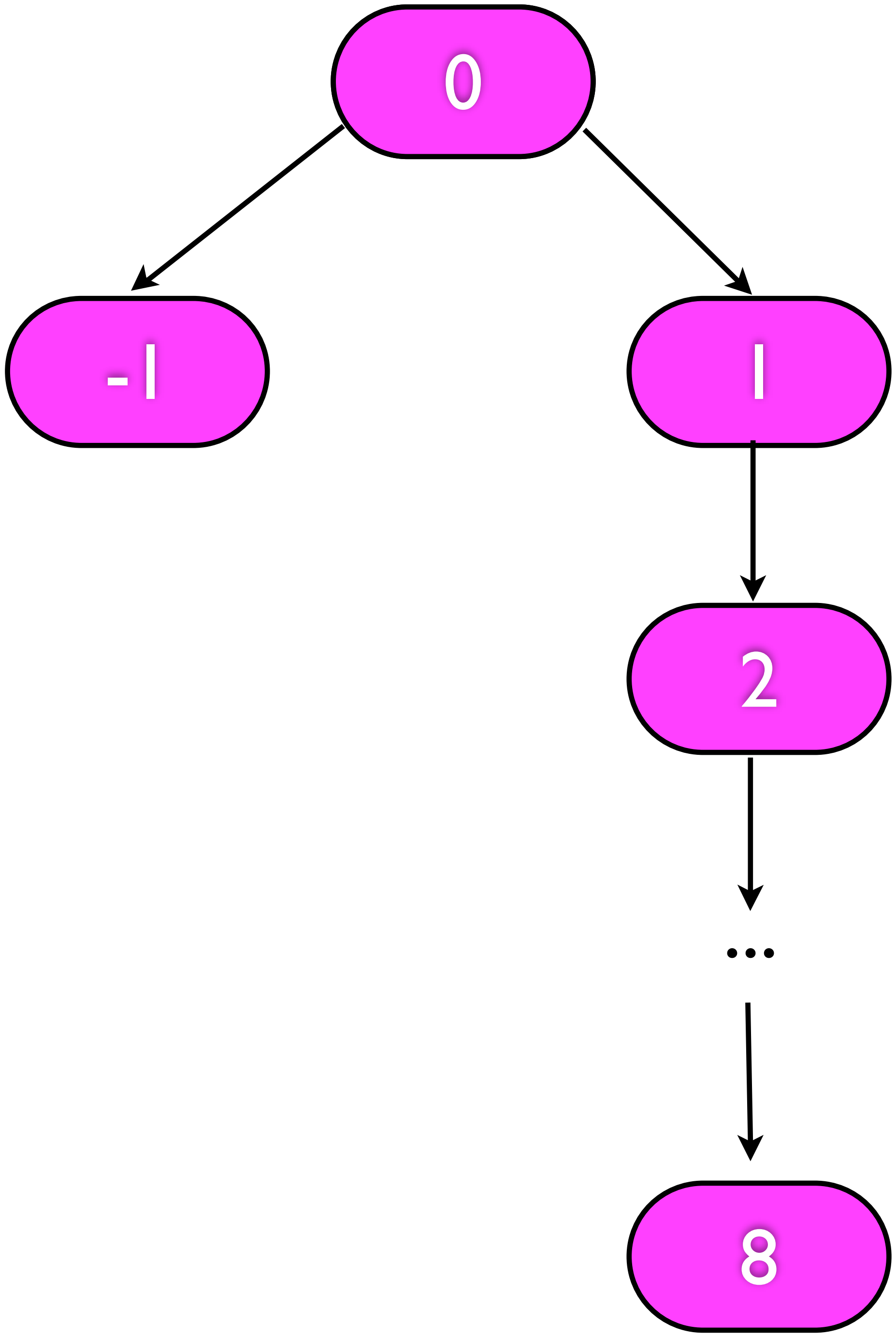
n==15 (range [0,15])

v==1 (only 1 input)

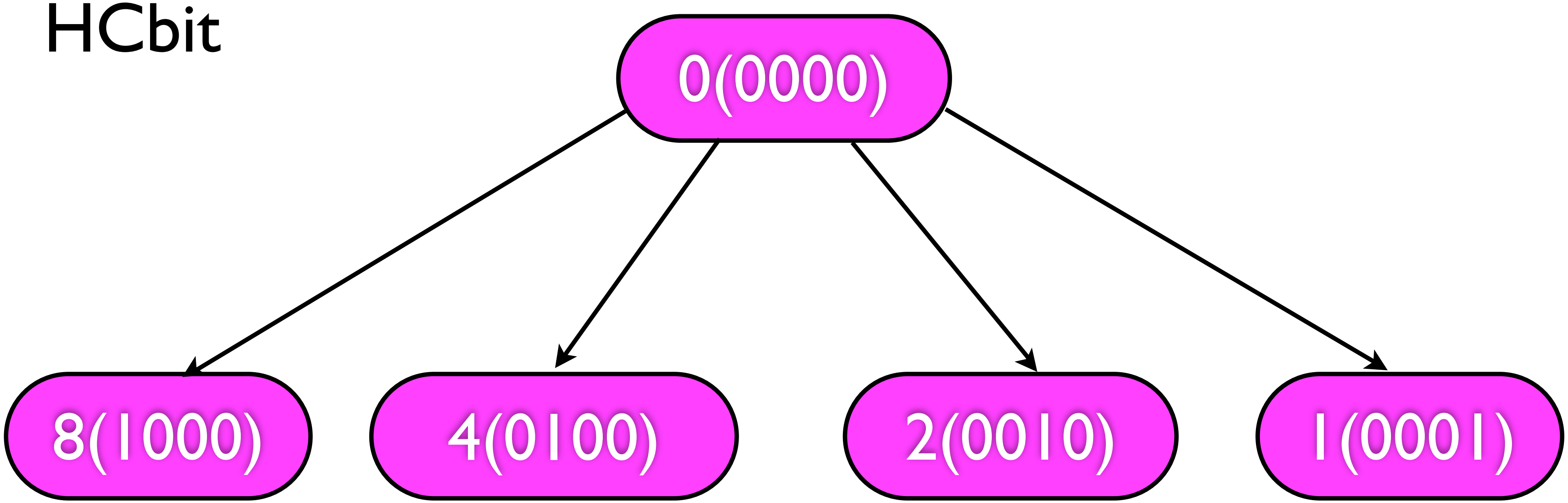
x==8 , binary(8)=1000

Let the (random) starting point be 0

HCint



HCbit

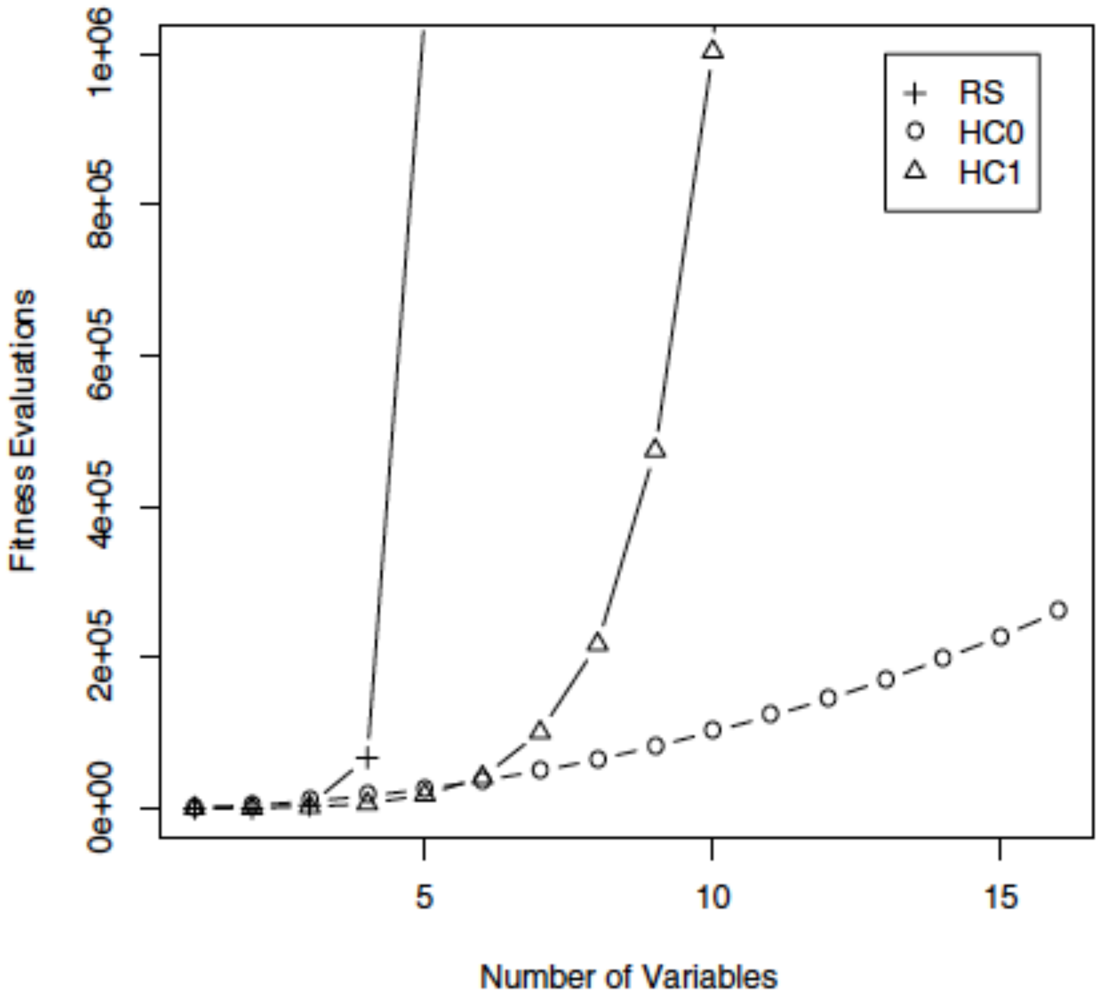


Runtime when v==1

- r(RS,1,n) = $\Theta(n)$
- r(HCint,1,n) = $\Theta(n)$
- r(HCbit,1,n) = $\Theta(\log(n)^2)$
- *HCint same asymptotic runtime as RS*

But for fixed *n* and large *v*...

- r(RS,v,n=c) = $e^{\Theta(v)}$
- r(HCint,v,n=c) = $\Theta(v^2)$
- r(HCbit,1,n=c) = $\Theta(v^2 b^v)$
- RS and HCbit have *exponential* runtime



For small *v*, HCint is not better than RS, and HCbit is faster

But HCint is the only one that *scales*, ie no exponential runtime

What's going on???

- Bit representation is faster, but has *local optima*
  - binary(7)=0111 is local optimum for target binary(8)=1000
- When local optimum, HC needs restart
- Number of restarts exponential in *v*
  - so negligible for low values of *v*

Some General Results

Combinatorial Testing (CT)

- *k* variables having *v* possible values
- *t* is the target combination strength to cover
    - eg t=2 consider all possible pairs
- *N*: test cases covering all *t*-wise combinations
- Goal: minimize *N* while covering all *t* combinations
- Assume no constraints among variables
    - all combinations are valid

CT Tools

- Many CT tools exist
- (Usually) scalability problems
  - eg when *k* is in the order of hundreds/thousands
  - can take hours, days, or simply crash...
- Why not generating the *N* test cases at *random???*

Is Random really bad?

- Random found more bugs than CT
- J. Bach and P. Schroeder. **Pairwise testing: A best practice that isn't**. In Proceedings of 22nd Pacific Northwest Software Quality Conference, pages 180–196, 2004.

Theoretical Analysis

- For **any** *k*, *v* and *t*, given the same number *N* as CT, Random has *at least* 63% of finding *t*-wise bug
- Probability goes to 100% for increasing *k*
- Example: *v*=4 and *k*=100 probability is *at least* 94%!!!

Implications of Theory

- Confirms and explains Bach&Schroeder's empirical results.
- For large *k*, Random as effective as CT for *t*
- If automated oracle: can generate/run more than *N* given same time
- More tests: higher fault detection for greater *t*

Sorry... going to show some theory... how to prove a 63% lower bound?

First, calculate probability p_f of random test finding a failure (if any exist)

$$p_f \geq \frac{1}{\prod_{i \in F_f} v_i} \geq \frac{1}{\prod_{i=1}^{t} v_i}.$$

Which is one over all possible combinations of *t* variables

Example: t=2 and v=4, p_f is at least1/4*4 = 1/16

Second, calculate probability of at least one test case fails out of N

$$P_t = 1 - (1 - p_f)^N \geq 1 - \left(1 - \frac{1}{\prod_{i=1}^{t} v_i}\right)^N .$$

Which is 1 minus probability of none failing. Probability of none failing is probability of pass (1-p_f) repeated N times

Now, thanks to the following inequality, we conclude the proof. Note: trivial lower bound for N is $\prod_{i=1}^{t} v_i$ ie, all possible combinations of *t* variables

$$\left(1 + \frac{w}{x}\right)^x < e^w \, .$$

$$\begin{aligned} P_t &\geq 1 - \left(1 - \frac{1}{\prod_{i=1}^{t} v_i}\right)^N \\ &\geq 1 - \left(1 + \frac{-1}{\prod_{i=1}^{t} v_i}\right)^{\prod_{i=1}^{t} v_i} \\ &\geq 1 - e^{-1} > 0.63 \, . \end{aligned}$$

References

- Andrea Arcuri. **Theoretical Analysis of Local Search in Software Testing**. In Symposium on Stochastic Algorithms, Foundations and Applications (SAGA), pp. 156-168, Japan, 2009.
- Andrea Arcuri and Lionel Briand. **Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing**. IEEE Transactions on Software Engineering, vol. 38, issue 5, pp. 1088-1099, 2012.

Conclusion

- Theory can tell you **why** things happen in a particular way
- Can answer scalability questions
- Just another tool to address research questions
  - Theory and empirical studies should go together hand-in-hand
- Might be difficult to carry out on real-world problems, if possible at all
  - see Theory track at GECCO