Random Testing with Theoretical Guarantees

Rupak Majumdar

Max Planck Institute for Software Systems (MPI-SWS), Germany
Despite Many Formal Approaches…

... practitioners test their code

... by providing random inputs

And despite our best judgment, ...

... testing is *surprisingly effective in finding bugs*

In this talk, we explore this unexpected effectiveness
Example: Jepsen

http://jepsen.io

A framework for black-box testing of distributed systems by randomly inserting network partition faults

CAP Theorem: No system has consistency, availability, and partition tolerance

etcd
Postgres
Redis
Riak
MongoDB
Cassandra
Chronos
Kafka
RabbitMQ
Consul
Elasticsearch
Aerospike
Zookeeper
Example: HitMC
https://gitlab.mpi-sws.org/rupak/hitmc

Black-box testing of message ordering bugs in distributed systems
Given the right *notion of depth*, every bug is *shallow*

Combinatorial structures

1. Set partitions
2. Partial orders

Algorithmic bounds on families of structures
Tests and Coverage

Coverage Goals

Tests

Covering Family of Tests = Set of tests covering all goals
“Small” covering family = More efficient testing
Random Testing

Pick tests at random, independently

Coverage Goals

Tests

Question: Will random testing find a small covering family?
Random Testing

Pick tests at random, independently

Suppose \( \Pr[\text{Test covers goal}] \geq p \)

Can we characterize covering families w.r.t. \( p \) and \( |G| \)?
Probabilistic Method

Let $G$ be the set of goals and $\Pr[\text{Random test covers a goal}] \geq p$

**Theorem.** *There exists a covering family of size $(1/p) \log |G|$.***

**Proof.**

$\Pr[\text{Random test does not cover goal } g] \leq 1 - p$

$\Pr[\text{K ind tests do not cover goal } g] \leq (1 - p)^K$

$\Pr[\text{K ind tests is not a covering family}] \leq |G| (1 - p)^K$

For $K = (1/p)\log |G|$, this probability is $< 1$

Then: *There must exist K tests that is a covering family!*

Existence, not constructive!
Probabilistic Method

Let $G$ be the set of goals and $\Pr[\text{Random test covers a goal}] \geq p$

Theorem. *There exists a covering family of size $(1/p)\log|G|$.*

*By repeated sampling, random testing overwhelmingly likely to find a covering family!*
Tests and Coverage

1. What are the coverage goals?
2. What are tests? What is the Probability space?
3. Can we bound $\Pr[\text{Test covers a goal}]$?
Plan

- Start with combinatorial puzzles
- Connect these puzzles to testing
  1. Come up with coverage goals
  2. Show how the puzzles relate to coverage goals
  3. Bound the probabilities
Part I: Partitions
Ninjas in Training

Round 1:

Round 2:

Training is **complete** if for every pair of ninjas, there is a round where they are in opposing teams.
Ninjas in Training

Training is **complete** if for every pair of ninjas there is a round where they are in opposing teams.

How many rounds make the training complete?

Naïve: $O(n^2)$ rounds

Can you do it in $\log n$ rounds?
Ninjas in Training

Now \( n \) ninjas are practicing in \( k \)-way fights:

Round 1:

1
2
3
... 
\( n \)

Round 2:
Ninjas in Training

Training is complete if for every choice of $k$ ninjas there is a round where they are each in a different team.

How many rounds make the training complete?
Ninjas in Training

Example:

Round 1:

Round 2:
Ninjas in Training

Example:

Round 1:

Round 2:
Ninjas in Training

Example:

Round 1:

Round 2:
Ninjas in Training

Example:

Round 1: 1 2 3 ... n

Round 2: ...
Ninjas in Training

Training is **complete** if for every choice of \( k \) ninjas there is a round where they are each in a different team.

How many rounds make the training complete?

Naïve: \( O(n^k) \)

Better: \( O(\exp(k) \log n) \) rounds

Exponentially better in \( n \), when \( k \) is constant.
From Training Ninjas to Distributed Systems with Partition Faults

ninjas
weapons
rounds
complete training

nodes in a network
blocks in a partition
tests = partitions

splitting family of partitions
Example: etcd
Example: etcd
Example: etcd
Example: etcd
Coverage Goal: Splitting Families

Given \( n \) nodes and \( k \leq n \),

a partition of nodes \( \mathcal{P} = \{B_1, \ldots, B_k\} \) **splits** a set of nodes \( \mathcal{S} = \{x_1, \ldots, x_k\} \) if \( x_1 \in B_1, \ldots, x_k \in B_k \).

A set of partitions \( \mathcal{F} \) is a **k-splitting family** if for every \( k \)-subset of nodes there is a partition in \( \mathcal{F} \) that splits it.
Why k-Splitting?

Many Jepsen bugs are explained by k-splitting

Chronos: A distributed fault-tolerant job scheduler

- Works in conjunction with Mesos and Zookeeper
- Three special nodes: Chronos leader, Mesos leader, Zookeeper leader

From Jepsen
https://jepsen.io
A partition isolating Chronos from the ZK leader can cause a crash #513

aphyr opened this issue on 7 Aug 2015 · 20 comments

aphyr commented on 7 Aug 2015

When a network partition isolates a Chronos node from the Zookeeper leader, the Chronos process may exit entirely, resulting in downtime until an operator intervenes to restart it.

A partition isolating Chronos from the ZK leader can *not* cause a crash #522

aphyr opened this issue on 14 Aug 2015 · 7 comments

aphyr commented on 14 Aug 2015

Per #513, Chronos is expected to crash when a leader loses its Zookeeper connection. In this test case, Chronos detects the loss of its Zookeeper connection and, instead of crashing, sleeps quietly and reconnects when the partition heals. #513 argues that to keep running would violate unspecified correctness constraints. To preserve safety, should Chronos also crash here?

air commented on 15 Aug 2015

Hi - you're referring to a statement that doesn't represent the design (it wasn't expressed carefully enough). Please disregard it and refer to the clarification in the thread. Make sense?
Tests and Coverage

Coverage Goals

1. k-Splitting

2. Randomly sampled partitions

Tests

3. Can we bound $\Pr[\text{Test covers a goal}]$?
Small k-Splitting Families

• Fix a k-element set S from an n-element universe U

• What is the probability a random partition splits S?

• A splitting partition uniquely corresponds to a map \( U \setminus S \rightarrow S \). There are \( k^{(n-k)} \) such maps

• So probability = \( k^{n-k} \left\{ \begin{array}{c} n \\ k \end{array} \right\}^{-1} \)

Stirling number of 2\(^{nd}\) kind = Number of partitions of n elements into k parts
Small k-Splitting Families

- Probability that a fixed set is split = \( k^{n-k} \binom{n}{k}^{-1} \)
- Can we get rid of n?
- Yes: \( k^n \geq k! \binom{n}{k} \geq k^{-k} \cdot k! \)

All functions from n to k

All surjections from n to k
Small k-Splitting Families

- \( \Pr[\text{Random partition splits } S] \geq \frac{k!}{k^k} \)

- From our general theorem: There is a k-splitting family of size \( \left( \frac{k^k}{k!} \right) k \log n \)

- Turns out: uniformly sampling k-partitions is hard
  
  - But sampling balanced partitions is sufficient

\[
k^n \left( \binom{n}{k} \right) \leq n^k \left\{ \binom{n}{k} \right\}
\]
Partitions and Jepsen

Coverage Goals

1. k-splitting families

2. (Balanced) partitions

Tests

3. $\Pr[\text{Test covers a goal}]$ grows as a function of $k$

Hence, splitting family of size $\exp(k) \log n$
Part II: Order
Hungry Ninjas

After training, the ninjas go to a banquet

A banquet is complete if for every pair of ninjas \((i, j)\), there’s a course that is served to \(i\) before \(j\) and one that is served to \(j\) before \(i\).

How many courses make a banquet complete?
A Complete Banquet

Two courses suffice:

1 2 \ldots n

n n-1 \ldots 1
A banquet is **3-complete** if for every *triple* of ninjas \((i, j, k)\), there’s a course served in the order \(i < j < k\).

How many courses make a banquet 3-complete?

There is a 3-complete banquet with \(O(n^3)\) courses.
A banquet is \textbf{d-complete} if for every \textbf{d-tuple} of ninjas \((i, j, \ldots, k)\), there’s a course served in the order \(i < j < \ldots < k\).

How many courses make a banquet d-complete?

There is a d-complete banquet with \(O(n^d)\) courses … but one can do exponentially better.
Masters at the Banquet

Ninjas, of course, form a hierarchy
A master is always served before their student
Masters at the Banquet

Again, two courses suffice for 2-completeness:

```
1 2 4 5 3 6 7
```

```
1 3 7 6 2 5 4
```

ldfs

rdfs
Ninjas at the Banquet

A banquet is **3-complete** if for every triplet \((i, j, k)\), there’s a course served to ninja \(i\) before \(j\), and \(j\) before \(k\).

Naive approach with \(n^3\) courses:

Pick a course for each \(\binom{n}{3} \cdot 3!\) orders.

Can you do it with \(O(\log n)\) courses?
From Eating Ninjas to Testing Distributed Systems

ninjas hierarchy
courses
d-complete banquet

events
partial order on events
tests = schedules/linearization
d-hitting family of schedules
Coverage Goal: Hitting Families

Given a poset of events, a schedule hits a d-tuple of events \((e_1, \ldots, e_d)\) if it executes the events in the order \(e_1 < \ldots < e_d\).

Given a poset of events, a family of schedules \(F\) is d-hitting if for every admissible d-tuple of events there is a schedule in \(F\) that hits it.
Why d-Hitting?

Many bugs in asynchronous programs involve small number of events—bug depth $d$

[Lu et al. ASPLOS '08] [Burckhardt et al. ASPLOS '10] [Jensen et al. OOPSLA '15] [Petrov et al. 2012]

$d = 2$: order violation

$d = 3$: atomicity violation

Cassandra’s Paxos bug (img. from Leesatapornwongsa et. al. ASPLOS'16)
Finding d-Hitting Families

**Antichains:** For an antichain of size $n$ and for each $d$, there is a $d$-hitting family of size $O(\exp(d) \log n)$.

Balanced binary trees, Series parallel graphs:

For any $d$, there is a hitting family of size $\exp(d) \cdot (\log n)^{d-1}$

Open: What is the optimal size of a hitting family?
Order dimension of a poset: Number of linearizations whose intersection is the poset [Dushnik & Miller 1941]

- Order dimension = size of the smallest 1-hitting family
- Antichains, trees: Dimension = 2
- [Yannakakis] Computing the order dimension is NP-complete in general (even for dimension 3)
What if the Poset is not known?

**Online upgrowing poset**: Elements of the poset are exposed one at a time, online. However, each element is maximal at the time it is presented.

Corresponds to events in a running distributed system.

**Online dimension**: Find the smallest set of linearizations online whose intersection is the poset.
A Dist. System Example

Handler → Logger → Terminator

Request → Log → Terminate

Flush → Flushed

Buggy if Flush executes before Log!
Online Dimension

- Game between Program and Scheduler
- Program presents a poset one element at a time
- Each element is maximal when it is presented
- Scheduler maintains a set of linear extensions whose intersection is the poset
- Online dimension: Best strategy of the Scheduler
- (May be parameterized by the width of the poset)
Adaptive Chain Covering

Adaptive chain covering: Given upgrowing poset, maintain a set of chains covering the poset

\[ \text{adaptive}(w) : \# \text{ of chains used by Scheduler on a poset of width } w \]

Theorem [Felsner1997, Koch2007]

For a poset of width at most \( w \):

\[ \text{adaptive}(w) = \text{online-dimension}(w) = 1\text{-hit}(w) \]
Online d-Hitting Families

Theorem [OzkanNiksicM.TabaeiWeissenbacher18]
For an online upgrowing poset of size $n$ and width $w$:

\[ d\text{-hit}(n, w) \leq \text{adapt}(w)(d - 1)! \binom{n}{d} \]

\[ d\text{-hit}(n, w) \leq \left( \frac{w + 1}{2} \right) (d - 1)! \binom{n}{d} \]
A Dist. System Example

Handler  Logger  Terminator

Request

Log

Terminate

Flush

Flushed

Upgrowing Poset:

Request

Log

Terminate

Flush

Flushed

The bug is detected with probability:

PCTCP: 1/2
Random walk: 1/4
HitMC: Probabilistic Concurrency Testing with Chain Partitioning

- Randomized testing algorithm to sample d-hitting schedules based on adaptive chain coverings

- Ensures the probability of hitting a d-tuple is at least

\[
\frac{1}{w^2 n^{d-1}}
\]

for an online upgrowing poset of n elements and (unknown) width w

[OzkanNiksicM.TabaeiWeissenbacherOOPSLA18]
The HitMC Algorithm

- Maintain an **online chain partition** of the poset
- Assign a random priority to each chain
- Always execute enabled events from the highest priority chain
- At d-1 random points in the execution, decrease the priority of a chain
Online Chain Partitioning

- If a poset has width $w$, it can be partitioned into $w$ chains [Dilworth’s Theorem]

But if the poset is presented incrementally, we may not be able to achieve a partition into $w$ chains

- [Agarwal and Garg] We can maintain online chain partitioning with at most $w(w+1)/2$ chains
Online Chain Partitioning

Input: Poset of unknown width $w$, presented in upgrowing manner

Keep sets of chains $B_1, \ldots, B_w$

Invariant: $|B_i| \leq i$ and the last elements of all chains in $B_i$ form an antichain

Given next element $y$:

- find least $i$ such that $y$ is greater than some element at the end of a chain $a$ in $B_i$ or $|B_i| < i$
- add $y$ to the end of $a$ (or, if $|B_i| < i$, make a new chain whose only element is $y$)
- Swap: $B_{i-1} = B_i \setminus \{a.y\}$ and $B_i = B_{i-1} \cup a.y$
A Dist. System Example

Request → Log → Terminate → Flush → Flushed

Buggy if Flush executes before Log!

Upgrowing Poset:
- Request
- Log
- Terminate
- Flush
- Flushed

Online CP:
- [Request, log]
- [Terminate, Flush, Flushed]
Part III: Fuzzing
Fuzzing

• A very successful way to find bugs in sequential code

• Many high-profile industrial projects: AFL, OSS-Fuzz, ...

  • Many systems and open source projects have default fuzzing for testing

• Essentially: random testing + local search
Basic Fuzzing Algorithm

T = initial random set of inputs

Repeat

\[ t = \text{pick}(T) \]

\[ t' = \text{mutate}(t) \]

run the program on \( t' \)

if the program crashes: add \( t' \) to CrashList

if \( t' \) is interesting, add \( t' \) to \( T \)

Until enough bugs or resource exhausted
Why does Fuzzing work so well?

- Short answer: We don’t have a theoretical explanation!
- Slightly longer answer: We are using local search to navigate a state space and local search heuristically performs well
- Even though theoretical results (rapid mixing of Markov chains) are mostly negative
- Careful: We don’t know if RP = NP
A Simple Observation

- Sometimes local search is provably good

- For 2SAT, a random local search solves satisfiability in polynomial time

- For k-SAT, a local search can solve satisfiability in $O(1.34^N)$ time, asymptotically better than simple enumeration
1. What are the coverage goals?
2. What are tests?
3. Can we bound \( \Pr[\text{Test covers a goal}] \)?
   ⇒ If so, “small” test sets exist
A Remark on Sampling

- Sampling from an underlying combinatorial space can be a hard problem.

- In fact, one can show the ability to sample \( \sim \) the ability to count.

- Thus, if you want to sample from the solutions of SAT, you have to solve \#SAT.

- Some interesting recent work on this.
Summary

1. What are the coverage goals?

2. What are tests?

3. Can we bound $\Pr[\text{Test covers a goal}]$?  
   $\Rightarrow$ If so, “small” test sets exist
Partitions

1. Splitting nodes in network partitions

2. Network partitions

3. Size of splitting family $\leq \exp(k) \log n$
Order

Coverage Goals

1. Hitting all d-tuples a.k.a. (online) d-dimension

Tests

2. Schedules

3. Size of hitting family for tree $\leq \exp(d) \log n$

For upgrowing orders $\leq \text{adapt}(w).n^{d-1} (d-1)!$
1. What are the coverage goals?

2. What are tests?

3. Can we bound \( \Pr[\text{Test covers a goal}] \)?
   \( \Rightarrow \) If so, “small” test sets exist
Summary

• Given the right notion of depth, every bug is shallow.

Combinatorial structures

1. Partitions
   - k-splitting families of partitions

Algorithmic bounds on the structures

2. Orders
   - Generalizations of order dimension
     - for trees: explicit
     - for upgrowing p.o.: randomized
Thank You

Joint work with Dmitry Chistikov, Burcu Ozkan, Filip Niksic Mitra Tabaei, Georg Weissenbacher

See Filip Niksic’s PhD Thesis!