Testing RESTful Web Services with EvoMaster

Prof. Andrea Arcuri

Kristiania University College, Oslo, Norway

In this talk

1. REST web services
2. Search Algorithms
3. *EvoMaster* tool
4. Demo

Web Services

- Providing APIs (Application Programming Interfaces) over network, remote servers
- Communications over UDP/TCP, with protocols like HTTP
- Different types of data transfer formats
  - JSON, XML, HTML, plain text, etc.
- Permanent storage:
  - eg, SQL/NoSQL databases

RESTful APIs

- Most common type of web services
  - others are *SOAP* and *GraphQL*
- Access of set of resources using HTTP
- REST is not a protocol, but just architectural guidelines on how to define HTTP endpoints
  - hierarchical URLs to represent resources
  - HTTP verbs (GET, POST, PUT, DELETE, etc.) as “actions” on resources

Example for a Product Catalog

- Full URLs, eg consider an hypothetical **www.foo.com/products**
- GET **/products**
  - (return all available products)
- GET **/products?k=v**
  - (return all available products filtered by some custom parameters)
- POST **/products**
  - (create a new product, with data in a JSON payload)
- GET **/products/{id}**
  - (return the product with the given id)
- GET **/products/{id}/price**
  - (return the price of a specific product with a given id)
- DELETE **/products/{id}**
  - (delete the product with the given id)

Resource Summary
- Files
- About
- Changes
- Children
- Parents
- Permissions
- Revisions
- Apps
- Comments
- Replies
- Properties
- Channels
- Realtime
- Teamdrives
- Standard Features

API Reference

This API reference is organized by resource type. Each resource type has one or more data representations and one or more methods.

Resource types

Files

For Files Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/drive/v2, unless otherwise noted		
get	`GET /files/fileId`	Gets a file's metadata by ID.
insert	`POST https://www.googleapis.com/upload/drive/v2/files` and `POST /files`	Insert a new file.
patch	`PATCH /files/fileId`	Updates file metadata. This method supports patch semantics.
update	`PUT https://www.googleapis.com/upload/drive/v2/files/fileId` and `PUT /files/fileId`	Updates file metadata and/or content.
copy	`POST /files/fileId/copy`	Creates a copy of the specified file.

Getting started with the REST API

The foundation of all digital integrations with LinkedIn

The REST API is the heart of all programatic interactions with LinkedIn. All other methods of interacting, such as the JavaScript and Mobile SDKs, are simply wrappers around the REST API to provide an added level of convienence for developers. As a result, even if you are doing mobile or JavaScript development, it's still worth taking the time to familiarize yourself with how the REST API works and what it can do for you.

API methods

by section | by oauth scope

account

/api/v1/me oauth
/api/v1/me/blocked oauth
/api/v1/me/friends oauth
/api/v1/me/karma oauth
/api/v1/me/prefs oauth
/api/v1/me/trophies oauth
/prefs/blocked oauth
/prefs/friends oauth
/prefs/messaging oauth
/prefs/trusted oauth
/prefs/*where* oauth

captcha

/api/needs_captcha oauth

flair

/api/clearflairtemplates oauth
/api/deleteflair oauth
/api/deleteflairtemplate oauth
/api/flair oauth
/api/flairconfig oauth

This is automatically-generated documentation for the reddit API.

The reddit API and code are open source. Found a mistake or interested in helping us improve? Have a gander at api.py and send us a pull request.

Please take care to respect our API access rules.

overview

listings

Many endpoints on reddit use the same protocol for controlling pagination and filtering. These endpoints are called Listings and share five common parameters: `after` / `before`, `limit`, `count`, and `show`.

Listings do not use page numbers because their content changes so frequently. Instead, they allow you to view slices of the underlying data. Listing JSON responses contain `after` and `before` fields which are equivalent to the "next" and "prev" buttons on the site and in combination with `count` can be used to page through the listing.

The common parameters are as follows:

- `after` / `before` - only one should be specified. these indicate the fullname of an item in the listing to use as the anchor point of the slice.
- `limit` - the maximum number of items to return in this slice of the listing.
- `count` - the number of items already seen in this listing. on the html site, the builder uses this to determine when to give values for `before` and `after` in the response.

Twitter Developer Documentation

Products & Services

- Best practices
- API overview
- Twitter for Websites
- Twitter Kit
- Cards
- OAuth
- REST APIs
  - API Rate Limits
  - Rate Limits: Chart
  - The Search API
  - The Search API: Tweets by Place

REST APIs

The REST APIs provide programmatic access to read and write Twitter data. Create a new Tweet, read user profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are in JSON format.

If your intention is to monitor or process Tweets in real-time, consider using the Streaming API instead.

Overview

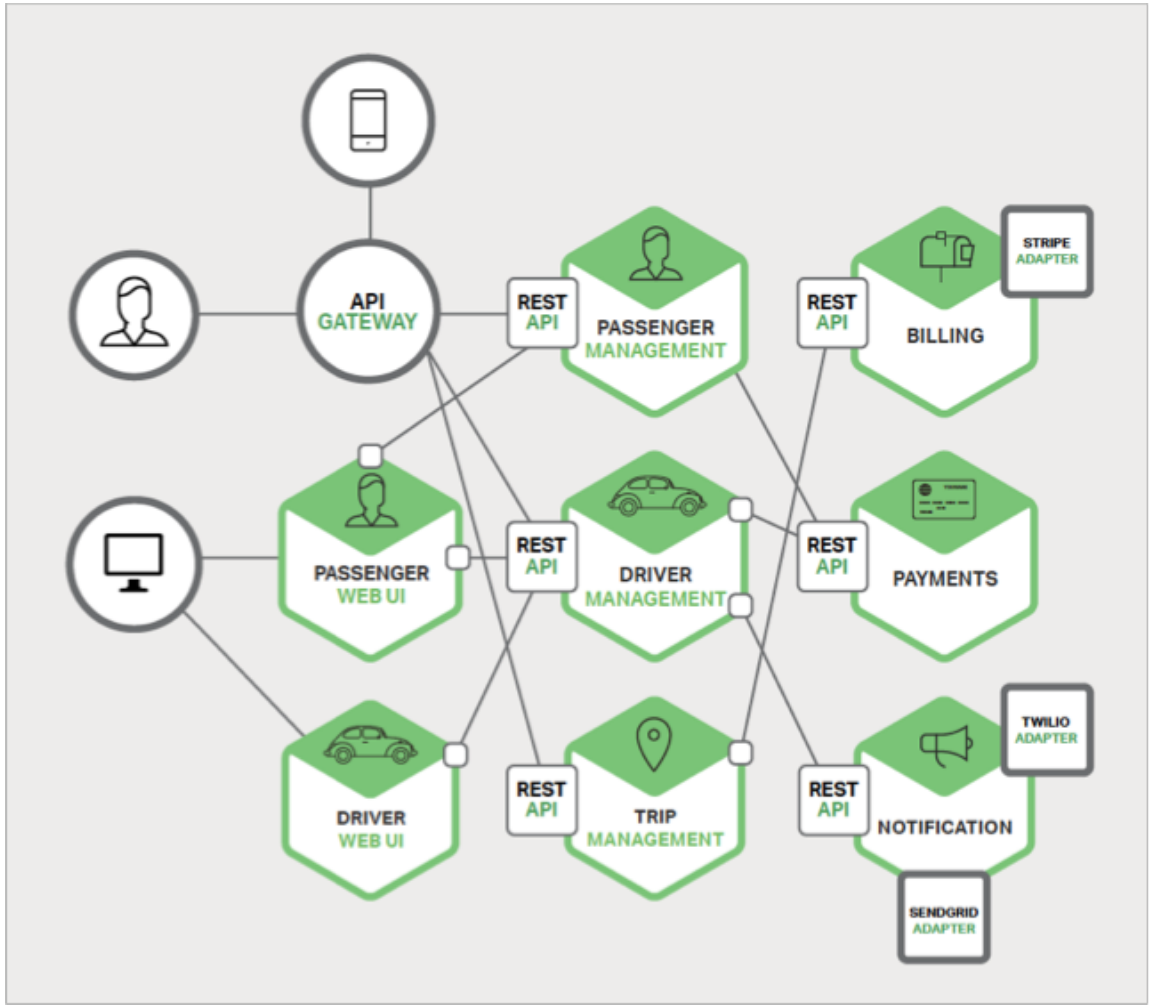
Below are some documents that will help you get going with the REST APIs as quickly as possible

- API Rate Limiting
- API Rate Limits
- Working with Timelines
- Using the Twitter Search API
- Finding Tweets about Places
- Uploading Media
- Reference Documentation

Default entities and retweets

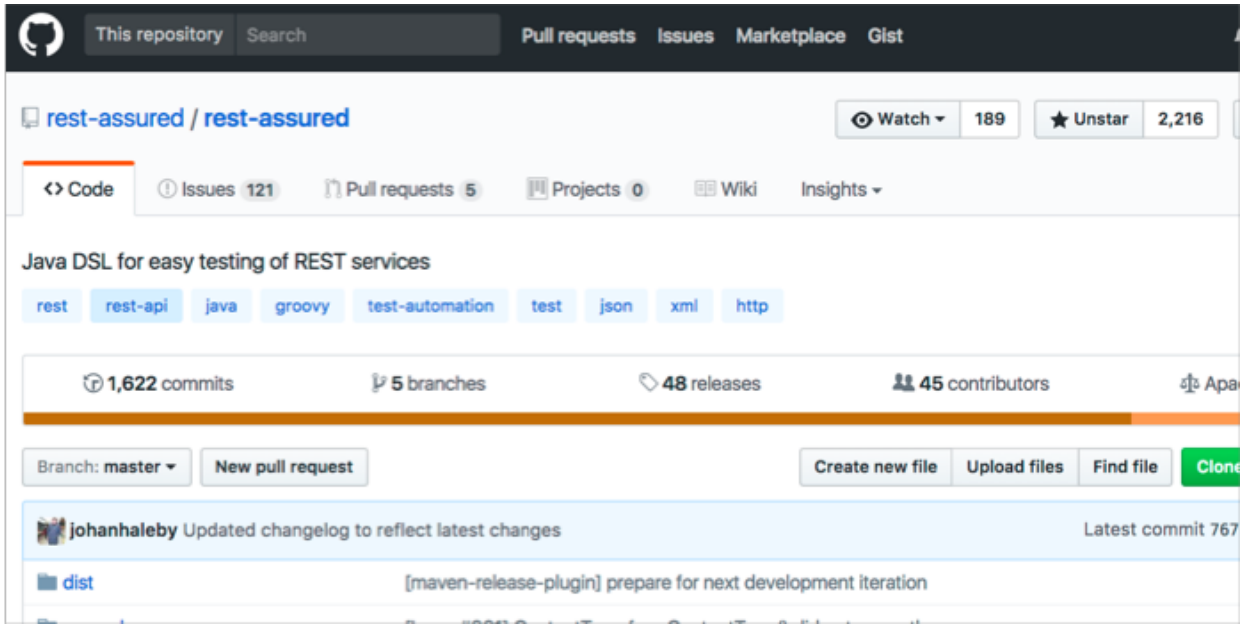
REST in Microservices

- Common trend in enterprises
  - Amazon, Netflix, etc.
- Split application in many small web services, typically REST
- Easier to scale and maintain
- User (browser/app) has no idea on how backed is architectured, only see one entry point, eg “API Gateway”



Testing of REST APIs

- Do HTTP calls, read responses
- Setup database states
- Specialized libraries, eg in Java the popular **RestAssured**




```
@Test
public void test0() throws Exception {

        given().header("Authorization", "ApiKey user")
                .accept("*/*")
                .get("www.foo.com/api/v1/media_files/42")
               .then()
                .statusCode(200);
}
```

Privat Bedrift

Søk

Logg inn hos Telenor

Logg inn på Mine sider

Brukernavn (e-postadresse)

Glemt brukernavn?

Passord

Glemt passord?

Husk brukernavn

Logg inn

Ny bruker på Mine sider

Registrer ny bruker

Administrer bedriftens abonnement på Min Bedrift.

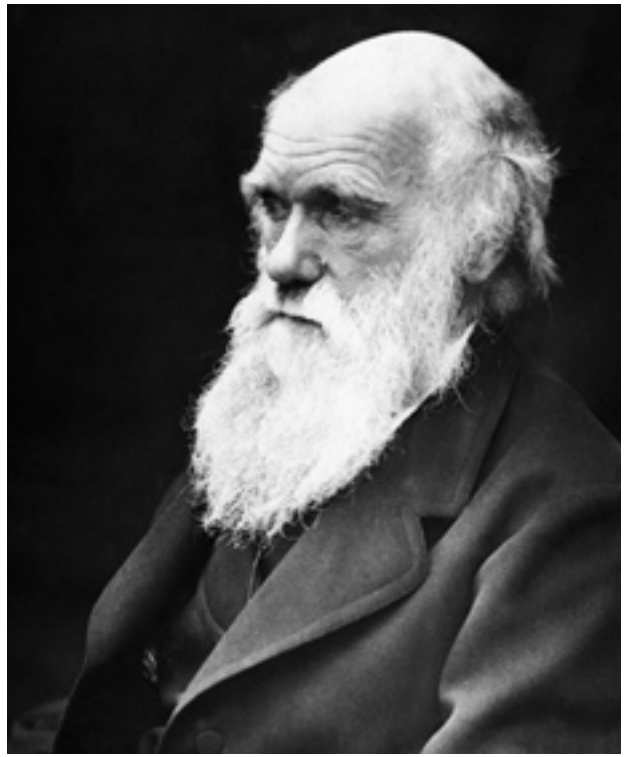
Med Mitt Telenor-appen får du full oversikt over forbruket ditt, både i Norge og utlandet. Last ned Mitt Telenor

What about **Automated Test Generation** for RESTful APIs?

- Would be very useful for enterprises
- No tool available (AFAIK)
  - (that I could use when I worked as a test engineer)
- In the past, quite a lot of work on **SOAP** web services
  - (which are not so common any more)
- Very few papers on testing REST
- Most techniques are **black box**

Search-Based Software Testing (SBST)

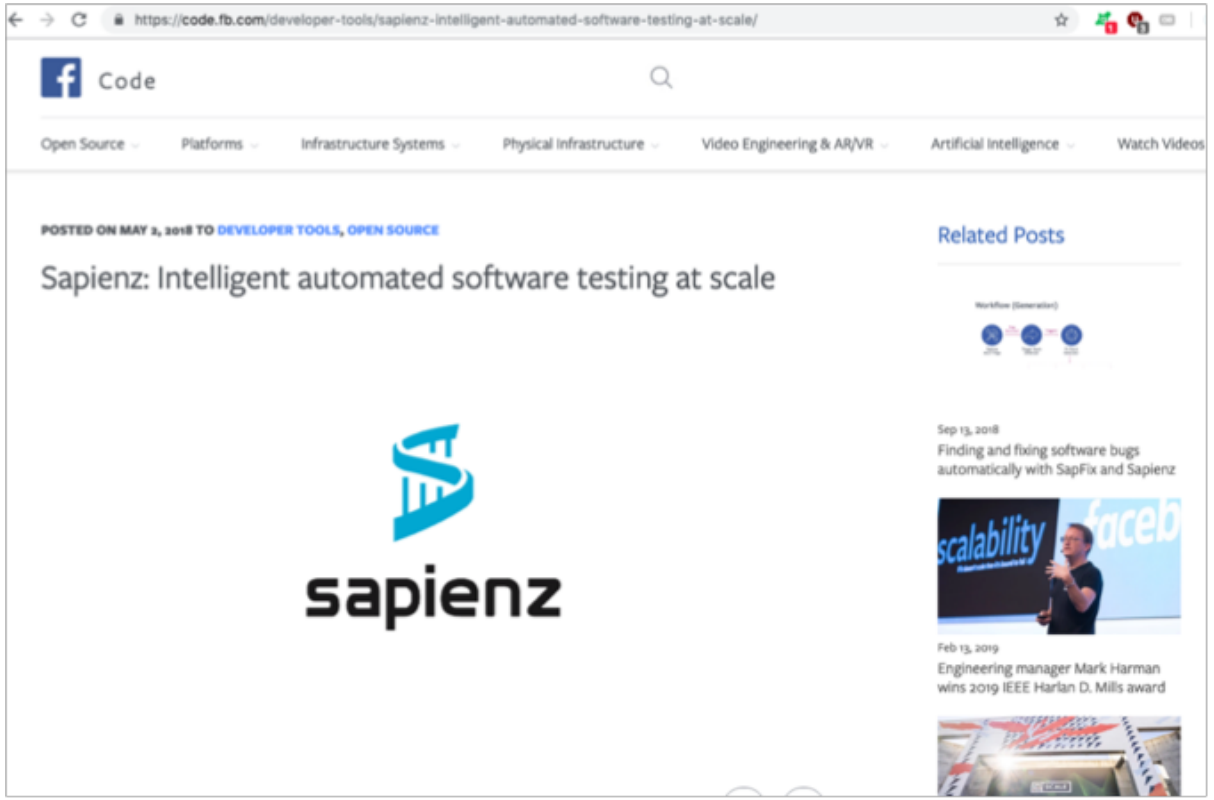
- Biology meets Software Engineering (SE)
- Casting SE problems into *Optimization Problems*
- *Genetic Algorithms*: one of most famous optimization algorithm, based on theory of evolution
- *Evolve* test cases



Success Stories: **Facebook**

Facebook uses SBST for automatically testing their software, especially their mobile apps

- eg, tools like *Sapienz* and *SapFix*



Properties of Optimization Problems

- 2 main components: *Search Space* and *Fitness Function*
- **Goal**: find the best solution from the search space such that the fitness function is minimized/maximized

Search Space

- Set X of all possible solutions for the problem
- If a solution can be represented with 0/1 bit sequence of length N, then search space is all possible bit strings of size N
  - any data on computer can be represented with bitstrings
- Search space is usually huge, eg 2^N
  - Otherwise use brute force, and so would not be a problem

Fitness Function

- $f(x)=h$
- Given a solution x in X, calculate an heuristic h that specifies how good the solution is
- Problem dependent, to minimize or maximize:
  - Maximize code coverage
  - Maximize fault finding
  - Minimize test suite size
  - etc.

Optimization Algorithms

- Algorithm that explores the search space X
- Only a tiny sample of X can be evaluated
- Use fitness *f(x)* to guide the exploration to fitter areas of the search space with better solutions
- Stopping criterion: after evaluating K solutions (or K amount of time is passed), return best *x* among the evaluated solutions
- Many different kinds of optimization algorithms…
  - But as a user, still need to provide the representation and f(x)

Search Operator

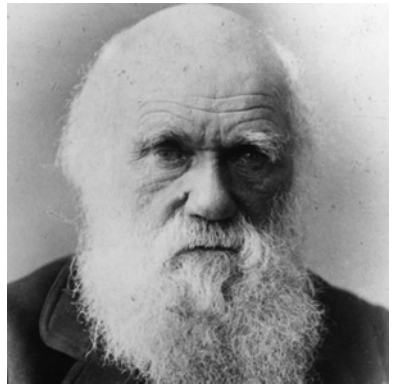
- $s(x) = x'$
- An operator that, from a solution x, gives a new one x'
- Still need to evaluate its fitness, ie $f(x')$
- The optimization algorithm will use the search operators to choose which new x' in X to evaluate
- The search operator will depend on the problem representation
- Example: flip a bit in a bit-sequence representation

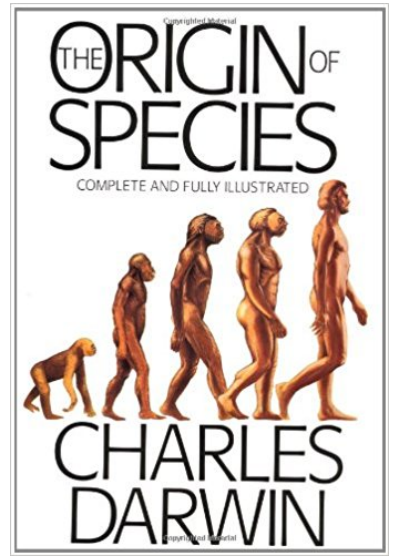
Nature Inspired Algorithms

- Nature is good at solving many different problems
- Idea: get inspiration from natural phenomena to create effective optimization algorithms
- E.g., *carbon-to-diamond process*: high temperature in Earth's mantle, cooled slowly while raising up to surface
  - *Simulated Annealing Algorithm*
- E.g., behavior of *ants seeking a path* between their colony and a source of food, based on pheromone trails
  - *Ant Colony Optimization Algorithm*

Theory Evolution

- Charles Darwin, *"The Origin of Species",* 1859
- Theory describing how different species *evolved* from unicellular organisms



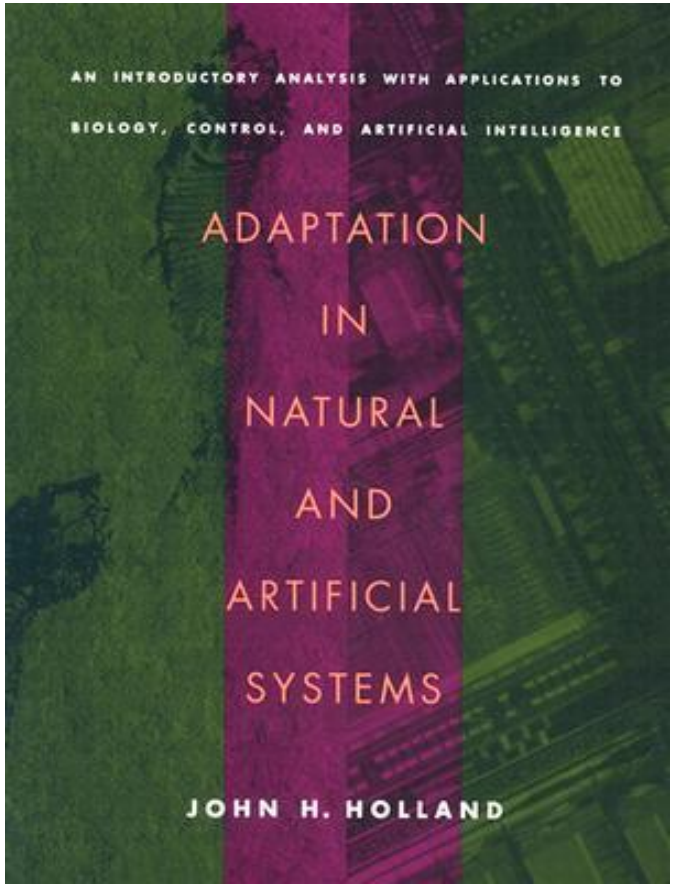


Evolutionary Algorithms (EAs)

- EAs are optimization algorithms based on the theory of evolution
- **(1+1) EA**: the simplest EA
- **Genetic Algorithms**: the most popular EA
- But there are more…

Genetic Algorithms (GAs)

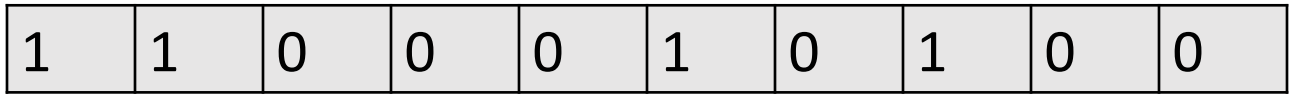
- 1950s: *Turing* proposed use of evolution in computer programs
- 1970s: John Holland created GAs to address optimization problems
- Simulate evolution of an entire *population* of individuals, which procreate sexually



GA Components

- *Chromosome*: the actual representation of the individuals, eg binary string
- *Population*: keep track of several individuals
- *Generation*: individuals mate and reproduce
- *Selection*: how mating is done, based on *fitness function*
- *Mutation*: offspring might have some of their “DNA” mutated
- *Crossover*: offspring inherit genetic material from the parents

Chromosome



- A solution to an optimization problem will be represented with some data
- In a computer, any stored data can be seen as 0/1 bitstring
- Bitstring is a common representation, but there can be customized for any addressed domain
  - eg, test cases as sequences of function calls with their inputs

Population

- Keep track of several individuals which we try to optimize
- At the end of the search, return the best solution in the population

Generations

- The search will be composed of 1 or more *generations*
- At each generation, select individuals for reproduction
- Create a new generation of same size K
- Kill the previous generation
  - Yes, evolution is really cruel…

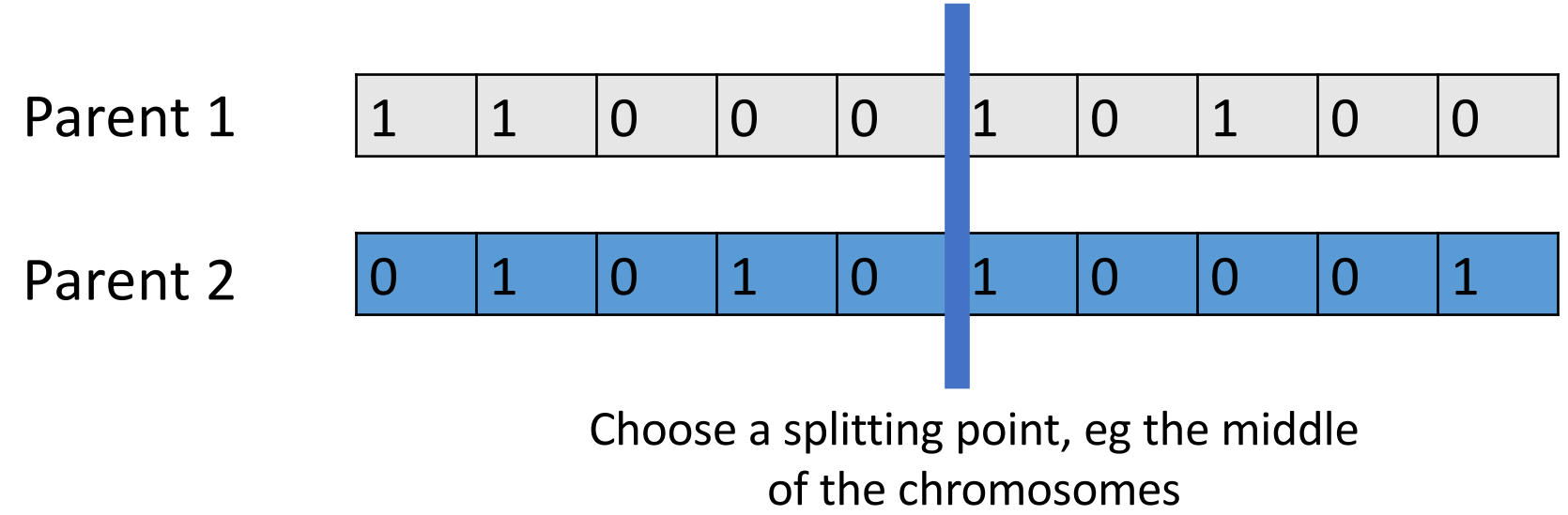
Selection

- The fittest individuals will have higher chances of reproduction
- Different strategies for parent selection
- Tournament Selection: sample T individuals *randomly* from the population, and choose the best among them
- *Fitness function is used to determine who is better*

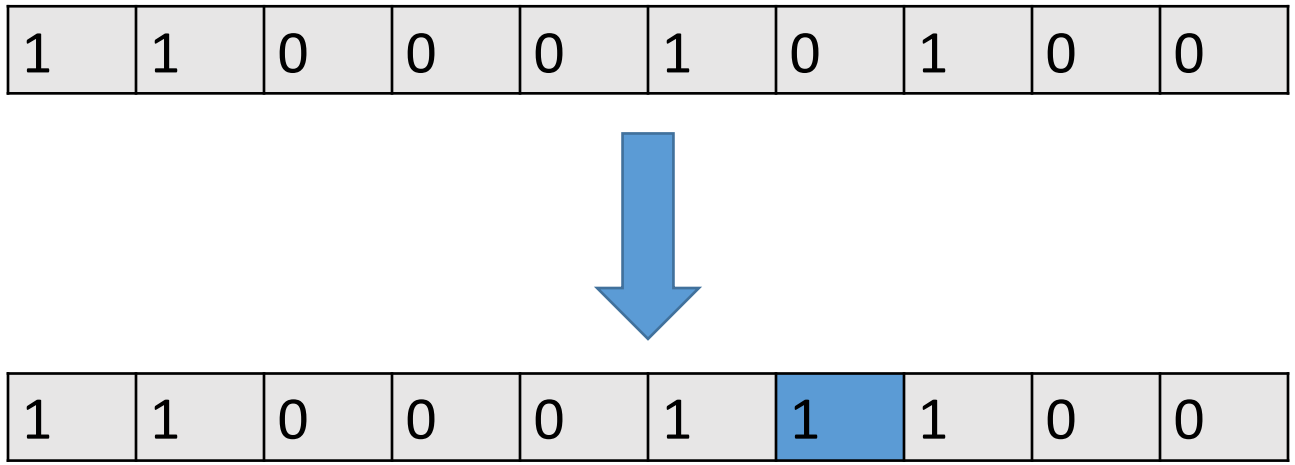
Sexual Reproduction

- Select 2 parents, which give birth to 2 offspring
  - Note, ignoring gender here…
- Offspring will share genetic material with their parents, via the *crossover* operation (aka xover)
- After xover, offspring still have chances of getting mutated

Crossover


Parent 1
1 1 0 0 0 1 0 1 0 0
Parent 2
0 1 0 1 0 1 0 0 0 1
Choose a splitting point, eg the middle of the chromosomes
Offspring 1
1 1 0 0 0 1 0 0 0 1
Offspring 2
0 1 0 1 0 1 0 1 0 0

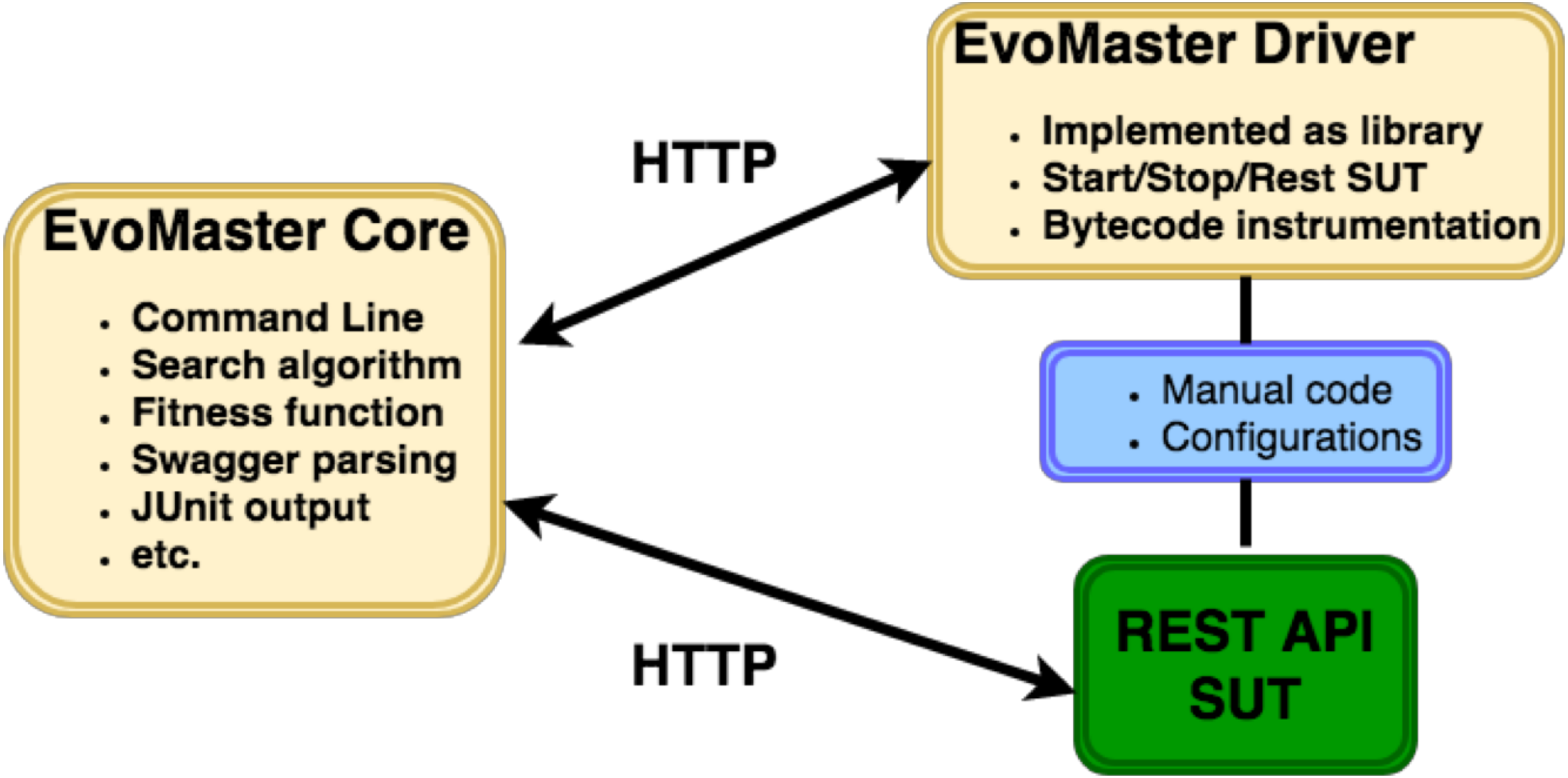
Mutation in 0/1 Sequence



- Mutation: given N bits, each bit can be flipped with probability 1/N
  - Eg 10% probability in the above example
- *On average,* only 1 bit is flipped per mutation operation

EvoMaster

- Tool to automatically generate tests for REST APIs
- **White box**
  - can exploit structural and runtime information of the SUT
- Search-based testing technique (**SBST**)
  - Evolutionary, Genetic Algorithms
- Fully automated
- Open-source prototype: *www.evomaster.org*
- Currently targeting JVM languages (eg Java and Kotlin)


EvoMaster Driver
Implemented as library
Start/Stop/Rest SUT
Bytecode instrumentation
HTTP
EvoMaster Core
Command Line
Search algorithm
Fitness function
Swagger parsing
JUnit output
etc.
Manual code
Configurations
HTTP
REST API
SUT

OpenAPI/Swagger

- REST is not a protocol
- Need to know what endpoints are available, and their parameters
- Schema defining the APIs
- Swagger is the most popular one
- Defined as JSON file, or YAML
- Many REST frameworks can automatically generate Swagger schemas from code

EvoMaster Core

- From Swagger schema, defines set of endpoints that can be called
- Test case structure:
  1. setup initializing data in DB with SQL INSERTs
  2. sequence of HTTP calls toward such endpoints
- HTTP call has many components:
  - Verb (GET, POST, DELETE, etc.)
  - Headers
  - Query parameters
  - Body payload (JSON, XML, etc.)
- Evolutionary algorithm to evolve such sequences and their inputs
- Output: *self-contained* JUnit tests
- Code language of SUT is *irrelevant*, as we use HTTP to communicate with it

Fitness Function

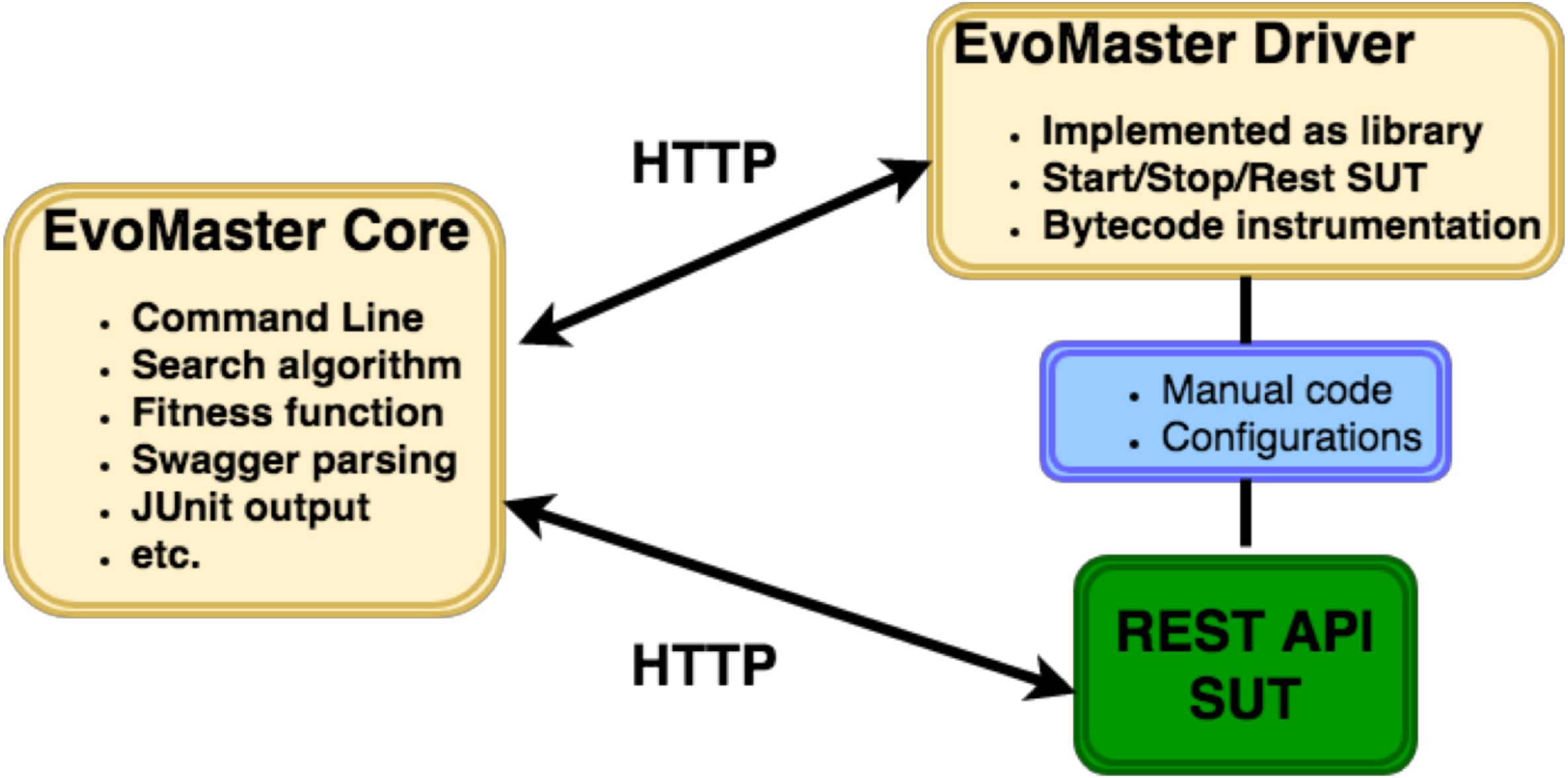
- Needed to drive the evolution
- Reward code coverage and fault detection
- HTTP return statuses as *automated oracles*:
  - Eg 2xx if OK, 4xx are user errors, but **5xx** are server errors (often due to bugs)
- Need guidance to be able to solve constraints in code predicates
  - “if(x == 123456 && complexPredicate(y))”
- Unlikely to achieve high code coverage with just random inputs

SBST Heuristics: Branch Distance

- Standard technique in the SBST literature, usually for *unit* testing
- Example: if(x==100)
- Both 5 and 90 do not solve the constraint, but 90 is *heuristically* closer
- Not just for integers, but also all other types, eg strings
- Need to *instrument* the code to calculate those branch distances
- *Bytecode manipulation*: EvoMaster does it *fully automatically* with class loaders and Java Agents
- Lot of technical details on how to achieve it efficiently

EM Driver: SBST Heuristics as a Service

- Core and Driver are running on different processes
- Code coverage and branch distances sent over the net, in JSON format
- Cannot send all data: too inefficient if per test execution
  - different techniques to determine only what is necessary
- *EM Driver is itself a RESTful API*
- **Why?** Because so we can use Driver for other languages (eg C# and JS) without the need to touch EM Core


EvoMaster Driver
Implemented as library
Start/Stop/Rest SUT
Bytecode instrumentation
HTTP
EvoMaster Core
Command Line
Search algorithm
Fitness function
Swagger parsing
JUnit output
etc.
Manual code
Configurations
HTTP
REST API
SUT

Search Algorithms for System Testing

- There are many search algorithms
  - Genetic Algorithms, Simulated Annealing, Ant Colony, etc.
- No Free Lunch Theorem
  - on all possible problems, all algorithms have same average performance, ie*, there is no best algorithm*
- Customized algorithms that exploit domain knowledge will give better results

Properties of System Testing (for Web Services)

- To increase coverage, you can add new tests to existing test suite
  - Testing objectives can be sought *independently*
  - Minimizing the number of tests is still important, but secondary
- System tests are expensive to run (eg compared to unit tests)
  - Less number of fitness evaluations: put more emphasis on *exploitation* vs *exploration* of the search landscape
- Many, many test objectives (e.g., lines and branches)
  - Even in the tens/hundreds of thousands…
- Some test objectives could be *infeasible*
  - Any resource spent in covering them is wasted

Many Independent Objective (MIO) Algorithm

- Multi-objective optimization
- Evolve populations of *test cases*
- Final output: a *test suite*
- At a high level, it can be considered like a multi-population (1+1)EA

Dynamic Number of Populations

- One population of tests *for each* testing target (eg, line or branch)
- Each population has up to *N* tests
- Initially 0 populations.
- Every time a target is reached but not covered, we create a population for it
  - Eg, a code block inside an *if* statement with complex predicate
- *Why*? Before running, do not know how many targets there are.

MIO Main Loop

```
override fun search(): Solution<T> {
    time.startSearch()

    while(time.shouldContinueSearch()){
      val randomP = apc.getProbRandomSampling()
      if(archive.isEmpty() || randomness.nextBoolean(randomP)) {
         val ind = sampler.sample()
         ff.calculateCoverage(ind)?.run { archive.addIfNeeded(this) }
         continue
      }
      var ei = archive.sampleIndividual()
      val nMutations = apc.getNumberOfMutations()
      getMutatator().mutateAndSave(nMutations, ei, archive)
    }

    return archive.extractSolution()
}
```

- Each iteration, sample a test
- Sample either at random, or from one population for non-covered targets, based on probability *P*
- If from population, apply *M* mutations
- Might copy to all existing populations in archive

Mutation Operator

- Standard, like in any evolutionary algorithm
- Small modifications
  - +- delta on numbers
  - change some characters in strings
  - etc.

Population Management

- A single population hold up to *N* tests (eg 10)
- Tests in a population X have fitness values based *only* on target X.
- When adding new test and size become > N, delete worst test
- When sampling from population, done at random
- If target gets covered, population shrinks to 1, keeping best test

Population Choice

- After few iterations, might have many populations
- In main loop, sample 1 test from 1 population
- Population is chosen at random among targets *not fully covered* yet
  - Ie, concentrate search on targets that still need to be covered

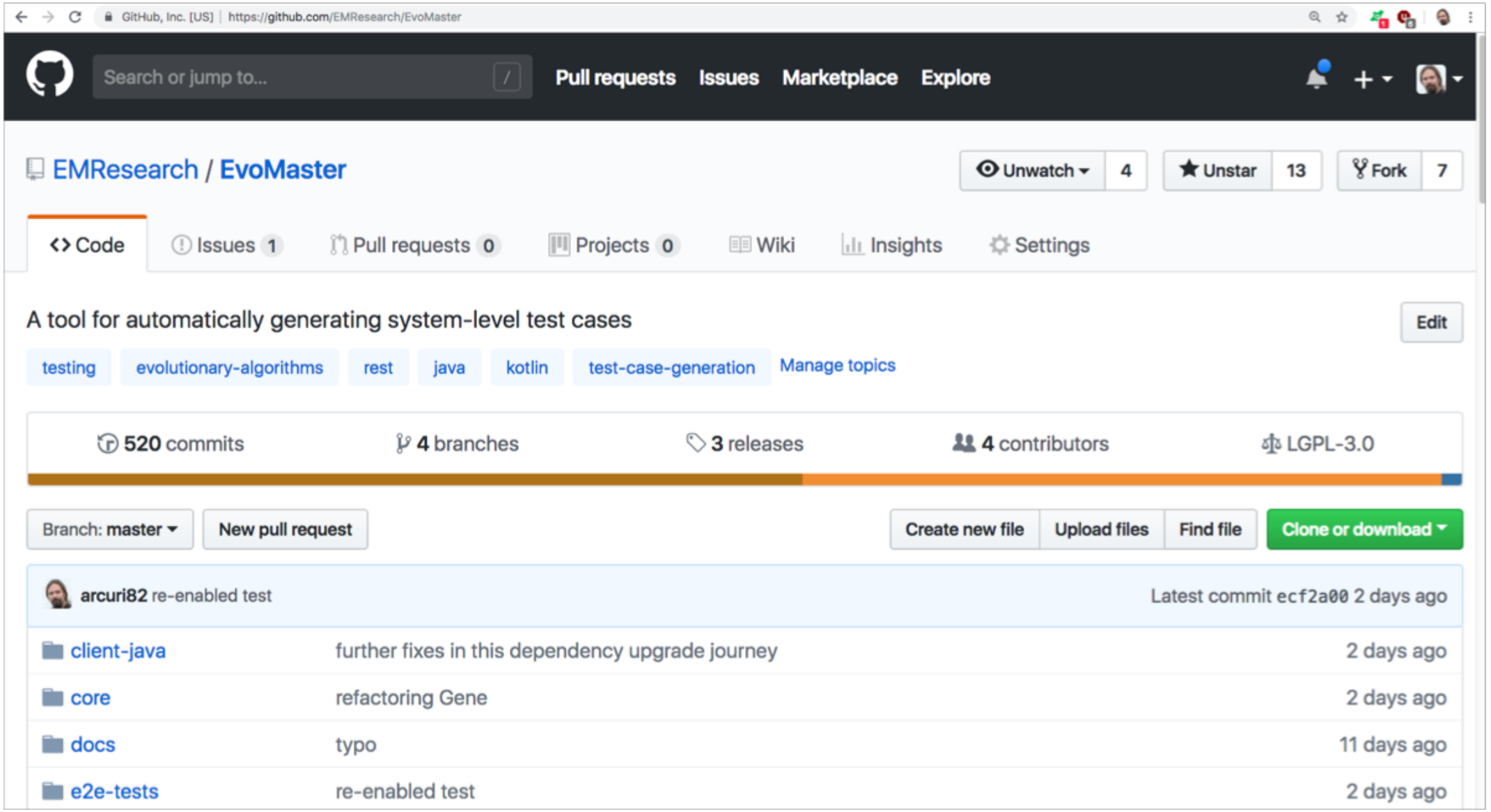
Feedback-directed Sampling

- If target is *infeasible,* waste of time sampling from its population
- But not possible to determine if target is infeasible
- Solution:
  - Add counter to each population, initialized to 0
  - Each time sampling from population, increase counter by 1
  - New better test added to population? Reset counter to 0
  - When choosing population to sample, *instead of* random, choose *lowest counter*
- Effects: concentrate search on targets for which we get fitness improvements
  - After a while, never sample again from populations for infeasible targets

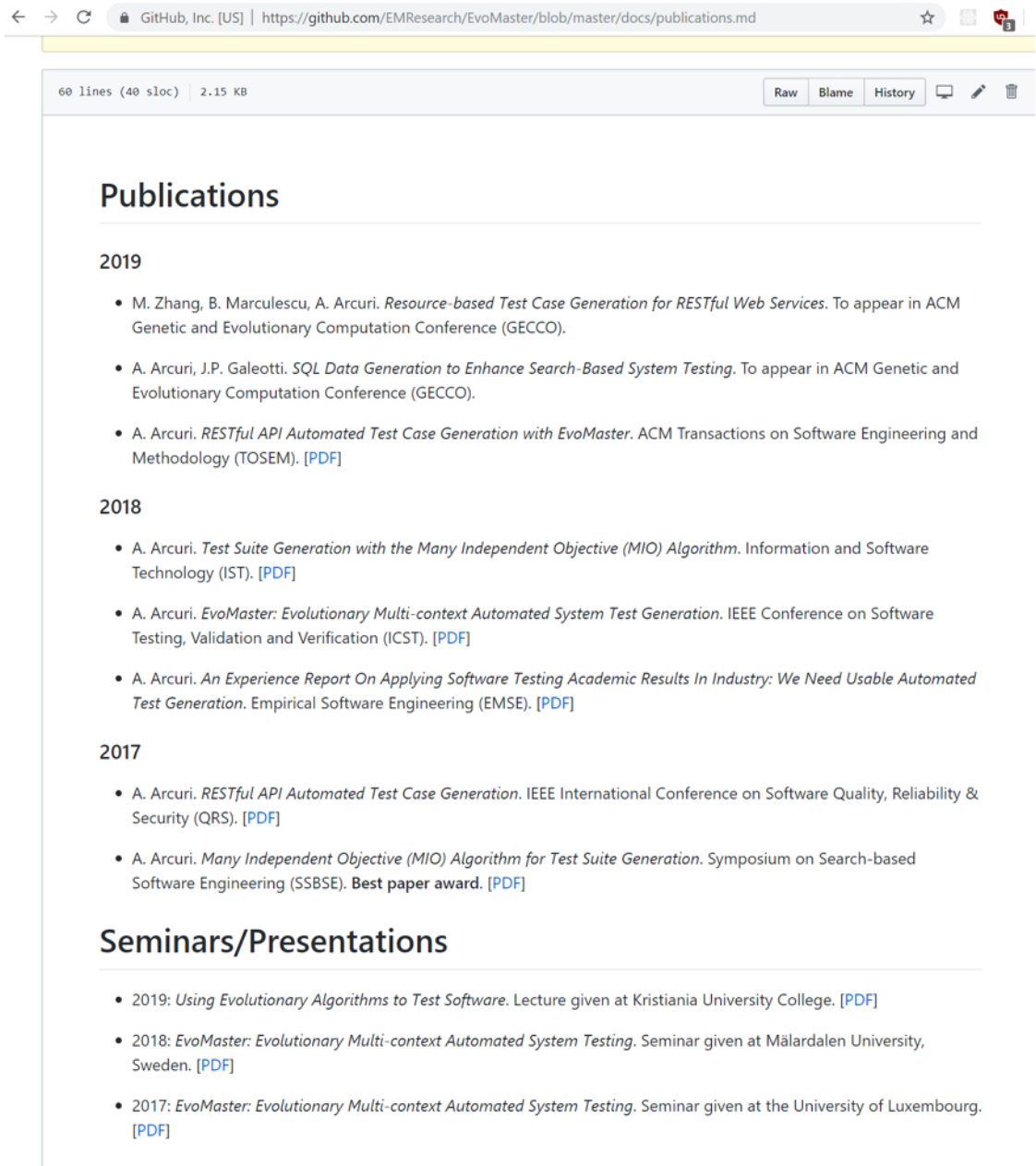
Exploration vs Exploitation

- Beginning of search, want to explore large parts of search landscape
- Later in the search, concentrate to improve current best tests
  - Not much left time to try new different tests
- This is controlled by 2 main parameters: sampling *P*, population size *N*
- *Parameter Control*: during search, decrease *P* and *N*
  - Similar to Simulated Annealing
  - When *N* shrink, remove worst test

www.evomaster.org



Slides and Paper PDFs all online



GitHub, Inc. [US] | https://github.com/EMResearch/EvoMaster/blob/master/docs/publications.md

60 lines (40 sloc) | 2.15 KB

Raw Blame History

Publications

2019

- M. Zhang, B. Marculescu, A. Arcuri. *Resource-based Test Case Generation for RESTful Web Services*. To appear in ACM Genetic and Evolutionary Computation Conference (GECCO).
- A. Arcuri, J.P. Galeotti. *SQL Data Generation to Enhance Search-Based System Testing*. To appear in ACM Genetic and Evolutionary Computation Conference (GECCO).
- A. Arcuri. *RESTful API Automated Test Case Generation with EvoMaster*. ACM Transactions on Software Engineering and Methodology (TOSEM). [PDF]

2018

- A. Arcuri. *Test Suite Generation with the Many Independent Objective (MIO) Algorithm*. Information and Software Technology (IST). [PDF]
- A. Arcuri. *EvoMaster: Evolutionary Multi-context Automated System Test Generation*. IEEE Conference on Software Testing, Validation and Verification (ICST). [PDF]
- A. Arcuri. *An Experience Report On Applying Software Testing Academic Results In Industry: We Need Usable Automated Test Generation*. Empirical Software Engineering (EMSE). [PDF]

2017

- A. Arcuri. *RESTful API Automated Test Case Generation*. IEEE International Conference on Software Quality, Reliability & Security (QRS). [PDF]
- A. Arcuri. *Many Independent Objective (MIO) Algorithm for Test Suite Generation*. Symposium on Search-based Software Engineering (SSBSE). **Best paper award**. [PDF]

Seminars/Presentations

- 2019: *Using Evolutionary Algorithms to Test Software*. Lecture given at Kristiania University College. [PDF]
- 2018: *EvoMaster: Evolutionary Multi-context Automated System Testing*. Seminar given at Mälardalen University, Sweden. [PDF]
- 2017: *EvoMaster: Evolutionary Multi-context Automated System Testing*. Seminar given at the University of Luxembourg. [PDF]

Demo