An introduction to

Property based testing

with

Kotest

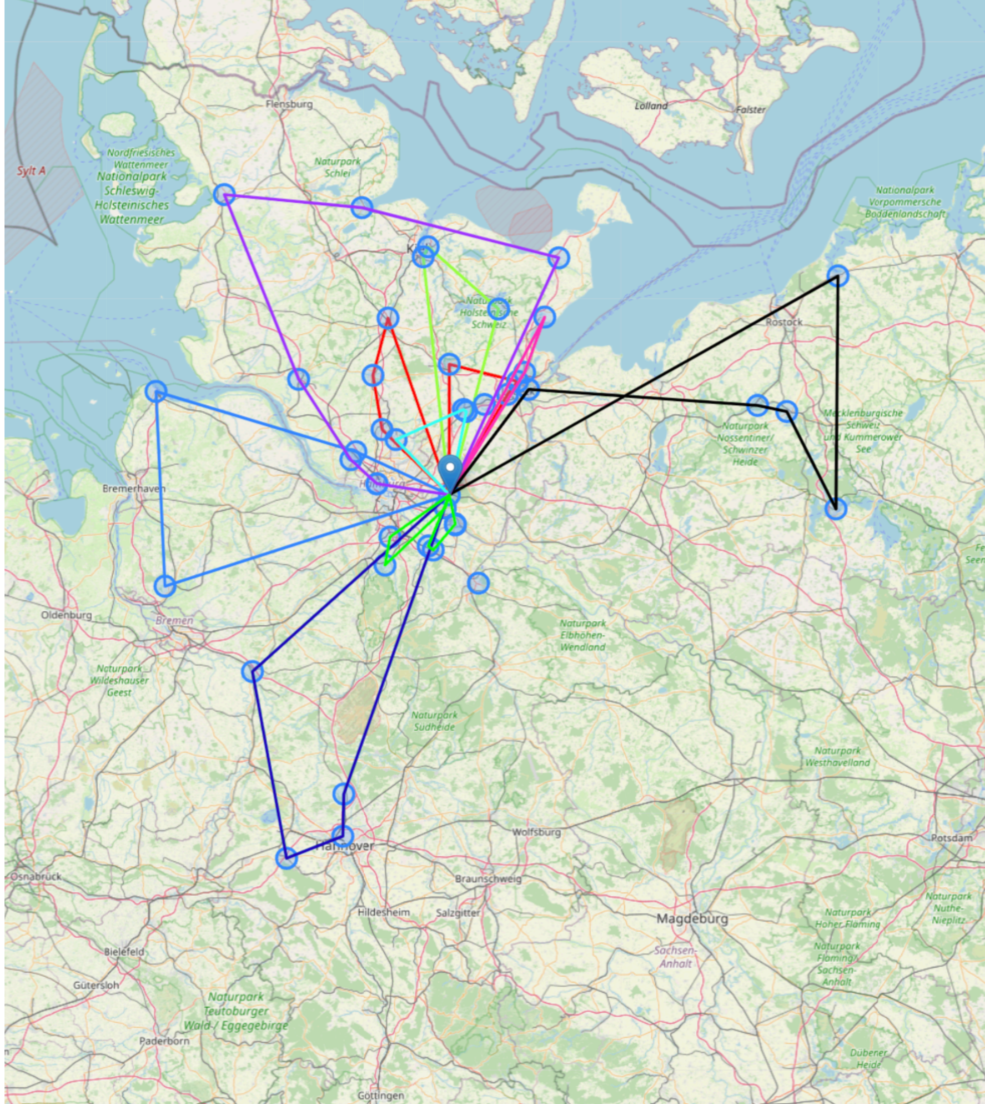
Peter Laggner

- 🏡 **Based** in Graz
- 🧑‍💻 **Working** remotely at Cargonexx
- 🚚 **Modelling and solving** a vehicle routing problem using Kotlin and Timefold
- **Kotlin** enthusiast since 2017

greyhairredbear

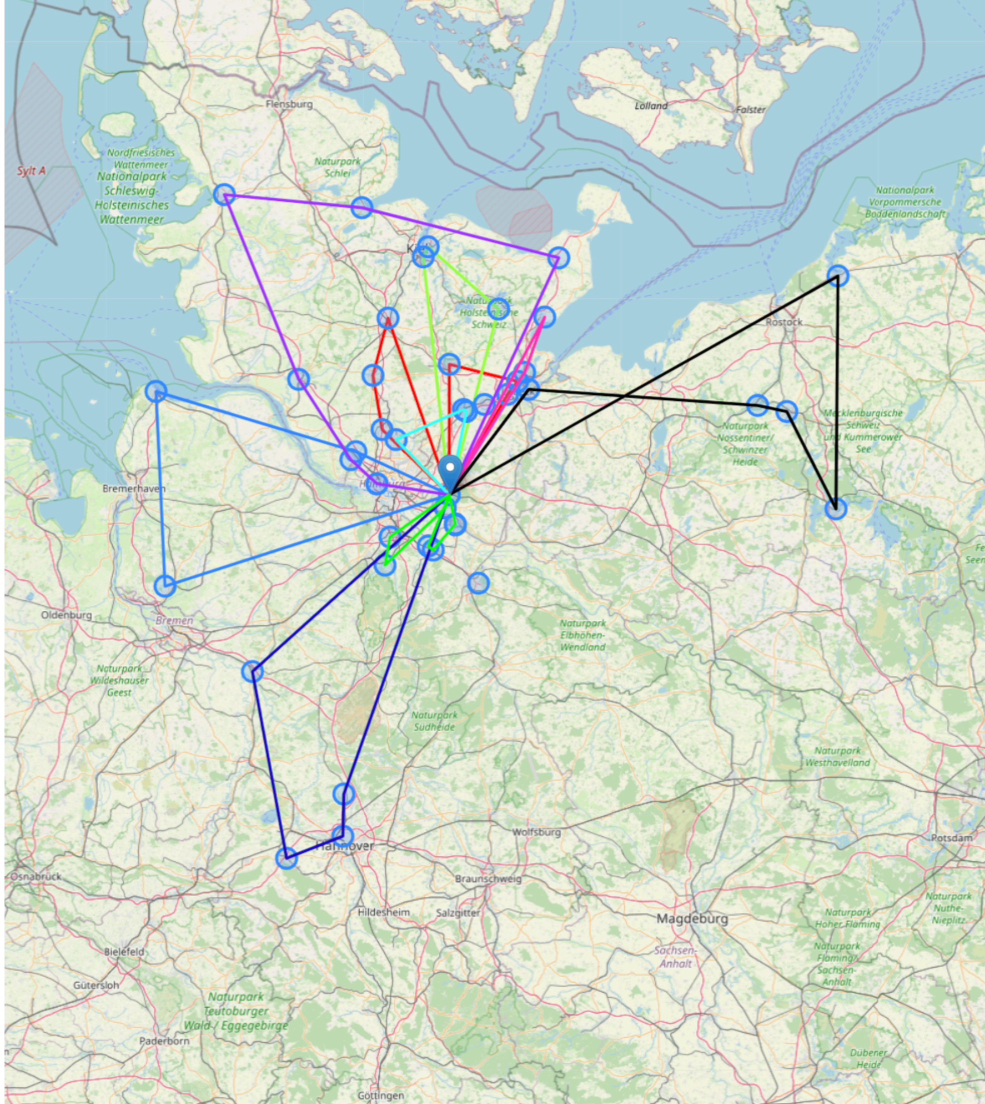
greyhairredbear

Motivation


Flensburg
Lolland
Falster
Sylt A
Nordfriesisches Wattenmeer Nationalpark Schleswig-Holsteinisches Wattenmeer
Naturpark Schlei
Kiel
Naturpark Holsteinische Schweiz
Nationalpark Vorpommersche Boddenlandschaft
Rostock
Naturpark Nossentiner/Schwinzer Heide
Mecklenburgische Schweiz und Kummerower See
Bremerhaven
Hamburg
Oldenburg
Bremen
Naturpark Elbhöhen-Wendland
Naturpark Wildeshauser Geest
Naturpark Südheide
Naturpark Westhavelland
Hannover
Wolfsburg
Potsdam
Osnabrück
Braunschweig
Hildesheim
Salzgitter
Magdeburg
Bielefeld
Sachsen-Anhalt
Gütersloh
Naturpark Teutoburger Wald / Eggegebirge
Paderborn
Naturpark Hoher Fläming
Naturpark Fläming/Sachsen-Anhalt
Naturpark Nuthe-Nieplitz
Dübener Heide
Göttingen

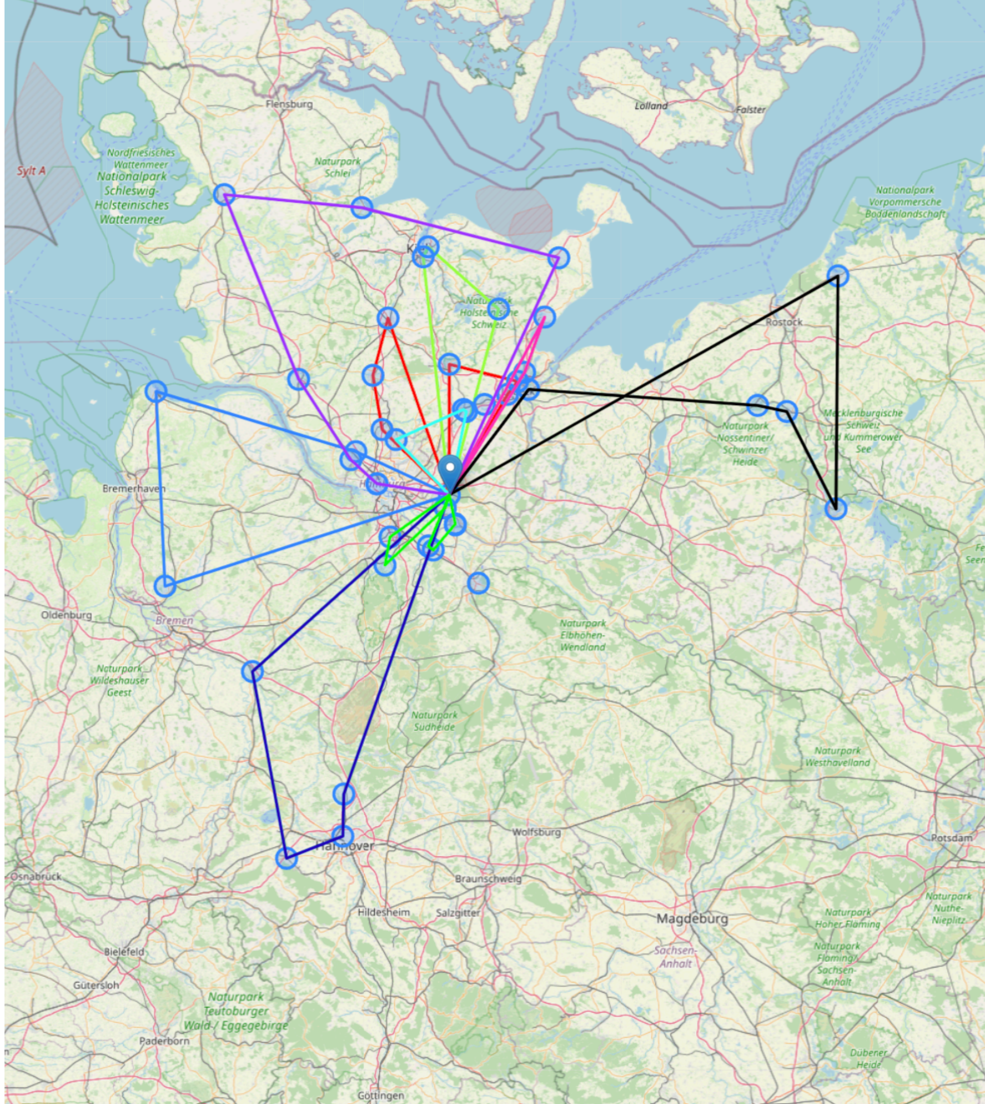
Motivation

- Vehicle Routing Problem (VRP)
    - Complex domain
    - Computationally complex



Motivation

- Vehicle Routing Problem (VRP)
  - Complex domain
  - Computationally complex
- Build trust with potential users
  - Ensure complex problem is solved correctly



A simple exercise*

"Write a function that adds two numbers..."

*shamelessly adapted from Scott Wlaschin

A simple exercise*

"Write a function that adds two numbers..."

…using altered form of ping-pong TDD

- you're paired with a colleague
- your job: Only write the tests

*shamelessly adapted from Scott Wlaschin

Meet your colleague

Meet your colleague

- Some say, he can be a bit tricky to work with

Meet your colleague

- Some say, he can be a bit tricky to work with
- Others say, he is a bit cynical about TDD

Meet your colleague

- Some say, he can be a bit tricky to work with
- Others say, he is a bit cynical about TDD
- Some even call him "Enterprise Developer from Hell" (EDFH)

Meet your colleague

- Some say, he can be a bit tricky to work with
- Others say, he is a bit cynical about TDD
- Some even call him "Enterprise Developer from Hell" (EDFH)
- But how bad can it get?

Let’s get to it

```
add(1, 2) shouldBe 3
```

Let’s get to it

```kotlin
fun add(a: Int, b: Int): Int = 3
```

Let’s get to it

```
add(1, 2) shouldBe 3
add(29, 13) shouldBe 42
add(-1, 5) shouldBe 4
add(1024, 1024) shouldBe 2048
```

Let’s get to it

```kotlin
fun add(a: Int, b: Int): Int = when {
    a == 1 && b == 2 -> 3
    a == 29 && b == 13 -> 42
    a == -1 && b == 5 -> 4
    a == 1024 && b == 1024 -> 2048
    else -> 31415926 // ¯\_(ツ)_/¯
}
```

Let’s get to it

```
// (╯°□°）╯︵ ┻━┻
```

Let’s try random input

Let’s try random input

```kotlin
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)
    val b = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(a, b) shouldBe a + b
}
```

Let’s try random input

```kotlin
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)
    val b = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(a, b) shouldBe a + b
}
```

Let’s try random input

```kotlin
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)
    val b = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(a, b) shouldBe a + b
}
```

EDFH doesn't really seem to bring out the best in us

Property based testing - Commutativity

Property based testing - Commutativity

```kotlin
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)
    val b = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(a, b) shouldBe add(b, a)
}
```

Property based testing - Commutativity

```kotlin
fun add(a: Int, b: Int): Int = a * b
```

Property based testing - Successor

Property based testing - Successor

```kotlin
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(add(a, 1), 1) shouldBe add(a, 2)
}
```

Property based testing - Successor

```kotlin
fun add(a: Int, b: Int): Int = 42
```

Property based testing - Identity

Property based testing - Identity

```
import kotlin.random.Random

repeat(1000) {
    val a = Random.nextInt(Int.MIN_VALUE / 2, Int.MAX_VALUE / 2)

    add(a, 0) shouldBe a
}
```

Property based testing - Identity

```kotlin
fun add(a: Int, b: Int): Int = a + b
```

Revisiting random inputs

How to …

Revisiting random inputs

How to …

- … make your test data generation reusable?

Revisiting random inputs

How to …

- … make your test data generation reusable?
- … reproduce test runs using random data?

Revisiting random inputs

How to …

- … make your test data generation reusable?
- … reproduce test runs using random data?
- … ensure seeds are deterministic for a test run?

Revisiting random inputs

How to …

- … make your test data generation reusable?
- … reproduce test runs using random data?
- … ensure seeds are deterministic for a test run?
- … shrink your test data?
  - *Shrinking*: Get the least complex input that fails your test

Kotest’s property test framework

Kotest's property test framework

- API for random input generation

Kotest's property test framework

- API for random input generation
- Lots of generators available (just like Kotest assertions)

Kotest’s property test framework

- API for random input generation
- Lots of generators available (just like Kotest assertions)
- Test shrinking

Kotest’s property test framework

- API for random input generation
- Lots of generators available (just like Kotest assertions)
- Test shrinking
- Nice documentation

Real example - VRP input pre-processing

Real example - VRP input pre-processing

- Every load entity makes computation harder

Real example - VRP input pre-processing

- Every load entity makes computation harder
- Grouping loads with same pickup and dropoff locations is beneficial (usually)

Real example - VRP input pre-processing

- Every load entity makes computation harder
- Grouping loads with same pickup and dropoff locations is beneficial (usually)

```
class Load(
    val id: String,
    val measurements: Measurements,
    val requirements: List<String>,
    pickupTimeWindow: Interval,
    val pickupLocation: Location,
    dropoffTimeWindow: Interval,
    val dropoffLocation: Location,
    val planningDate: LocalDate,
    val containedLoadIds: List<String> = listOf(id)
) { ... }
```

Real example - VRP input pre-processing

- Every load entity makes computation harder
- Grouping loads with same pickup and dropoff locations is beneficial (usually)

```
fun List<Load>.grouped(
    loadGroupingConfig: LoadGroupingConfig
): List<Load>
```

Real example - VRP input pre-processing

- Every load entity makes computation harder
- Grouping loads with same pickup and dropoff locations is beneficial (usually)

```
data class LoadGroupingConfig(
    // loading (milli)meters, weight in kg
    val maxMeasurements: Measurements,
    val minTimeWindowOverlapInHours: Long,
)
```

Real example - VRP input pre-processing

Real example - VRP input pre-processing

Property: Grouping a second time should not change the outcome

Real example - VRP input pre-processing

Property: Grouping a second time should not change the outcome

```
checkAll(
    inputLoadGen,
    groupingConfigGenerator
) { input, groupingConfig ->
    val result = input.grouped(groupingConfig)
    result shouldContainExactlyInAnyOrder result.grouped(groupingConfig)
}
```

Real example - VRP input pre-processing

Real example - VRP input pre-processing

Property: Don’t drop any loads during grouping

Real example - VRP input pre-processing

Property: Don't drop any loads during grouping

```
checkAll(
    inputLoadGen,
    groupingConfigGenerator
) { input, groupingConfig ->
    input.grouped(groupingConfig).flatMap {
        it.containedLoadIds
    } shouldContainExactlyInAnyOrder input.map { it.id }
}
```

Real example - VRP input pre-processing

Real example - VRP input pre-processing

Property: Don’t assign loads to more than one group

Real example - VRP input pre-processing

Property: Don't assign loads to more than one group

```
checkAll(
    inputLoadGen,
    groupingConfigGenerator
) { input, groupingConfig ->
  val result = input.grouped(groupingConfig)
  result.flatMap { it.containedLoadIds }.shouldNotContainDuplicates()
}
```

Real example - VRP input pre-processing

Real example - VRP input pre-processing

Property: Don’t add more loads during grouping

Real example - VRP input pre-processing

Property: Don't add more loads during grouping

```
checkAll(
    inputLoadGen,
    groupingConfigGenerator
) { input, groupingConfig ->
  input.grouped(groupingConfig).count() shouldBeLessThanOrEqual
      input.count()
}
```

Real example - VRP input pre-processing

Real example - VRP input pre-processing

Property: Adhere to maximum measurements of grouping configuration for grouping loads

Real example - VRP input pre-processing

Property: Adhere to maximum measurements of grouping configuration for grouping loads

```kotlin
forAll(inputLoadGen, groupingConfigGenerator) { input, groupingConfig ->
    val groupedLoadsMeasurements =
        input
            .grouped(groupingConfig)
            .filter { it.containedLoadIds.count() > 1 }
            .map { it.measurements }

    groupedLoadsMeasurements.none {
        it.ldmm > groupingConfig.maxMeasurements.ldmm ||
            it.weightInKg > groupingConfig.maxMeasurements.weightInKg
    }
}
```

Generators

Generators

DSL for generating random input

Generators

DSL for generating random input

```kotlin
fun givenLoadGroupingConfigGenerator(): Arb<LoadGroupingConfig> =
    arbitrary {
        LoadGroupingConfig(
          givenMeasurementGenerator().bind(),
          Arb.long(1L..96).bind(),
        )
    }
```

Assumptions

Assumptions

Filter test data with assertions

Assumptions

Filter test data with assertions

```
checkAll(givenVehicleGenerator()) {
    val firstAction = it.actions.firstOrNull()
    assume {
        firstAction.shouldNotBeNull()
        firstAction.isOnSameVehicleAsRelatedAction.shouldBeTrue()
    }

    firstAction!!.isScheduledBeforeRelatedAction.shouldBeTrue()
}
```

Remarks

Remarks

Property testing has limitations

Remarks

Property testing has limitations

- Sometimes too coarse

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

BUT

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

BUT

- Helps you think about your program in a more general way

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

BUT

- Helps you think about your program in a more general way
- Great for detection of edge cases

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

BUT

- Helps you think about your program in a more general way
- Great for detection of edge cases
- Property tests are less prone to change

Remarks

Property testing has limitations

- Sometimes too coarse
- Finding properties is not always easy

BUT

- Helps you think about your program in a more general way
- Great for detection of edge cases
- Property tests are less prone to change

Bottom line: Mix property tests with your regular example based tests!

References

References

- Scott Wlaschin on property testing:

  https://fsharpforfunandprofit.com/series/property-based-testing/

References

- Scott Wlaschin on property testing:

  https://fsharpforfunandprofit.com/series/property-based-testing/

- Kotest’s property testing framework:

  https://kotest.io/docs/proptest/property-based-testing.html

References

- Scott Wlaschin on property testing:
  https://fsharpforfunandprofit.com/series/property-based-testing/
- Kotest's property testing framework:
  https://kotest.io/docs/proptest/property-based-testing.html
- Slides created with slidev: https://sli.dev

References

- Scott Wlaschin on property testing:

  https://fsharpforfunandprofit.com/series/property-based-testing/
- Kotest's property testing framework:

  https://kotest.io/docs/proptest/property-based-testing.html
- Slides created with slidev: https://sli.dev
- This presentation: https://github.com/greyhairredbear/presentations

  (`/intro-property-based-testing`)

Questions