HPS
https://valerius.me

Bianca Vetter & Valerius Mattfeld

Cars in the traffic of a city network and resulting traffic jams in Go

Predicting and Identifying Traffic Bottlenecks using Go MPI Simulations

Table of contents

Why Go?

- Open source

Figure: Go Brand Logo

The Go Programming Language, Documentation - The Go Programming Language, Go Brand Logo

Why Go?

- Open source
- Simple and clean syntax

Figure: Go Brand Logo

The Go Programming Language, Documentation - The Go Programming Language, Go Brand Logo

Why Go?

- Open source
- Simple and clean syntax
- Concurrency via `goroutines`

Figure: Go Brand Logo

The Go Programming Language, Documentation - The Go Programming Language, Go Brand Logo

Why Go?

- Open source
- Simple and clean syntax
- Concurrency via `goroutines`
- Auto-typing at variable declaration

Figure: Go Brand Logo

The Go Programming Language, Documentation - The Go Programming Language, Go Brand Logo

Why Go? (cont.)

- Fast compilation

Figure: Go Brand Logo

Documentation - The Go Programming Language,Go Brand Logo

Why Go? (cont.)

- Fast compilation
- Build-in garbage collection

Figure: Go Brand Logo

Documentation - The Go Programming Language,Go Brand Logo

Why Go? (cont.)

- Fast compilation
- Build-in garbage collection
- Big standard library

Figure: Go Brand Logo

Documentation - The Go Programming Language,Go Brand Logo

Why Go? (cont.)

- Fast compilation
- Build-in garbage collection
- Big standard library
- Many helper / Q.O.L. tools

Figure: Go Brand Logo

Documentation - The Go Programming Language,Go Brand Logo

Why Go? (cont.)

- Go Modules for dependencies.

Figure: Go Brand Logo

Documentation - The Go Programming Language

Why Go? (cont.)

- Go Modules for dependencies.
- comparable to `pip`, `cargo`, `npm`, etc.

Figure: Go Brand Logo

Documentation - The Go Programming Language

Why Go? (cont.)

- Go Modules for dependencies.
- comparable to `pip`, `cargo`, `npm`, etc.
- `go.mod`

Figure: Go Brand Logo

Documentation - The Go Programming Language

Go Syntax Example

Logging an `add()`-function implemented in Go

main.go

```
package main // package scope definition
import (
    "github.com/rs/zerolog" // using a third-party package
    "github.com/rs/zerolog/log"
)

func add(a, b int) int { // function implementation
    return a + b
}
func main() { // entry point
    n := 5 // variable declaration
    log.Println(add(n, 5)) // logging library call
}
```

OpenStreetMap

- Pick an area and export the data as an `.osm` file

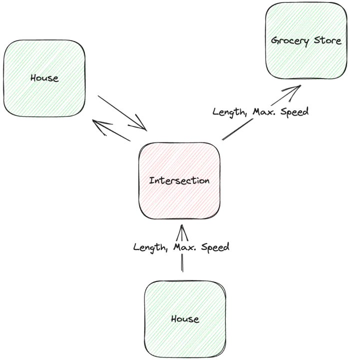

Figure: Example: Node-Edge Relationship

OpenStreetMap

- Pick an area and export the data as an `.osm` file
- Data includes:

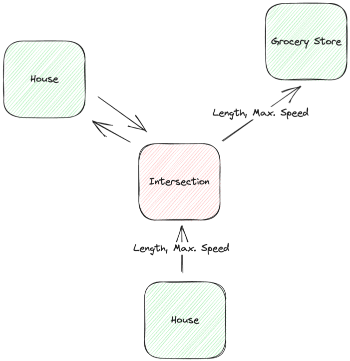

Figure: Example: Node-Edge Relationship

OpenStreetMap

- Pick an area and export the data as an `.osm` file
- Data includes:
  - Nodes - includes ID, Geo-coordinates
    - Object node (e.g. House, Store)
    - Intersection node

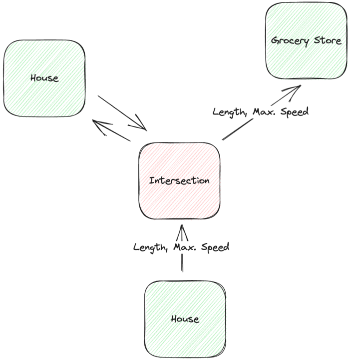

Figure: Example: Node-Edge Relationship

OpenStreetMap

- Pick an area and export the data as an `.osm` file
- Data includes:
  - Nodes - includes ID, Geo-coordinates
    - Object node (e.g. House, Store)
    - Intersection node
  - Edges
    - Def.: Has source and target node
    - Includes tags (max. speed, length, etc.)

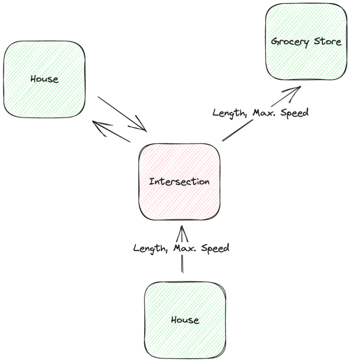

Figure: Example: Node-Edge Relationship

Preprocessing

- Using OSMnx Python package for OpenStreetMap
- It makes it easy to extract the data in a DataFrame similar format

u	v	key	osmid	oneway	lanes	name	highway	maxspeed	reversed	length	ref	geometry	access	service	area	width	bridge	landuse
28095800	**316420843**	**0**	25299426	True	2	Groner Landstraße	secondary	50	False	14.601	NaN	LINESTRING (9.92684 51.53318, 9.92667 51.53327)	NaN	NaN	NaN	NaN	NaN	NaN
28095826	**155062449**	**0**	15540548	True	2	Berliner Straße	primary_link	50	False	25.882	NaN	LINESTRING (9.92882 51.53627, 9.92886 51.53635...	NaN	NaN	NaN	NaN	NaN	NaN
28095837	**173163461**	**0**	28538211	True	NaN	Godehardstraße	tertiary	50	False	12.508	NaN	LINESTRING (9.93032 51.53718, 9.93043 51.53709)	NaN	NaN	NaN	NaN	NaN	NaN
28095839	**4029069313**	**0**	28538183	True	2	Berliner Straße	primary	50	False	62.761	B 3	LINESTRING (9.93092 51.53724, 9.93174 51.53749)	NaN	NaN	NaN	NaN	NaN	NaN
28095862	**28095866**	**0**	28665635	True	2	Weender Landstraße	tertiary	50	False	12.310	NaN	LINESTRING (9.93382 51.53838, 9.93391 51.53828)	NaN	NaN	NaN	NaN	NaN	NaN

Figure: Edge data example

OSMnx 1.5.1 documentation

Preprocessing (cont.)

- Map preprocessing includes:
  - Fully filtering object nodes
  - Removing irrelevant edges (e.g. cycleways)

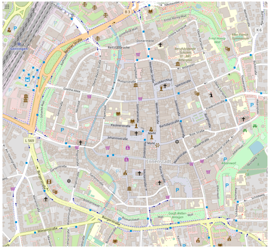

Figure: Raw OSM Map: Göttingen city centre

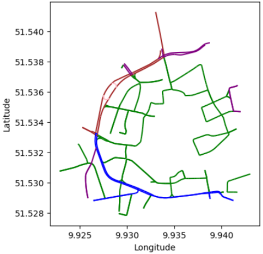

Figure: Processed map: Göttingen city centre

Redis

Figure: Redis Logo

- **RedisGraph** is part of the preprocessing

Sanfilipo, *Redis*, *OSMnx 1.5.1 documentation*, *RedisInsight | The Best Redis GUI*, *Graph | Redis Documentation Center*

Redis

Figure: Redis Logo

- **RedisGraph** is part of the preprocessing
- Uses the *Cypher* Syntax

Sanfilipo, *Redis*, *OSMnx 1.5.1 documentation*, *RedisInsight | The Best Redis GUI*, *Graph | Redis Documentation Center*

Redis

Figure: Redis Logo

- **RedisGraph** is part of the preprocessing
- Uses the *Cypher* Syntax
- Helps us to parse the data from OSMnx into a directed Graph-Database

Sanfilipo, *Redis*, *OSMnx 1.5.1 documentation*, *RedisInsight | The Best Redis GUI*, *Graph | Redis Documentation Center*

Redis

Figure: Redis Logo

- **RedisGraph** is part of the preprocessing
- Uses the *Cypher* Syntax
- Helps us to parse the data from OSMnx into a directed Graph-Database
- Has a GUI client (RedisInsight), which allows us to query for the needed data

Sanfilipo, *Redis*, *OSMnx 1.5.1 documentation*, *RedisInsight* | *The Best Redis GUI*, *Graph* | *Redis Documentation Center*

Implementation

- Initializes in-memory graph from RedisGraph

Implementation

- Initializes in-memory graph from RedisGraph
- Parses the incoming graph data into `Vertex` and `Edge` structs

Implementation

- Initializes in-memory graph from RedisGraph
- Parses the incoming graph data into `Vertex` and Edge structs
- Uses *Depth-First-Search* for path finding

Implementation

- Initializes in-memory graph from RedisGraph
- Parses the incoming graph data into `Vertex` and Edge structs
- Uses *Depth-First-Search* for path finding
- Omits realistic, microscopic driver model

Implementation - Vehicle

Vehicle Struct in Go

vehicle.go

```go
type Vehicle struct {
	ID                 string // Unique identifier
	Path               []int  // Vertex IDs
	DistanceTravelled  float64
	Speed              float64
	Graph              *graph.Graph[int, GVertex] // Parent Graph Reference
	IsParked           bool // Is done travelling
	PathLengths        []float64 // Edge lengths of the path
	PathLimit          float64 // Maximum distance of the Path
}
```

Mattfeld and Vetter, *Github - PCHPC - Vehicle*

Implementation - Graph Structs

graph.go

```go
type Edge struct {
	ID          int // Unique identifier
	Source      int // Source vertex ID
	Target      int // Desitination vertex ID
	Length      float64 // Length of the edge
	MaxSpeed    float64 // Speed Limit
	Data        EdgeProperties // Holds HashMap with current Vehicles
}

type GVertex struct {
	ID int // Unique identifier
	X  float64 // Longitude (GPS)
	Y  float64 // Latitude (GPS)
}
```

Mattfeld and Vetter, *Github - PCHPC - Graph*

Benchmarking the Sequential Implementation

- Benchmarking with following parameters
  - Number of vehicles: **100** (default)
  - Go-routines activated: **False** (default)
  - Randomized speed: $5,5\frac{m}{s} \leq v \leq 8,5\frac{m}{s}$, where v is velocity. (default)

Benchmarks - Sequential Without Goroutines

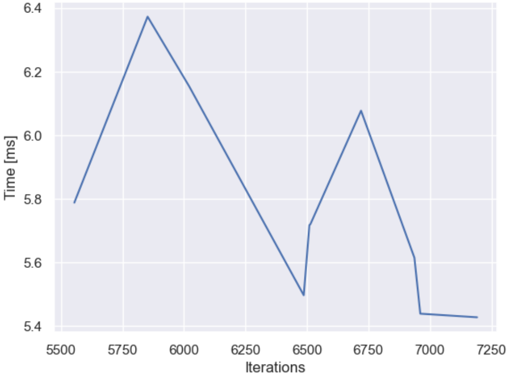

Figure: Sequential Benchmark

- Average Iteration: $6474,80$
- Average ms/op: $5,78$

Benchmarks - Sequential With Goroutines

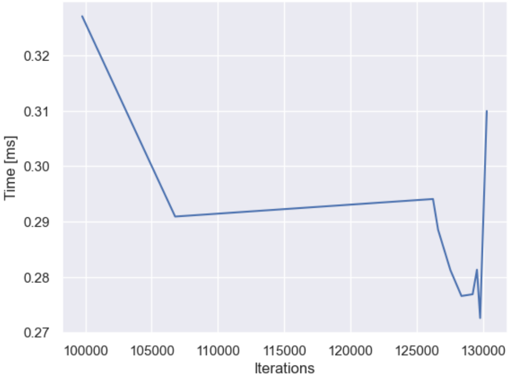

Figure: Sequential Benchmark with Goroutines

- Average Iteration: $123424,80$
- Average ms/op: $0,29$

Benchmarks - Sequential With Goroutines

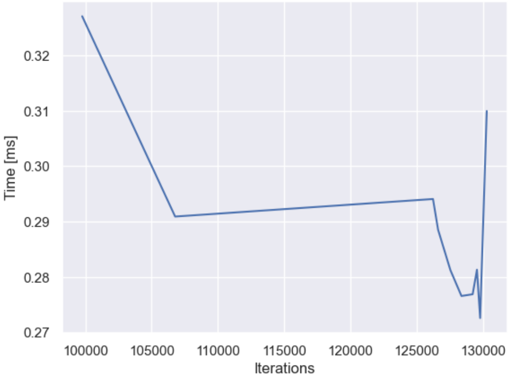

Figure: Sequential Benchmark with Goroutines

- Average Iteration: $123424, 80$
- **19,06x more** iterations
- Average ms/op: $0, 29$
- ca. **94.97%** faster

First implementation - Edge based partition

- **Goal**: Find a simple implementation, that can be distributed between ranks

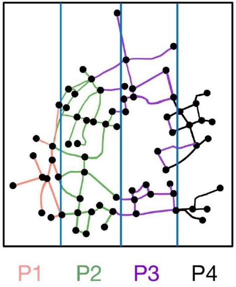

Figure: Example: Edge based partition

First implementation - Edge based partition

- **Goal**: Find a simple implementation, that can be distributed between ranks
- **Idea**:
  - Graph split induced by vertex GPS coordinates and # ranks
  - Each process has a subgraph that includes feature-complete edges

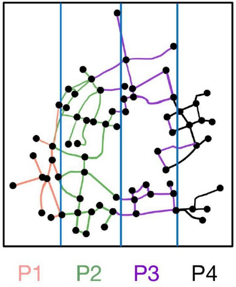

Figure: Example: Edge based partition

First implementation - Edge based partition (cont.)

- **Implementation:**
  - Edges and vertices must be serialized for MPI-IPC
  - Vehicle methods must be modified to switch ranks for MPI

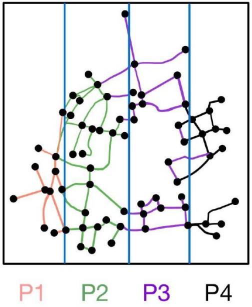

Figure: Example: Edge based partition

Second implementation - Path based partition

- **Goal**: Minimizing the communication between processes

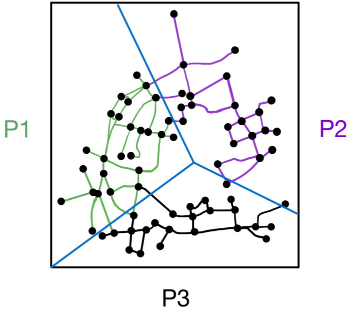

Figure: Example: Path based partition

Second implementation - Path based partition

- **Goal**: Minimizing the communication between processes
- **Idea**: Each process manages n nearest neighbours

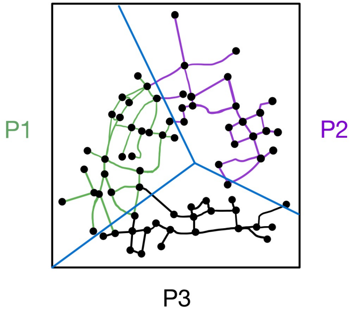

Figure: Example: Path based partition

Second implementation - Path based partition

- **Goal**: Minimizing the communication between processes
- **Idea**: Each process manages n nearest neighbours
- **Implementation**: Architecture not fully developed yet

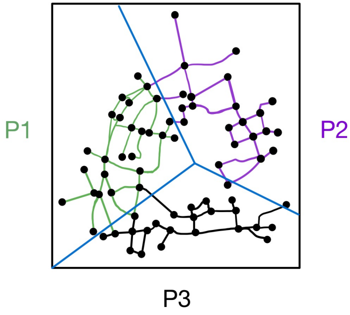

Figure: Example: Path based partition

Current Work

- **Focus:** Improvements on *Edge-based Partitioning* approach on MPI

Current Work

- **Focus:** Improvements on *Edge-based Partitioning* approach on MPI
- **Candidate:** Implementation of the *Path-based Partitioning* approach on MPI

Current Work

- **Focus:** Improvements on *Edge-based Partitioning* approach on MPI
- **Candidate:** Implementation of the *Path-based Partitioning* approach on MPI
- **In progress:** Simplification of the map input process; omitting Redis

Future Work

- Microscopic driver model implementation
- GUI Visualization
- Map input generalisation: *Add any coordinates from OpenStreetMaps*

References

Documentation - The Go Programming Language. en. URL: https://go.dev/doc/ (visited on 07/04/2023).
Go Brand Logo. en. URL: https://go.dev/blog/go-brand (visited on 07/10/2023).
Graph | Redis Documentation Center. URL: https://docs.redis.com/latest/stack/deprecated-features/graph/ (visited on 07/10/2023).
Mattfeld, Valerius and Bianca Vetter. *Github - PCHPC - Graph*. URL: https://github.com/valerius21/pchpc/blob/main/streets/redisInfo.go#L33-L44 (visited on 07/10/2023).
— .*Github - PCHPC - Vehicle*. URL: https://github.com/valerius21/pchpc/blob/c33d3de5f7d55ffb9e21bc3fec3b8ed0f04e2e26/streets/vehicle.go#L16-L25 (visited on 07/10/2023).
OpenStreetMap. en-GB. URL: https://www.openstreetmap.org/ (visited on 07/10/2023).
OSMnx 1.5.1 documentation. URL: https://osmnx.readthedocs.io/en/stable/ (visited on 07/10/2023).
RedisInsight | The Best Redis GUI. en. URL: https://redis.com/redis-enterprise/redis-insight/ (visited on 07/10/2023).
Sanfilipo, Salvatore. *Redis*. en. URL: https://redis.io/ (visited on 07/10/2023).
The Go Programming Language. original-date: 2014-08-19T04:33:40Z. July 2023. URL: https://github.com/golang/go (visited on 07/05/2023).