

Analysis tool targeting environmentally aware markets through DSM-auctioning framework

Grupp 9 - lp2 & lp3 2023-2024

Arvid From, arvidfrom93@gmail.com, arvfro-4
Oskar Svensson, sevosa-1@student.ltu.se, sevosa-1
Oliver Östrot, olistr-1@student.ltu.se, olistr-1
Daniel Sternelind, danset-1@student.ltu.se , danset-1

Summering

Rapporten behandlar utvecklingen av ett verktyg som kan underlätta strikt analys av utsläppen från organisationer när de ska transportera diverse produkter.

För att möjliggöra detta så byggdes verktyget över en tidigare implementerad simulation av ett DSM-auktionsbeteende. Det som tillades var till stor del i fördel för att systemet skulle reflektera mer verklighetstroga transportbeteenden.

Vid undersökning av det grundläggande systemet, så har en addition av riktiga distanser och platser anmärkts som extra användbart. Därifrån har en miljöanalys som baseras på hur varor levereras växt fram. Det färdiga verktyget ger möjlighet till en användare att testa ett leveransnät av valfri mängd städer för att se hur effektivt leveranser blir, både för miljö och rättvist utbyte, i addition till alla funktioner det gamla systemet hade i grunden.

Verktyget implementeras i form av en trestegslösning. I första hand ett verktyg som bygger upp ett effektivt leveransnät givet en lista av städer, med hjälp av Dijkstras algoritm.

I andra hand en ändring av det ursprungliga systemet där en abstraktion av platser och distanser har ersatts med verkliga sådana, och slutgiltigen ett GUI som gör arbetet av simulatören lättare att analysera och möjligtvis förbättra för framtida intressenter.

Syftet i utvecklingen av verktyget i sig har varit att underlätta för bolag att göra goda val för miljön, samt för bolagskontrollanter att enkelt kunna se när det inte görs. Vilket vi hoppas, i längden, kan hjälpa till att underlätta arbetet mot att nå klimatmålen.

Inledning

1.1 Bakgrund

Med mer miljömedveten allmän befolkning så skapas en ny sorts konsument, som vi inte tror är fullt studerad vad gäller dess generella beteende. Denna okunskap kan leda till många ledningsmisstag för diverse bolag där de antingen tappar kunder eller inte tar till sig sin optimala kundbas när attityder förändras.

Dessa fenomen leder till att söka nya marknader med geografisk omplacering av produktion eller leverans-nav. Vi anser att det finns ett tydligt behov för verktyg som kan ta sådana situationer i beaktning och få en bra överblickande vy av hur klimatkänslighet påverkar den generella kundbasen.

Grundmålet med utvecklingen av detta verktyg blev således att skapa ett verktyg som lätt kan iaktta dessa fenomen i en öppen marknad och återge värdefull information för marknadsgranskare.

Olika mjukvara med mål att simulera system som iakttar miljöbehov mellan konsumenter, bolag och regeringar har designats och implementerats åtminstone sedan ETS startades 2005. ETS (emissions trading systems [1]) är ett av EUs främsta verktyg idag för att motarbeta klimathotet. Organisationen gör detta genom att begränsa mängden CO₂ ett givet bolag får släppa ut, samtidigt som en möjlighet att köpa till sig traktamenten vars kostnad sedan ska kunna hjälpa mot klimatkrisen på andra vis.[2]

Diverse organisationer har arbetat med denna modell och tagit fram sina egna simulationsverktyg som kan återfinnas idag i både den agrikulturella industrin och allmän handel[3]. En produkt som liknar Projektet som ska beskrivas i denna rapport är Carbon Sim[4] vilket är en simulator byggt av den globala non-profiten EDF (Environmental Defense Fund [5]), som simulerar mängden utsläpp som bolag gör i mål att besvara frågan: "Vilken effekt har begränsningar på utsläpp och klimatmålsdeadlines på klimat och handel?". Precis som detta projekt så har Carbon Sim som fokus att uppskatta effekten av eventuella rörande delar i handelsvärlden i förhållande till klimat och informera samtliga berörda aktörer.

Vid en hastig undersökning på tidigare projekt så hittades ingen direkt parallell med verktyget senare beskrivet i denna rapport men det är ett rimligt antagande att liknande simuleringsmodeller redan finns utvecklande någonstans i världen. Då marknadsundersökning på alternativ som är mer vänliga mot klimatet rimligen har skett under en lång tid och varit fokus i många delar av industrin i takt med att klimatmedvetenhet har blivit en mer pressande fråga.

Vårt specifika arbete bygger på tidigare arbeten. Huvudsakligen ett projekt tidigare skrivet i Python Github användaren Elliotpk [6 som står för DSM-beteendet]. Verktyget använder sig av den grova majoriteten av det arbetet, med undantag för kundbeteenden och automatisering. I addition

använder sig verktyget av andra API:er och moduler, till exempel HTML, Python flask, MongoDB och Google Maps Directions API. Vi använder oss också av en publik csv-fil som ska innehålla alla världens städer taget av world cities database [7].

Konceptuellt så bygger projektet på Raj Jain's fairness index, Dijkstra's algorithm. och ett generellt antagande att längre transporter innebär värre klimatavtryck.

Projektet är skrivet med Python i grund, men använder sig av Javascript delvis i implementationen av GUI:n.

1.2 Problembeskrivning

Problemet detta projekt ämnar att lösa är att företag och organisationer inte alltid vet vad de bästa alternativen är när det kommer till att utföra sina ämbeten miljövänligt. Den specifika metoden som ska kunna lösas med verktyget är avvägningen över var det kan vara rimligt att bygga en fabrik eller intermediär punkt där leveranser kan ske från företaget till kund.

Specifikt i förhållande till hur mycket mer eller mindre utsläpp som kommer göras via biltransport ifall en sådan punkt skapas.

I och med denna lösning så ska en ämbetsman också kunna utföra en analys av beslut som görs av bolag och se om det faktiskt uppfyller ett teoretiskt klimatmål.

Syftet med projektet är att skapa ett analytiskt verktyg som ska avläsa och utifrån olika kriterier som går att kalibrera, visa på hur uppbyggnaden av ett leveransnätverk kan påverka utsläpp, samt rättvis handel för konsumenten.

Vi anser att ett verktyg för simulation är fördelaktigt för just det här sortens problem då man behöver ett helhetsperspektiv över marknaden för att kunna avgöra viktiga slutresultat som strukturen producerar. För att uppnå detta perspektiv kan man angripa problemet antingen med direkt analys av verkligheten eller en simulation. Och en simulation kan mycket bättre behandla hur saker skulle kunna se ut givet många ändringar snarare än en direkt analys, som snarare är fördelaktigt när det behövs en korrekt representation av det nuvarande tillståndet.

Detta verktyg ska kunna avläsa distans från en resa från startpunkt utanför leveransnätet, genom ett effektivt strukturerat leveransnät med noder som kan conceptualiseras som postkontor eller varuhus, sen till en slutpunkt utanför nätet. Det ska också gå att analysera miljöaspekter av den nuvarande simulationsmiljön och leveransnätverket för att avgöra om det nätverket ger goda utfall för miljön eller ej. Raj Jain's Fairness Index är också en relevant faktor som räknas in och som kan kunna ställas emot miljövänlighet.

1.3 Uppgift och mål

I det underliggande projektet som simulerar ett generellt DSM auktionsbeteende nämnt i bakgrunden så ska en miljökostnadsberäkning läggas till. En generell hanterare för uppbyggnad av ett leveransnätverk ska implementeras. I addition så ska grundattributen av köpare ändras för att ta miljö i beaktning. Systemet ska också ha en visuell återgivning för att lätt kunna analysera resultaten.

För att lyckas lösa ovanstående problem ska olika mål, via en minimal viable product, sättas ihop, detta innebär några minimala mål. Vi kan därefter utöka produkten med mer funktionalitet given möjlighet. Allt detta beror på hur mycket tid kvar det blir på slutet av kursen för eventuella tillägg.

För Projektets MVP sattes målen:

- Simulation som använder sig av APIs för att avgöra klimativänlig distans mellan köpare och leverantör.
- Grafisk Användargränssnitt som visar kartdatan för användaren.
- Visar minst en rutt per köpare
- Olika köpare i olika platser
- Ska fungera i både Europa och Amerika med en rutt för havsresa mellan dem.

Detta är våra minimala krav, men det finns även andra stretch goals som kan uppfyllas om ovanstående krav blir genomförda tidigt.

Några exempel på stretch goals som kan genomföras är:

- Flera havsrutter mellan Nordamerika och Europa och eventuellt andra världsdelar
- Multipla möjliga färdrutter återgivna samtidigt i den grafiska återgivningen
- Olika Bränslekällor
- Olika färdsätt

När projektet är klart så ska man kunna ange preferenser och platser för köpare och leverantörer. Vid körning av programmet ska också en grafisk återgivning ges för vilken rutt ett eventuellt paket tog, vilken kostnad som kändes mest rättvis för köpare givet klimatavtrycket, och Raj Jains Fairness index för samma utbyte

2. System Design

I och med att diagrammen som beskriver systemet är för stora för att lätt kunna läsas bland text så kan de återfinnas i rapportens appendix.

UML Diagram

Detta diagram kan återfinnas i figur A.1.

Systemdesignen för p har för det mesta baserats på att strukturen ärvs av verktygen detta verktyg är byggt på. Det föregående auktionssystemet har många beroenden mellan alla delar av programmet. På grund av detta så har vi tyvärr behövt bygga på våra funktionaliteter utanför och veva in dem i det tidigare nätet.

Den centrala pelaren i den tidigare strukturen var refCalc, eftersom att alla filer skickade vidare sin data dit för beräkning. För att hålla ordning har vi implementerat GUI och databaser helt utanför det tidigare systemet, men de har också behövt skickas vidare till refCalc för att kunna hanteras rätt inom det existerande systemet.

SekvensDiagram

Dessa diagram kan återfinnas i figur A.2

Det finns två huvudsakliga processer som sker under programmets gång. Den som kör simulationen i fråga med auktionssystem, miljöberäkningar och sorteringar och den som bygger databaserna med verkliga platser och rutter. Dessa är separata processer och har därför illustrerats i två olika diagram.

Simulationsdelen centraliseras runt arbetet som refCalc gör. refCalc kallar på funktioner inom funktioner för att dra ut ruttinformation från databasen. Resultaten skickas senare vidare till main, som sedan kallar på GUI:n med sina resultat.

I och med att refCalc inte är medveten om vad som är den optimala auktionen, så måste dock GUI:n referera tillbaka till databasen med de slutgiltiga platserna för att se vilken rutt den ska ta mellan städer i ritningen.

Databasdelen, tar tre manuellt framställda filer, kör en master funktion som bygger upp många csv filer till en början som sen till slut summeras med en fil som illustrerar rutter tagna från varje stad i nätverket till varje annan stad `shortest_paths.csv`.

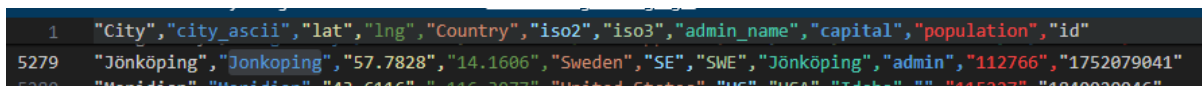
3. Implementation

Innan programmet ska kunna köras i sin helhet ska en mängd paket installeras och en del mjukvara. Programmet körs i Python 3.12. Så se till att ha det installerat på datorn som kör programmen. Mer information om paketen samt annat som behövs hittas i README-filen på [github](#)[9]

Det finns 3 delar till att köra programmet, så vi kommer gå igenom dem i ordning här:

3.1.1 Databasbygge

Databasbyggaren gör ett rutnät som leveranser kan skickas igenom. Det kan vara intressant att testa olika kombinationer av städer och för detta behöver du skriva in i några databasfiler. Det första som behöver göras är att bestämma vilka städer man är intresserad av att ha i sitt nätverk, och sätta in dem i filen varuhus.csv. Notera att städer med samma land måste vara i direkt kontakt med varandra, och vara skrivna i samma ASCII format som städerna i databasfilen worldcities.csv.



```
1 "City","city_ascii","lat","lng","Country","iso2","iso3","admin_name","capital","population","id"
5279 "Jönköping","Jonköping","59.7828","14.1606","Sweden","SE","SWE","Jönköping","admin","112766","1752079041"
5280 "Meridiano","Meridiano","42.6416","-116.2077","United States","US","USA","Idaho","","115227","1840020046"
```

Figur 3.1 Hur en stad med icke ASCII karaktärer ser ut i worldcities.csv



```
412 Jonköping,Sweden,Europe
413 Norrköping,Sweden,Europe
414 Lulea,Sweden,Europe
415 Pitea,Sweden,Europe
416 Umea,Sweden,Europe
417 Sundsvall,Sweden,Europe
418 Zurich,Switzerland,Europe
419 Geneva,Switzerland,Europe
420 Basel,Switzerland,Europe
421 Lausanne,Switzerland,Europe
422 Bern,Switzerland,Europe
```

Figur 3.2: hur städer ska se ut i varuhus.csv, notera avsaknaden av icke ASCII karaktärer och grupperingen av länderna

Sedan måste Neighbours.csv konfigureras för hand. Den här filen ska innehålla alla berörda länder i din simulation, skriven på engelska tillsammans med de ländernas grannar.


```

1 Country,Neighboring Countries
2 Albania,Greece,Macedonia,Montenegro,Serbia
3 Andorra,France,Spain
4 Armenia,Georgia,Turkey
5 Austria,Czech Republic,Germany,Hungary,Italy,Liechtenstein,Slovakia,Slovenia,Switzerland
6 Azerbaijan,Armenia,Georgia,Russia
7 Belarus,Latvia,Lithuania,Poland,Russia,Ukraine
8 Belgium,France,Germany,Luxembourg,Netherlands
9 Bosnia and Herzegovina,Croatia,Montenegro,Serbia

```

Figur 3.3: exempel på hur rader kan se ut i neighbours.csv

Efter att det är fixat så är det bara att köra Database_Master.py Från scopet av mappen Src. så ska du kunna få ut ett nätverk som visas i filen shortest_paths.csv.

```

Stockholm,Neisingsborg,Stockholm -> Norrköping -> Linköping -> Jönköping -> Gothenburg -> Neisingsborg,6647348
Stockholm,Malmö,Stockholm -> Norrköping -> Linköping -> Jönköping -> Gothenburg -> Malmö,657.7484308416372
Stockholm,Fredrikstad,Stockholm -> Norrköping -> Linköping -> Jönköping -> Gothenburg -> Fredrikstad,591.855

```

Figur 3.4 exempel på en koppling mellan två städer i nätverket shortest_paths.csv

Notera att körningen kan ta lång tid då simulationen blir mer intressant då det körs med en större mängd städer och big O notationen för average case av database_master.py är flera gånger större än n^2 . Ifall du är intresserad av att undersöka metoderna som bygger databaserna så finns dem i Src/Database och databasfilerna som skapas finns i Src/Database/Network_Database

3.1.2 DSM simulation

Själva auktionssystemet kontrolleras huvudsakligen från filen config.yaml. Där kan man ställa in hur många köpare och säljare man vill ha, hur många paket man vill att varje säljare ska ha till salu samt hur tillgång och efterfrågan ser ut. Max distans som en individ är villig för en leverans att åka, mängden simultana auktioner, avslutningsvillkor för auktioner och om miljövänlighet eller Raj jain's fairness index ska prioriteras och till vilken grad i sorteringen av bästa resultat.

I main så finns det en funktion som kan sortera ut bästa resultat efter antingen fairness, miljövänlighet eller din skräddarsydda preferens för hur hårt Fairness eller miljövänlighet ska tynga på sorteringen. Vilken sortering man vill ha kan man bestämma på rad 314 i main.py

```

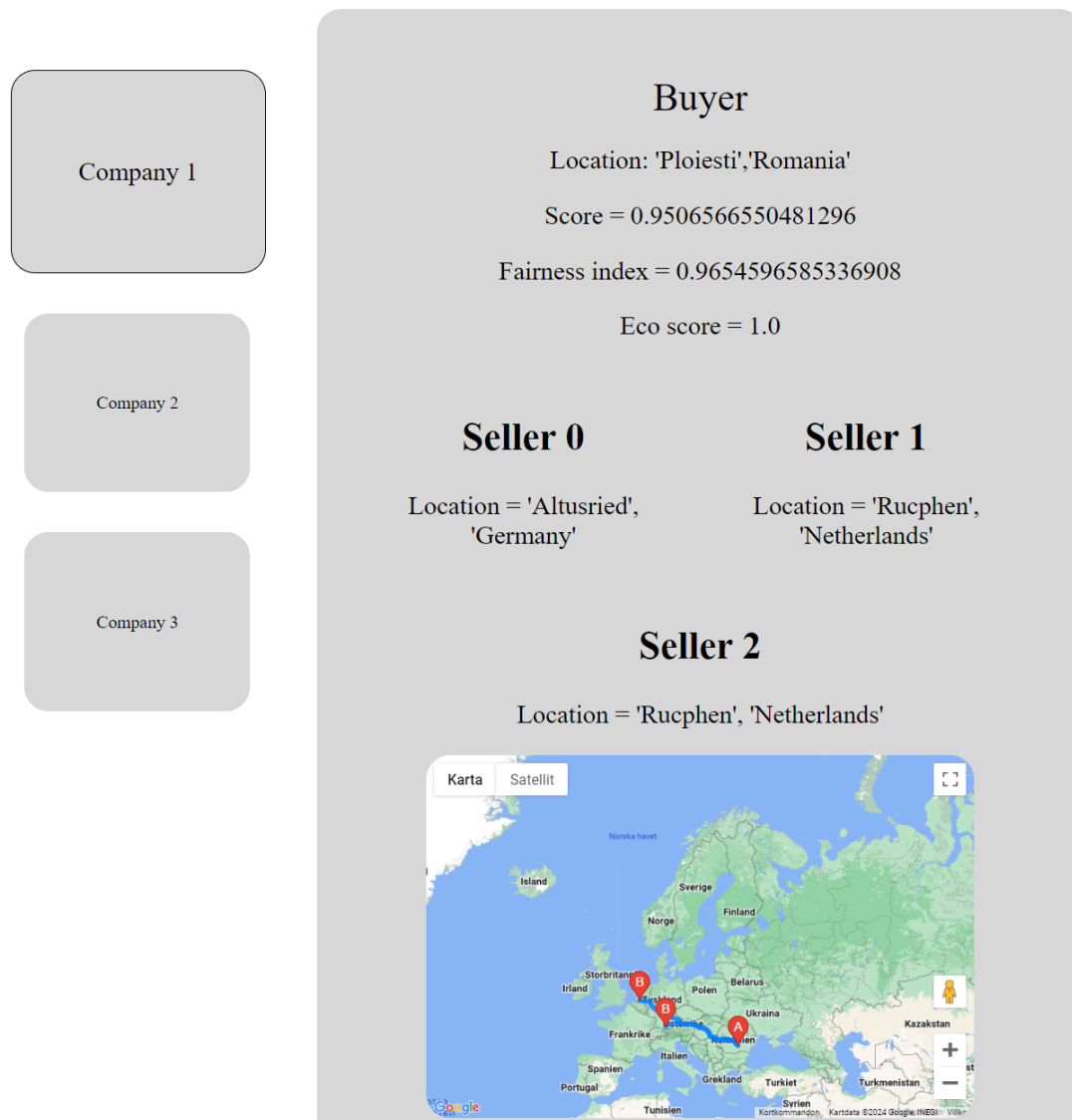
309 def start(skipPrompts):
310     'Master function'
311
312     slotSize, endThreshold, sellerList, bidderList = readConfig(skipPrompts)
313
314     sortingMode =2          #modes choose what to sort by 1 is fairness, 2 is score, 3 is average distance

```

Figur 3.5: här kan du bestämma vad man sorterar efter för bästa utfall

3.1.3 Front-End

Sedan är det front-end vilket är hemsidan som visar resultaten. På hemsidan går det att se resultatet av simuleringen så att Score, Fairness Score och Eco Score samt lokalisering av säljare och köpare visas upp genom en google maps där leveransvägen ritas ut mellan köpare och säljare. Detta kontrolleras huvudsakligen från GUI.py som tar ut alla resultat och information från mongoddb, som sedan skickas till en hemsida som python flask sätter upp med hjälp av html-filerna som ligger i Src/static/templates mappen. Filen GUI.py räknar även ut den kortaste ruten mellan en köpare och en säljare genom att använda vår databas shorest_paths. Hemsidan dyker inte upp direkt när programmet körs detta händer eftersom hemsidan inte laddas för ens efter main har gjort klart alla andra beräkningar.



Figur 3.6: Exempel på hur hemsidan ser ut efter körning

4. Resultat

4.1 Relation till ursprungliga uppgifter och mål

Som tidigare beskrivits i punkt 1.3 "Uppgift och mål", så var målet till en början att API:s skulle användas för att beräkna en mer korrekt distans mellan köpare och säljare. Men detta blev senare ändrat till en formel för beräkning av distans som använde sig av latitud och longitud för att estimerar distanser mellan auktionen och köparens plats.

Grafiskt användargränssnitt blev implementerat och uppfyllde vårt ursprungliga mål med kartdata för minst en rutt per köpare, och olika köpare på olika platser.

4.2 Uppdaterad designbeskrivning

Nuvarande version av programmet tar hjälp av några databaser som innehåller städer i varje land i Europa, ländernas grannländer, största städerna i varje land med mera. Denna information använder programmet för att beräkna den bästa möjliga ruten för leverans från en startpunkt till en slutpunkt, baserat på ett nät som bildas under körningen.

Detta rutnät tar upp till ett visst antal kopplingar till närliggande städer, och skapar rutter för hur leverans genom detta nät skulle se ut. Simulationen beräknar sedan miljömässigt, distans och vad som är mest rättvist för köparna för varje rutt.

Dessa rutter blir sedan sorterade beroende på vilken rutt som är antingen mest miljövänlig, mest rättvis eller har kortast medeldistans. Sedan skickas denna information vidare till databasen MongoDB. Denna data tar sedan användargränssnittet och visar upp i form av värden på miljövänlighet, rättvishet, och medeldistans samt visar upp rutterna på en karta.

4.3 Realisering

Programmet är skrivet i programmeringsspråket Python och använder versionen Python 3.12.0. För att förenkla körningen och implementationen av olika delar användes följande paket / bibliotek:

- Pymongo
- Pyyaml
- python flask
- Networkx
- Pandas
- Requests

Det krävs även en API nyckel för att komma åt följande:

- Directions API
- Distance Matrix API
- MAPS JavaScript API

En installation av MongoDB på datorn och MongoDB Compass för översikt av resultatet kan också behövas. Detta inkluderar att skapa en egen lokal host.

4.4 Utvecklingsgrad

I stort sett fungerar programmet som det ska, men det finns ytterligare arbete att göra för att projektet ska vara fullt funktionellt i samtliga fall.

Potentiella utvecklingspunkter är till exempel att beräkning av distans i backend inte använder sig av en API. Det skulle också kunna implementeras en annan nätverksbyggare för större nätverk för att minska tiden som programmet behöver köras.

När det kommer till GUI skulle kartritaren kunna utvecklas så att rutten som ritas är från säljarens varuhus till köparnas varuhus igenom kortaste distansen istället för att ta bilvägen från säljare till köpare.

4.5 Potentiell utveckling

Det finns ett par funktioner vi inte ännu implementerat som nästkommande grupper skulle kunna implementera. En förbättring hade varit att simulationen skulle fungera utanför Europa, vilket var ett ursprungligt mål.

En annan förbättring hade varit att använda sig av en API i backend för att beräkna distanserna mellan köpare och säljare för att få en mer korrekt distans.

Det finns också förbättringar för databas och inställningar, där centralisering av inställningar och nätverksbyggande implementeras.

Simulering på olika färdssätt och bränslekällor är också en potentiell förbättring, där även rutter över havet kan implementeras.

Sedan finns det också mer generella förbättringar som till exempel att få koden att köras snabbare.

5. Slutsatser

Trots att målet för projektet ändrades mellan sprint 1 och 2, så har det skapats en bra grund för att bygga vidare på. I projektets början var målet att enbart implementera riktiga platser och inkludera fler kontinenter men vid sprint 2 ändrades det till att jobba inom Europa och att skapa ett leveransnät med hjälp av varuhus (lite som företag som postnord som skickar sina varor mellan lager innan de når sitt mål). Detta gjorde att det tidigare arbetet fick backas lite för att dra projektet mot sin nya väg. Dock även med den korta tiden av en sprint så har majoriteten av de nya funktionerna implementerats och plats för vidare arbete öppnats upp.

Vi har, som nämnt tidigare i rapporten, tagit hjälp av databasfiler på Europas städer/länder och skapat nya csv-filer som innehåller rutter för det så kallade leveransnätet. Programmet tar städer från vår databas, tilldelar dessa till köparna och säljarna i auktionsdelen av programmet och sedan tar GUI:n in informationen för att enklare representera den.

För att jobba vidare på detta projekt så skulle vi rekommendera en mer strukturerad planering från start för att förenkla arbetet (ett tydligare mål gör det lättare att ta sig dit). Denna nya del av projektet är långt från perfekt och kan delas upp i delar som behöver fixas, allt från att göra rutterna mer optimerade för mer miljövänliga leveranser till att göra frontEnd bättre så att den kan visa datan på ett snyggt sätt.

Då projektet tog en ny riktning så hamnade GUI:n på efterkälken eftersom att majoriteten av dess tidigare arbete behövde göras på ett annat sätt. I och med det så finns det en del att jobba på där, huvudsakligen att få rutter att ritas på kartan. Leveransrutterna kan även behöva en del arbete då distanser den jobbar med baseras på en formel som ger fågelvägen i princip, och därav skapas inte verklighetstroga vägbaserade rutter, vilket var ett av projektets ursprungliga mål. I vissa fall kopplas några städer enbart till varandra och skapar ett kluster vilket kan göra att ingen rutt finns till dessa städer till städer utanför klustret och därav inte går att jobba med heller.

Dock trots förhinder så har vi, som det nämns tidigare, byggt på Elliots DSMsim [6] och de nya funktionerna är att varje "person" i simuleringen utgår från en riktig plats i Europa, att leveranser sker via varuhus som i sin tur gör att den har ett leveransnät för vart saker ska skickas. Därefter visualiseras det med en GUI som visar rutt samt information om dealen.

6. Referenser

- [1] https://climate.ec.europa.eu/eu-action/eu-emissions-trading-system-eu-ets_en
- [2] https://climate.ec.europa.eu/eu-action/eu-emissions-trading-system-eu-ets/what-eu-ets_en
- [3] <https://openknowledge.worldbank.org/server/api/core/bitstreams/ed9e859f-663d-5b24-bd8de79746d88045/content>
- [4] <https://www.edf.org/sites/default/files/content/brochure.pdf>
- [5] <https://www.edf.org/>
- [6] <https://github.com/elliotpik/DSMSim>
- [8] <https://simplemaps.com/data/world-cities>

7. Appendix

Länkar till sprintplanering:

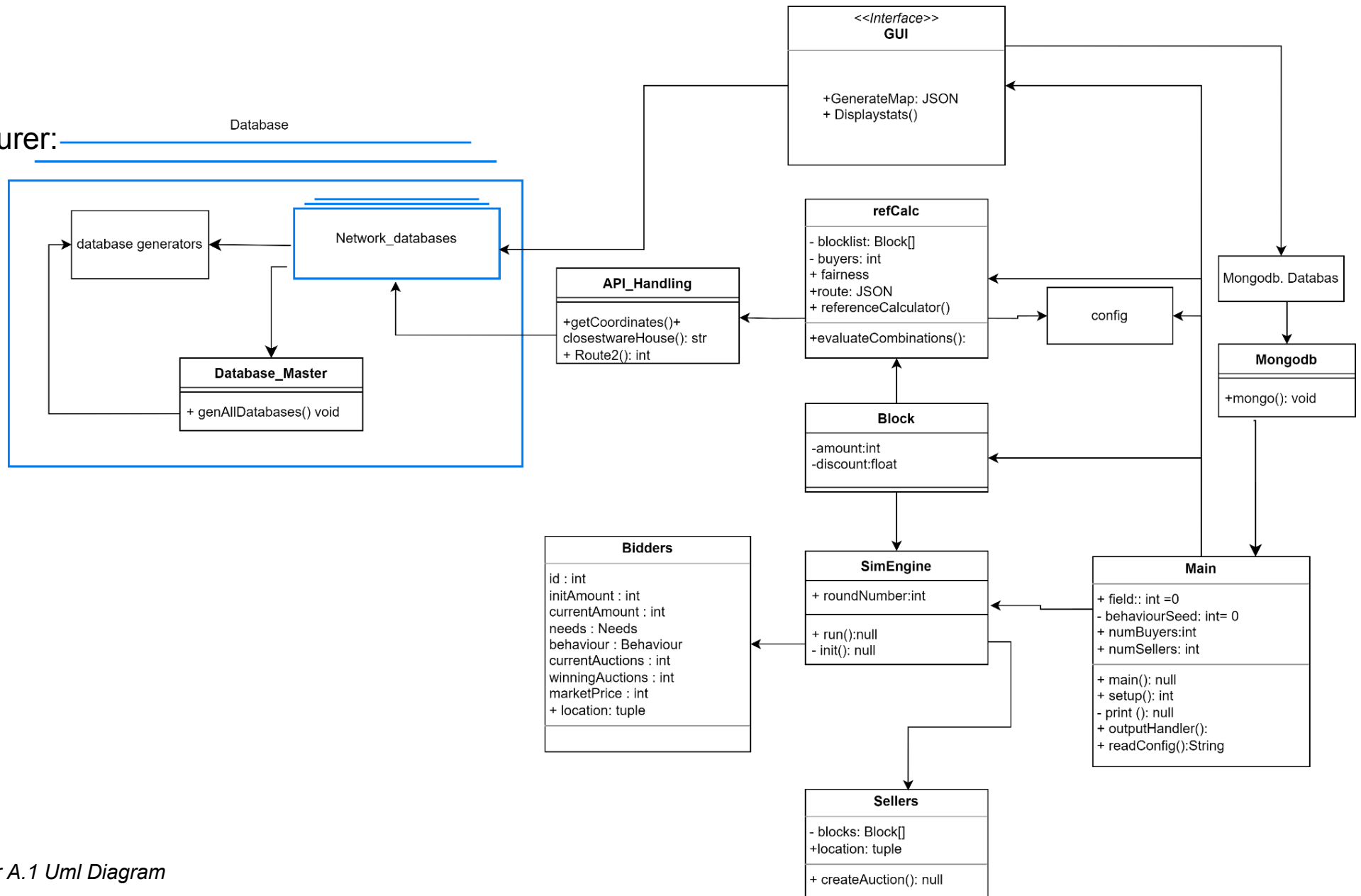
https://docs.google.com/document/d/1HA7kEzn_z3OKa76jQjEOLhCYKMDm5USpi3GxBYRp8gg/

<https://docs.google.com/document/d/1kcOmikYqxbYH1UqSKFjA7aRgKZbTnhmv2cqEgKzuPt8/>

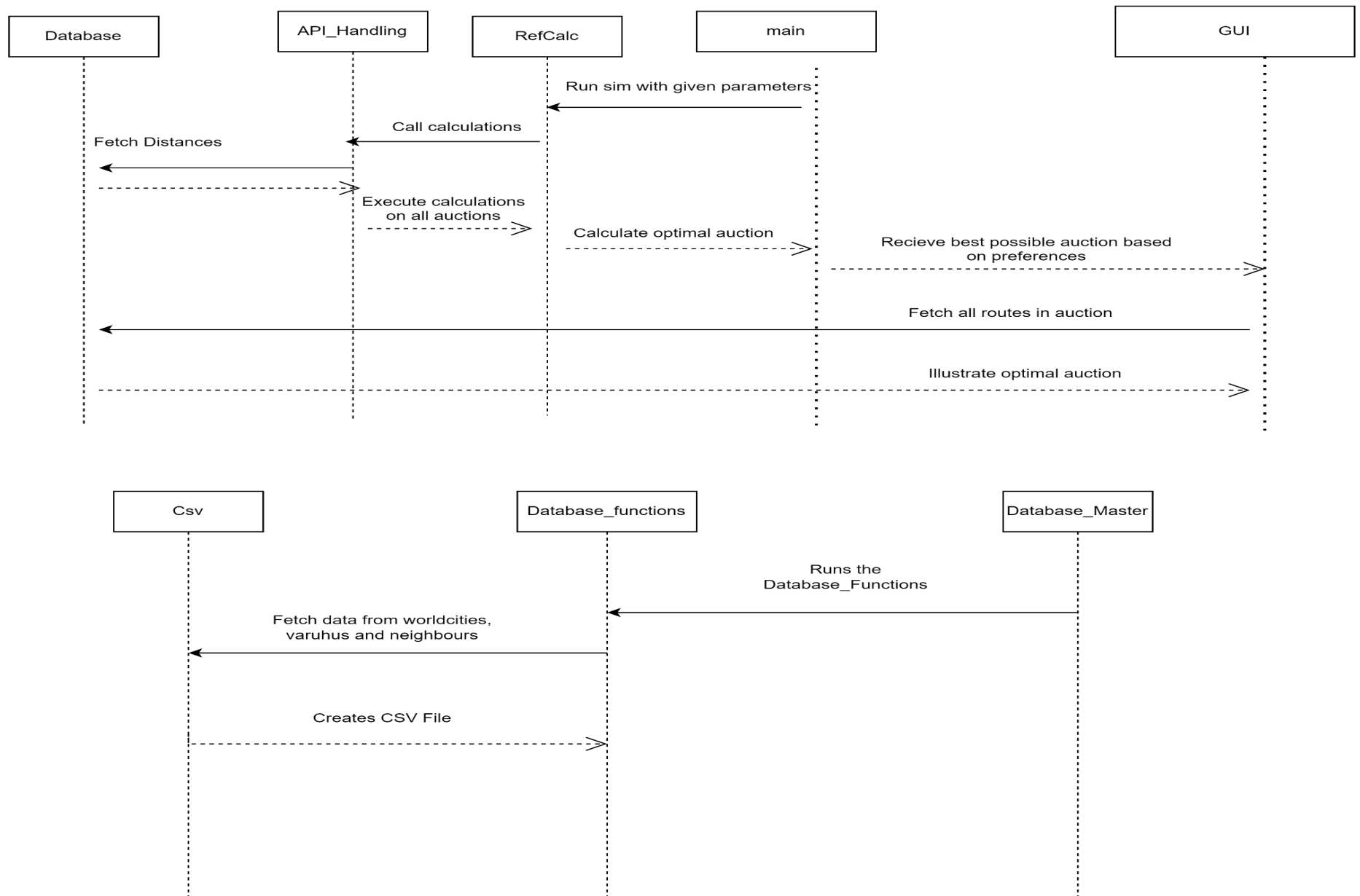
Länk till projekt i Github:

<https://github.com/arvfroLTU/Grupp9/tree/master>

Figurer:



Figur A.1 Uml Diagram



Figur A.2 Sekvensdiagram