PRESENTING

INTERSHIP'S REPORT

Auctions and General auction players

SUPERVISED BY **PERRUSSEL LAURENT** AND **MUNYQUE MITTELMANN**

2021/2022

Ikrame GHAFRANE

ACKNOWLEDGEMENTS

This internship would not have been what it was without the presence and support of a large number of people. I would like to express my sincere and heartfelt thanks to each one of them.

Starting with professor **Perrussel Laurent,** PhD student **Munyque Mittelmann** and professor **Sylvain Bouveret** to have accepted me within their team to work on the **AGAPE**[1]project, to have offered me a valuable experience in a research laboratory and to give a preview of how interesting a career in research is. I am also very greatful to have been supported and guided throughout the whole internship, either with relevant remarks and encouragements, or just given hints to apply certain notions already learned in our MIAGE bachelor year and/or go above and beyond and figure it out ourselves which would eventually only lead to developping our way of work.

I should also mention that it's been really nice working with my collegue, a fellow MIAGE student, **Raphaêl Dejus**. Teaming up with him made the journey a whole lot easier ; helping one another whenever there was ever a dead-end that we had to overcome to move on to the next task, sharing little moments of despair and success. Overall a great team mate.

Last but not least, words cannot describe how grateful i am to have had such an immense support from my best friends and my family even miles and miles away. It meant a lot . Thank you.

[1]**AGAPE** An Auction LanGuage for GenerAl Auction PlayErs

TABLE OF CONTENTS

Abstract

Auction-based markets are widely used for automated business transactions and AGAPE is a project investigating strategic reasoning behind these auctions by establishing a connection between rationality and actions taken by agents all while taking into consideration if the information that the agent has is either perfect or imperfect, and the type of strategy used. The goal is to also shed light into what makes a good bidding strategy.

Introduction

During a 4 month internship working as a research intern at the IRIT Lab (Institut de Recherche en Informatique de Toulouse) my tasks consisted of, **first**, learn all about auctions ; more specifically combinatorial auctions, all the possible bidding strategies and the Game-theoretic models, only to, **second** and foremost, learn how to develop automatic agents willing to participate in the auction and implement said strategies to connect with the previously implemented auction server, and by doing so, we will have means to justify which strategy was better for each agent and to perform optimal actions in future participation based on either perfect or imperfect informations which would eventually help establish a game theory and have a logical game without having the notion of "chance" play as a factor. And **third**, create a web server that would improve and simplify the process of holding and organizing these competitions.

IRIT's Presentation

Created in 1990, The Institut de Recherche en Informatique de Toulouse (IRIT) is a Joint Research Unit of the Centre National de la Recherche Scientifique (CNRS), consisting of : Université Toulouse1 Capitole (UT1), Université de Toulouse Jean Jaurès (UT2J), Université Paul Sabatier Toulouse 3 (UT3) and Institut National Polytechnique de Toulouse (INP).

IRIT is one of the largest UMR (Unité Mixte de Recherche) at the national level, is one of the pillars of research in Occitanie region with its 600 members, permanent and non-permanent, and about 100 external collaborators. Due to its multi-tutorial nature (CNRS, Toulouse Universities), its scientific impact and its interactions with other fields, the laboratory constitutes one of the structuring forces of the IT landscape and its applications in the digital world, both at regional and national level.

IRIT has focused its research on five major scientific issues and six strategic application areas.

- Health, Autonomy, Living, Well-being
- Smart City
- Aerospace and Transportation
- Social Media, Digital Social Ecosystems
- e-Education for learning and teaching
- Heritage and People Safety

As well as strategic action:

- Scientific Computing
- Big Data
- AI

The 24 research groups of the laboratory are dispatched in seven scientific departments:

Dpt ASR : Architecture, Systems, Networks (5 teams)
Dpt CISO : HPC, Simulation, Optimization (2 teams)
Dpt FSL : Reliability of Systems and Software (4 teams)
Dpt GD : Data Management (3 teams)
Dpt ICI : Interaction, Collective Intelligence (3 teams)
Dpt IA : Artificial Intelligence (3 teams)
Dpt SI : Signals, Images (4 teams)

Overview of The work undertaken

I- Learning All about auctions

What is an auction ?

Auction is an process of buying and selling goods or services by offering them up for bid, taking bids, and then selling the item to the highest bidder.

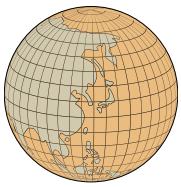
The bidders, in turn, create their own market, determining on an individual basis how much they want to pay for an item, rather than having prices dictated by the seller.

Perhaps the most important use of auctions has been to facilitate the transfer of assets from public to private hands

History of auctions

Auctions have been used since antiquity as early as 500 b.c. for the sale of a variety of objects as well as humans according to ancient Greek scribes.
The Romans also used the auction to liquidate their own property.

The most outstanding auction occurred in the year 193 C.E. when the entire Roman Empire was put on the auction block by the Praetorian Guard, which lead to a brief civil war and ended up in beheading the highest bidder.



Modern day auction vs Traditional auction

Traditional Auction

- Geographical limitation,
- Physical presence obligation,
- Limited to collectors who are ready to pay extra or more then the market price,
- Time consumption
- Keeping watch on the media for new auctions

Modern Auction

- Agents acting on behalf of Auction houses
- Providing a global platform for bidder by getting it to masses
- Any interested parties can put their goods for auction

Types of Auctions

↗ English auction : The auctioneer begins by calling out a low price and raises it as long as there are at least two interested bidders. The auction stops when there is only one interested bidder. The price rises continuously and bidders indicates their interest in the good by raising their hand. Once a bidder finds the price to be too high, they express their lack of interest by lowering their hand.

↘ Dutch auction : The auctioneer begins by calling out a price high enough so that presumably no bidder is interested in buying the object at that price. This price is gradually lowered until a bidder indicates their interest in the good and gets sold to them at the given price.

✉ The sealed-bid first-price : A bidder submit their bids in sealed envelopes; the person submitting the highest bid wins the object and pays what they bid.

✉→✉ Sealed-bid second-price : Same as concept as The sealed-bid first-price but what the highest bidder pays is what the second highest bid is.

Game-theoretic models

A game-theoretic auction model is a mathematical game represented by a set of players, a set of actions (strategies) available to each player, and a payoff vector corresponding to each combination of strategies.
Game-theoretic models of auctions and strategic bidding generally fall into either of the following two categories:

- In common value auctions, bidders compete for an item that has the value of the object is known for everyone. Typically the item value is uncertain and bidders base their decisions on estimates of the true value ;
- This stands in contrast to a private value bidders know their own value for the commodity with certainty but are unsure about others' valuations ;
- Affiliated values model, in which the bidder's total utility depends on both their individual private percpective and some unknown common value.

What is a combinatorial auction

Combinatorial auctions are auctions in which bidders bid on bundles, instead of on individual items in hopes to bid on the most beneficial of bundles.

In a combinatorial auction, two or more auctioned items can be complementary to each other when value of a bundle of the two items, g({i,j}, is higher than sum of their individual values. and substituble if their sum is lower.

BIDDING STRATEGIES

Due to the exponential growth of bundles depending on the numbers of goods, researchers have been working on multiple theories on how to bid on these bundles and how to choose the one that is the most beneficial, hence the multiple bundling strategies that exists.

Berhault on single-round combinatorial auctions :

- Tree-combination : In which the agent builds a tree combination consisting of all the possible outcomes, then it selects the action that leads to the highest and most beneficial bundle.
- Smart-combination : Bidding on all bundles that contain only one or two targets.
- Greedy : Each player chooses some optimal bid for the next round, assuming that the other players merely repeat their previous bid.
- Graph cut : This strategy has been said to be the most effective, where the original graph containing all options is cut in parts to create smaller graphs mainly for energy minimization and easier processing.

Schwind insight on multirounded combinatorial auction mainly consists of comparing agressive biding agents and smooth biding agents and demonstrated that an agressive impatient bidder preferred to bid high prices to quickly make resources accessible to themselves and at times can choose their bids for the next round assuming that other players will merely repeat their previous bid.

Last but not least, N. AN strategy on single round combinatorial auctions that includes Internal based strategy and Competition based strategies :

- Internal-based strategy (INT): With internal-based strategy, agents will only consider their own values to choose bundles.

$$
\begin{aligned}
V_B^i &= \sum_{j=1}^{|B|} v_j^i + \text{Synergy Value of Bundle B} \\
&= \sum_{j=1}^{|B|} v_j^i + \frac{2}{(|B|-1)} \sum_{j=1}^{|B|} \sum_{k>j} Syn^i(j,k) \\
&= |B| * (\text{Average Item Value in B} + \text{Average Pairwise Synergy Value in B}) \\
&= |B| * AC_B^i
\end{aligned}
$$

- Competition-based strategy I (COMP I) : With COMP1, we compare the value of the actual bidder in the bundle with the sum of average values of bundle's items for other bidders.

$$VR_B^i = \frac{V_B^i}{\sum_{j \in B} v_j^{-i}} \quad \text{and} \quad v_j^{-i} = \frac{1}{m-1} \sum_{q \neq i} v_i^q.$$

- Competition-based strategy II (COMP II) : With COMP2 we compare the value of each item for the actual bidder with the average value of each item for other agents

⚠ Both COMP I and COMP II are competition-based algorithms and they take into account other bidders' values of bundles in the bidding process.

Table 1: Notation

N; n	:	Set of individual items auctioned; size of N ($\|N\|$).
B	:	Bundles of items.
m	:	Number of bidders.
v_j^i	:	The value of item j for bidder i.
v_j^{-i}	:	The average value of item j for all bidders other than i.
$Syn^i(j,k)$	:	The pairwise synergy value between items j and k for bidder i.
V_B^i	:	The value of bundle B for bidder i.
AC_B^i	:	Average individual value of items in B + Average pairwise synergy of items in B, for bidder i.
VR_B^i	:	Value ratio of bidder i for bundle B.

II- General Auction Handling System

The General Auction Handling System (GAHS) provides a general framework for organizing auction-based competitions for software agents. Typically, an auction-based market is described by a set of rules stating what are the available actions to the participants, which strategy is taking place, how the winner is determined, and what price should be paid by the winner. Having a language for describing auctions from a general perspective is then at first interest. In this section, we will learn everything from figuring out how to run the auction server to receiving palpable and tangible results and interpreting them in the end.

1-Running the auction server

Starting with setting up and installing Haskell, running the auction server starts by running "auction-server --env-file .env-dev" in a cmd window and waiting for the competition description to be sent.

Figure 1: Running server through cmd command

2- Sending the Competition description

A competition description contains all the information necessary for the auction server to function properly; competition_id, title, description, the mechanism adopted, list of the goods, a list of the participating agents as well as all of the goods they're either interested in buying or willing to sell along with the budget of each agent, the private valuations and allocations. To properly showcase all this information it's only fitting to assign a json file to send the competition description to the auction server via a python application using the request library.

bidder.py | auction_starter.py | auction.json

auction_starter.py > ...

```python
import urllib.request

AUCTION_FILENAME = 'auction.json'
AUCTION_SERV_URL = "http://localhost:8800"


with open(AUCTION_FILENAME, 'r') as auction_file:
    auction_desc = auction_file.read()

    req = urllib.request.Request(f'{AUCTION_SERV_URL}/competitions',
                                 headers={'Content-type': 'application/json'},
                                 data=auction_desc.encode('utf-8'))
    res = urllib.request.urlopen(req)
    htmldoc = res.read().decode('utf-8')
    print(htmldoc)

```

Figure 2 : Python application able to read the desirable json file and send out the necessary informations to the auction server.

COMPETITION DESCRIPTIONS EXAMPLE :


```
{
  "competition_id":"competition123",
  "title":"lorem ipsum",
  "description":"lorem ipsum",
  "starts":"2021-10-25T18:25:43.511Z",
  "response_clock":4,
  "bid_clock":40,
  "mechanism":"combinatorial_exchange",
  "goods":[
    "Vintage Table",
    "Vintage Chair",
    "Hand-sewn Napkins",
    "Glasses"
  ],
  "agents":[
    {
      "id":"agent1",
      "url":"http://localhost:5000/bidder1",
      "valuation":{
        "node":"ic",
        "value":10,
        "min":1,
        "max":2,
        "child_nodes":[
          {
            "node":"leaf",
            "value":-10,
            "units":-2,
            "good":"Vintage Table"
          },
          {
            "node":"leaf",
            "value":-6,
            "units":-8,
            "good":"Vintage Chair"
          }
        ]
      },
      "allocation":{
        "Vintage Table":2,
        "Vintage Chair":8,
        "Hand-sewn Napkins":0,
        "Glasses":0
      },
      "budget":44
    },
    {
      "id":"agent2",
      "url":"http://localhost:5000/bidder2",
      "valuation":{
        "node":"ic",
        "value":10,
        "min":2,
        "max":2,
        "child_nodes":[
          {
            "node":"leaf",
            "value":9,
            "units": 2,
            "good":"Vintage Table"
          },
          {
            "node":"leaf",
            "value":6,
            "units": 8,
            "good":"Vintage Chair"
          }
        ]
      },
      "allocation":{
        "Vintage Table":0,
        "Vintage Chair":0,
        "Hand-sewn Napkins":0,
        "Glasses":0
      },
      "budget":60
    }
  ]
}
```

To translate this json code into simpler words, competition123 is based on a combinatorial_exchange mechanism and contains a list of goods the agents can bid on and/or sell such as Vintage Tables, Vintage Chairs, Hand-sewn Napkins as well as Glasses.

For each agent we had to assign an id, a url, private valuation, allocation and budget. Agent1, for exemple, runs locally on port number 5000 route "/bidder1" and from node "leaf" we can tell that the agent is willing to sell two vintage tables for a value of 10 and 8 vintage chairs for 6. From node "ic", the agent is willing to bid on at least one good and on a bundle of two goods for a value of 10. Agent2, on the other hand is bidding on two vintage table for 9 and 8 vintage chairs for 6, and wants to obtain a bundle of both of the goods for a value of 10.

3- Sending Agent's bid requests

After succefully sending the competition description via the python application "auction_starter.py" , we ought to send each agent their bid requests. To simplify the process and to automatise it, we build yet another python application for this sole purpose alone where we assign a send_bid function where it exports bid requests of each agent from their respective json file and sends them out to the auction server's address using the request library as well as parallerlly adding the agent ids in the agent_ids list.

bidder.py | auction_starter.py | auction.json

bidder.py > ...

```python
import json
from flask import Flask, request
from threading import Thread
import time
import urllib.request

AUCTION_SERV_URL = "http://localhost:8800"

app = Flask(__name__)

agent_ids = [None, None]
competitionID = [None]

def send_bid(agent_id):
    with open(str(agent_id) + '.json', 'r') as bid_file:
        auction_bid = bid_file.read()

        req = urllib.request.Request(f'{AUCTION_SERV_URL}/' + competitionID[0],
                                     headers={'Content-type': 'application/json'},
                                     data=auction_bid.encode('utf-8'))
        res = urllib.request.urlopen(req)
        htmldoc = res.read().decode('utf-8')
        print(htmldoc)
```

Figure 3 : Python application able to read the desirable agent json file, extract their bids and send out the necessary informations to the auction server.

Through out the second part of the python application, the second function we have is capable of creating as many agents there are in the agent_ids list, and depending on the message type it return specific messages.

```
27  def default_bidder(bidder_number):
28      def log(msg):
29          print(f'Bidder {bidder_number+1} > {msg}')
30      global valuations
31      message = json.loads(request.data.decode("utf-8"))
32      if message['message_type'] == 'start':
33          log('start message received')
34          agent_ids[bidder_number] = message['agent_id']
35          competitionID[0] = message['competition_id']
36          return 'ready'
37      if message['message_type'] == 'bid_request':
38          log('bid request message received')
39          t = Thread(target=send_bid, args=(agent_ids[bidder_number],))
40          t.start()
41          return 'received'
42      if message['message_type'] == 'stop':
43          log('stop message received')
44          return ''
45      else:
46          log('description received')
47          return ''
48      return 'Unimplemented request'
```

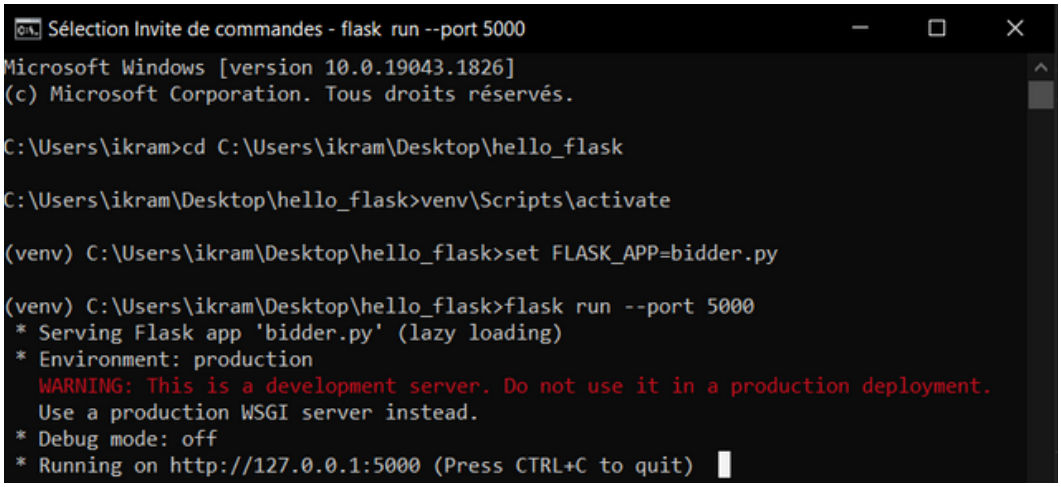
Figure 4 : Function capable of setting the number of bidders based on the competition description to automatically create agents that will interact with the auction server.

Depending on the bidder_number, the respective bidder flask function returns the agent informations to the bidder's route which has to correspond with the agent's url.

```
51  @app.route("/bidder1", methods=["GET", "POST"])
52  def bidder1():
53      return default_bidder(0)
54
55  @app.route("/bidder2", methods=["GET", "POST"])
56  def bidder2():
57      return default_bidder(1)
```

Figure 5 : Flask functions capable of returning bid requests containing the agent's bid to the auction server

To activate the agents's port number and display the messages from the default_bidder() function we have to assign a cmd window and activate it as a flask app as follows :



After sending the competition description to the auction server, and activating the port, we should receive the messages previously mentionned as be able to send the bid request messages to the server as shown in the ... figure

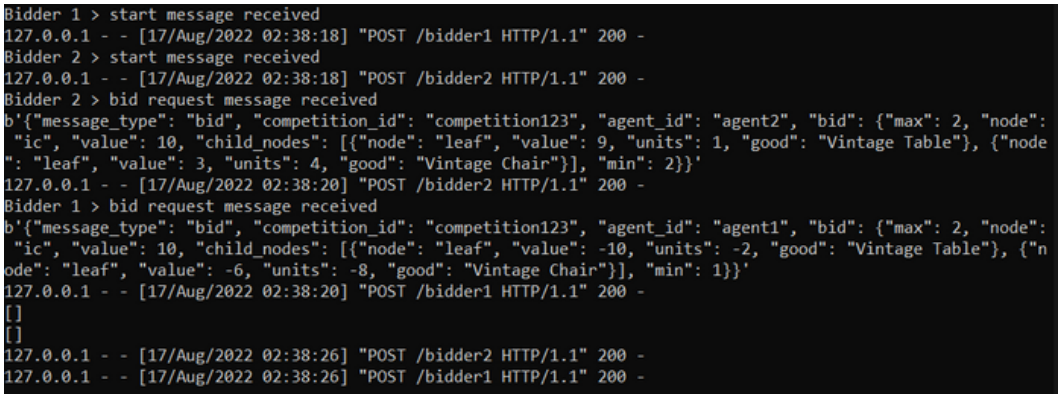


Figure 6 : The 5000 port cmd window and all the messages received

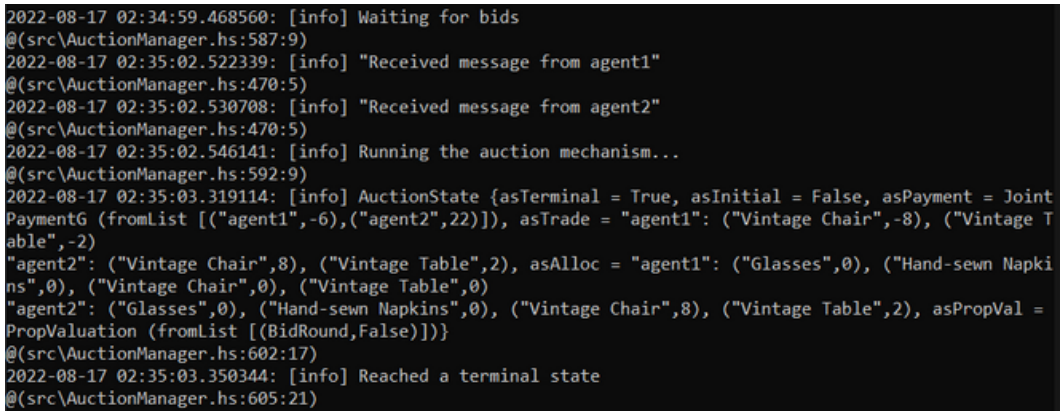


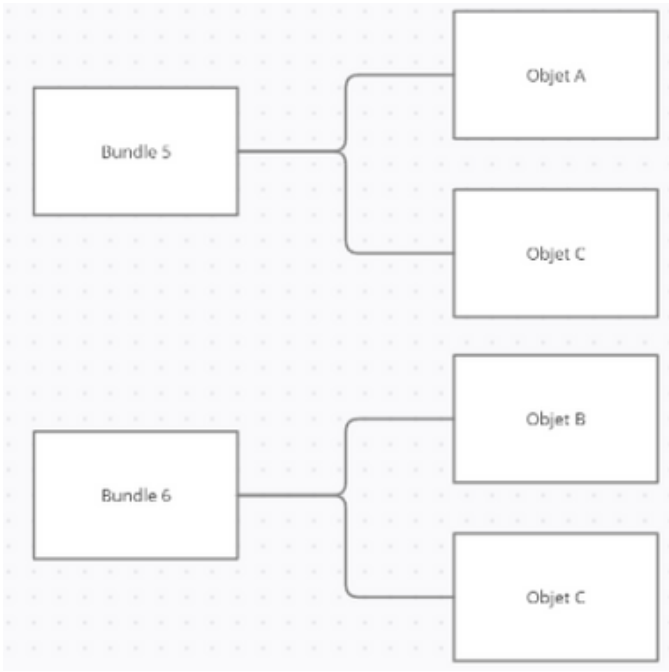
Figure 7 : Auction server's side

As results to this transaction, the agent1 has successfully sold to agent2 two vintage tables ans has 8 less vintage chairs . It's also worth mentionning that to do the math we have opted for the Cplex library. To properly showcase that the auction has ended, on the auction server's side we receive a "Reached a terminal state" message.

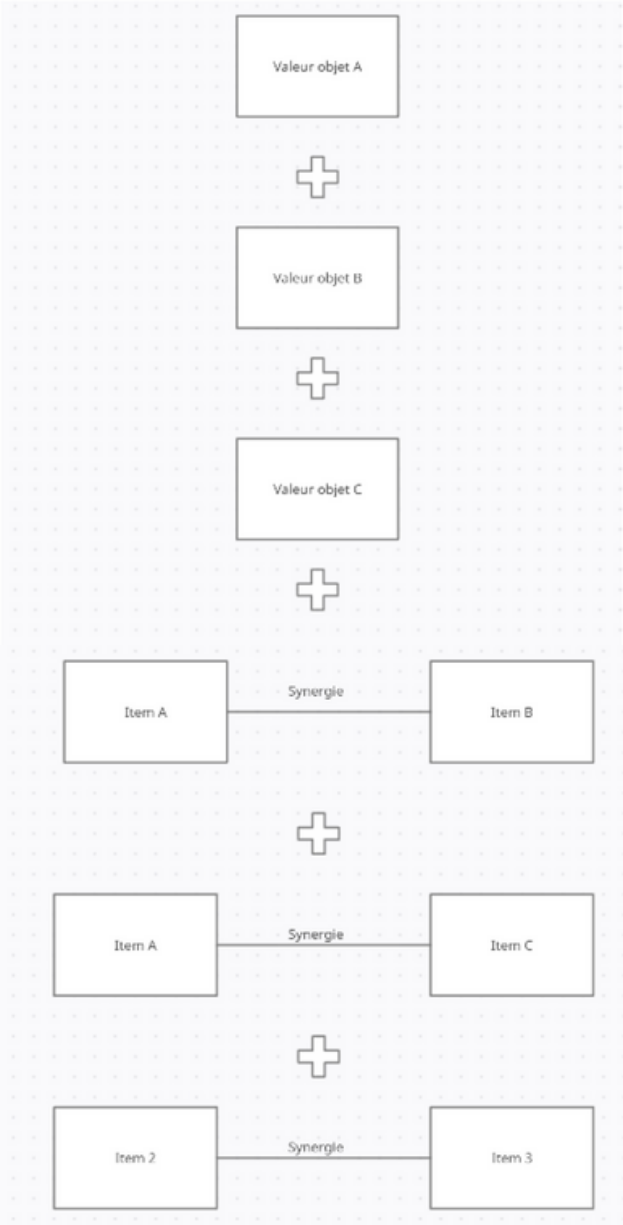
4- Bundles, Synergies and strategy application

In competition description and in bid requests, the bundles are generally represented as graph tree. In the Competition description we can have only bundles of one or two goods, because later on we can calculate the values of the bundle with whatever amount of goods wanted thanks to synergies of bundles of two objects.

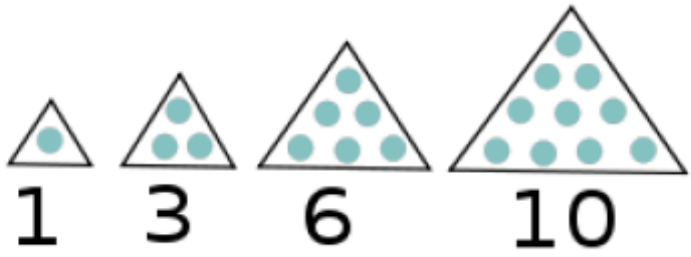
In the example below we can see that our agent is interested in 3 goods, therefore we will be having a 6 bundles each containing a combination of 2 goods . And the said agent can bid on 7 bundles, the 6 bundles and an extra one containing all 3 goods.



For the bundle of 3 goods, the value of the bundle will be determined as such :



To automatize the agent's actions, we first had to get the number of objects and bundles and to simplify the process, my team mate has sat a for loop that goes through the tree in the competition description json file and stops as soon as there are two objects in a branch / bundles of two objects to get the exact number of goods and that is because we first fill in the single objects and then the bundles of two objects. And used the triangular numbers to determine the number of possible bundles. The number of possible combinations can be determined from the number of objects as shown in the diagram below.



If there is one good, there's going to be only one bundle. If there is two, we will have a bundle of each good and one bundle containing both of the goods, hence the 3 bundles, and so on. From then, we should be able to to extract the value of each of the bundles and build a matrix to be able to properly display the values and extract them as we would like depending on the strategy that the agent would like to use.

For the INT strategy, the AC value of each bundle gets compared to another bundle and if the value of the first bundle is superieur it gets considered as a the diserable bundle and gets compared to anothe bundle and so on until we obtain a list of desirable bundles. Parralelly, the value of the non desirable bundle gets modified to 0 in the agent's request.json file. And if the bundle happens to be a desirable one the new calculated AC value is set instead of the original one.

The strategy of Comp I on the other hand is very similar to INT, but each bundle is compared to the average of the other agent's bundle value. Similarly, if the value of the bundle is superieure to the said average it is now considered a desirable bundle and the new value is set as the new bundle's value in the json file.

The following figure is a diagram that summarizes and demonstrates the process of all the elements needed for the auction to run smoothly.

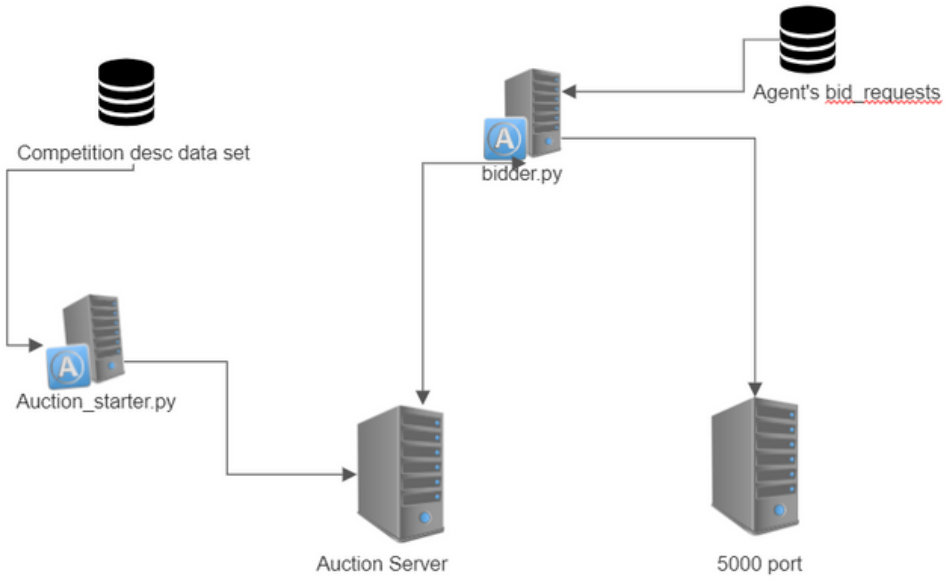


Figure 8 : The diagram summarizing the innerworkings of the Auction server

III- Creating the webserver

Creating a web server is the most optimal way to create competitions depending on the characteristics that we would want to set, such as a having a specific number of agents, setting this specific value for a bundle of specific goods. The said web server, build with Python, Flask, html and jinja2, would have to respond to client requests and display website content to the user depending on the user's input. For example, if the user requested to have a competition containing three agents and four goods, the following diagram is what will be displayed.

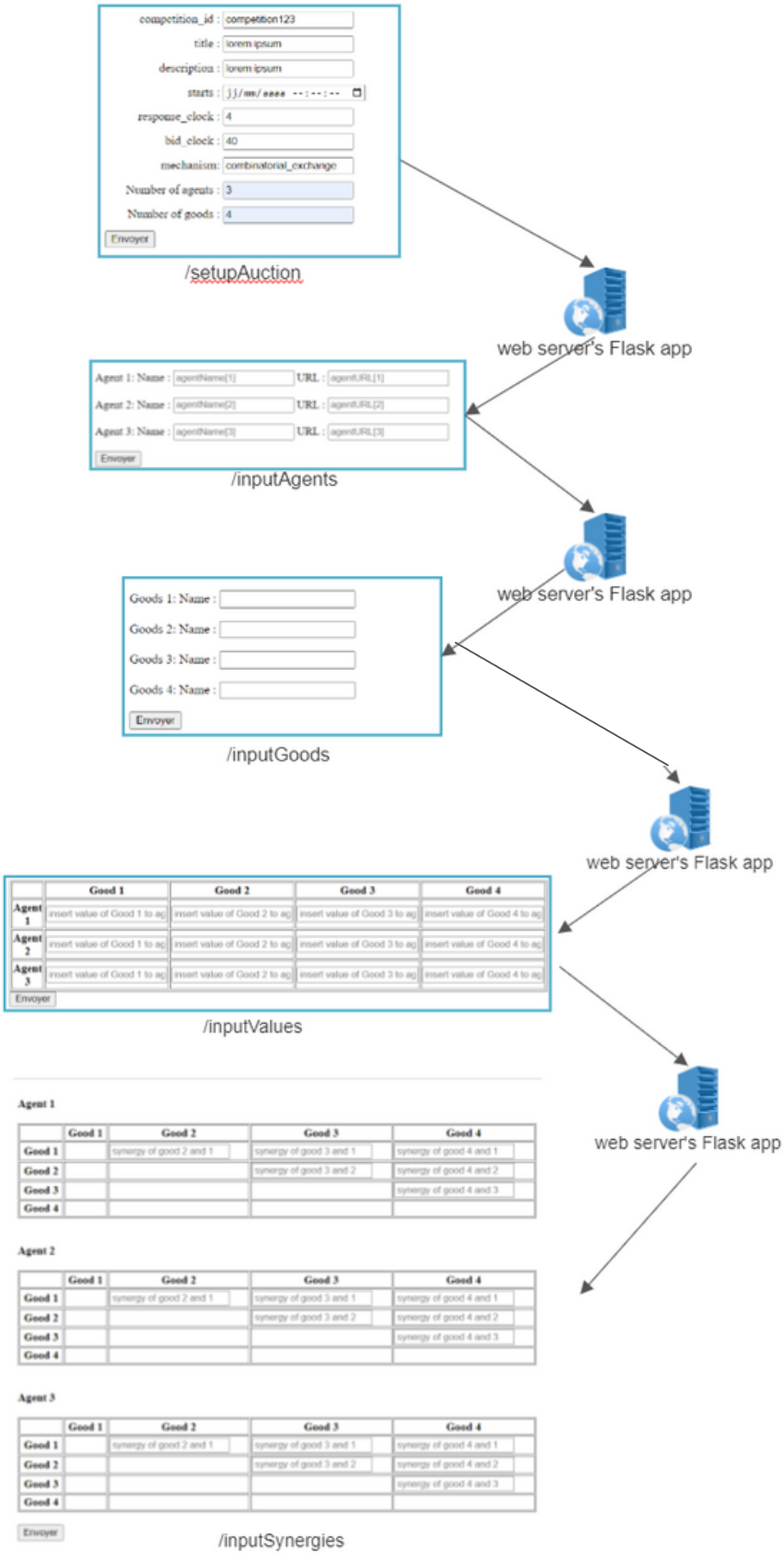


Figure 9 : The diagram summarizing the innerworkings of the Web Server

The first steps of creating the web server was to create the html pages along with the jinja2 templates to help add the dynamic part of the html. For exemple to create a competition description containing five agents , we start by writing 5 as the number of agents wanted, the flask app with the next page's route will request the /setAuction route's arguments to extract the number of agents requested and sets it, in the inputAgents's HTML,as the limit of n's range in the for loop to give out the following page :

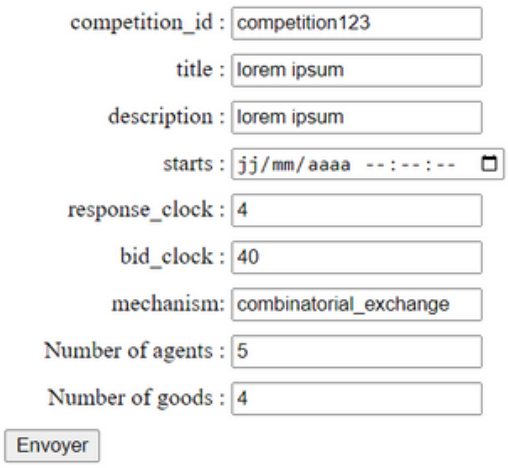


Figure 10 : SetupAuction HTML page

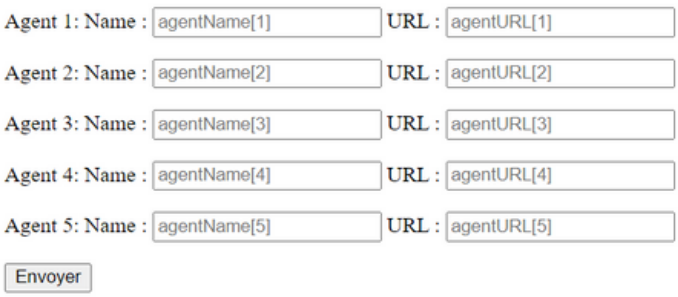


Figure 13 : inputAgent final HTML

```
<form action="/inputGoods" method=['Get'] enctype="multipart/form-data">
 {% for n in range(1, nAgents + 1) %}
    <p>
Agent {{ n }}:
   Name : <input name="agentName[{{ n }}]" placeholder="agentName[{{ n }}]" type="text">

   URL : <input name="agentURL[{{ n }}]" placeholder="agentURL[{{ n }}]" type="text">
    </p>
 {% endfor %}

    <input type="submit" Label="Go!">
</form>
```

Figure 12 : inputAgent html and the jinja2 for loop

```
28  @app.route('/inputAgents', methods=['GET'])
29  def route_inputAgents():
30
31      #if request.method == 'GET':
32          args = request.args
33          session['nAgents'] = int(args["nbAgents"])
34          nAgents=session['nAgents']
35          session['nGoods'] = int(args["nbGoods"])
36          nGoods=session['nGoods']
37
38          data={
39              'nAgents' : nAgents,
40              'nGoods' : nGoods
41          }
42
43          json_data=json.dumps(data, indent=4)
44          print(json_data)
45
46
47          return render_template('inputAgents.html', nAgents = nAgents,nGoods=nGoods,json_data=json_data)
```

Figure 11 : inputAgents flask app route

To extract the generated data from the form's user input from every html page and in an organized manner based on competition description's order, we opted to use the session library to finally generate the competition description's json and lastly send it directly to the auction's server to run the customizable auction.

To do so, we should get the data that we extracted from all of the other routes to build the final CompDesc (figure 14).

```
@app.route('/synergydisplay', methods=['GET'], endpoint='route_synergydisplay')
def route_synergydisplay():


#Getting data that we extracted from all of the other routes to build the final CompDesc
    nGoods=int(session['nGoods'])
    nAgents=int(session['nAgents'])
    setupAuction_data=session['setupAuction_data']

    InputGood_Data=session['InputGood_Data']

    Synergy_data=session['Synergy_data']

    imd_Agent=session['imd_Agent']

```

Figure 14 : Passing all the data extracted from a route to another by session library

For exemple, to extract each agent's child nodes, we extract all the values and the units wanted of the goods from the synergy_data session :

```
ChildNodes=[]

for y in range(1,nAgents+1):
    AgentChildNodes=[]
    for x in range(1,nGoods+1):
        Nodes={
            "node": "leaf",
            "value" : Synergy_data["Value of Good "+str(x)+" to agent "+str(y)],
            "units": Synergy_data["Units of Good "+str(x)+" to agent "+str(y)],
            "good": InputGood_Data["GoodName["+str(x)+"]"]
            }
        #AgentChildNodes=["agent "+str(y) :  Nodes]
        AgentChildNodes.append(Nodes) #assemble all the child nodes of a single agent

    ChildNodes.append(AgentChildNodes) #assemble all the child nodes into one list

Nodes_dat=json.dumps(ChildNodes, indent=4)
#print(Nodes_dat)
```

Figure 15 : Generating ChildNodes dictionaries

Like wise for allocations, to build small dictionaries to add to each agents dictionary :

```
allocation=[]
for y in range(1,nAgents+1):
    #GoodAllocationPerAgent=[]
    dicts={}
    for x in range(1,nGoods+1):

        dicts[InputGood_Data["GoodName["+str(x)+"]"]]=Synergy_data["Allocation of Good "+str(x)+" to agent "+str(y)]

    allocation.append(dicts)

allocation_data=json.dumps(allocation, indent=4)
```

Figure 16 : Generating allocation dictionaries

```
254
255   ####### Building a list containing agent informations
256       imd_Agent=session['imd_Agent']
257       AgentData=[]
258       for n in range(1,nAgents+1):
259
260               data={
261                   'id' : imd_Agent['agentName['+str(n)+']'],
262                   'url' : imd_Agent['agentURL['+str(n)+']'],
263                   "valuation":{
264                       'node':"ic",
265                       'value': 0,
266                       'min':1,
267                       'max': 1,
268                       'child_nodes': ChildNodes[(n-1)]
269                       },
270                   'allocation':allocation[(n-1)],
271                   'budget':imd_Agent['agentBudget['+str(n)+']']
272
273                   }
274
275               AgentData.append(data)
276
277       json_dat=json.dumps(AgentData, indent=4)
278       #print(json_dat)
```

Figure 16 : Generating Agents dictionaries

And finally, building the complete competition description's json :

```
for n in range(1,nGoods+1):
    data={  'competition_id' : setupAuction_data['competition_id'],
            'title' : setupAuction_data['title'],
            'description' : setupAuction_data['description'],
            'starts': setupAuction_data['starts'],
            'response_clock': setupAuction_data['response_clock'],
            'bid_clock': setupAuction_data['bid_clock'],
            'mechanism': setupAuction_data['mechanism'],
            'goods': ListOfGoods,
            'agents': [ AgentData

            ]
    }


json_data=json.dumps(data, indent=4)
print(json_data)
```

Figure 17 : Generating Agents dictionaries

PERSONAL ASSESSMENT

1- Technical Knowledge

From a technical point of view, this internship give me the opportunity to practice Full Stack Developement.

Back-end wise, i have had the possibility to discover Haskell, getting to practice Python and Flask and having my first "real" project to work on. Learning all about socket programming, and how requests work all while practicing Json and help making the process of creating competition descriptions easier with the session library.

Front-end wise, having had to develop a web server, HTML,/CSS and Jinja2 were my best tools to help simpilfy the making of the customized competion descriptions

2- Personal Input

From a personal point of view, this internship made me improve in alot of ways. It was my first experience as research intern where i had to experience the life of a researcher first hand;

- Gaining a deeper understanding of the scientific process,
- Discovering new knowledge and programming languages and libraries and practice what i learned through out my academic process.
- Effectively communicate your ideas and theories of how to do certain tasks.
- Eventually publish our work as a guidline for other interns to follow and understand our work during the last 4 months.

As for life skills; professionalism, time management, organisation, team work and collaboration were just as important.

Conclusion

Interning in the research field was a first for me and i had a lot to learn. It allowed me to apply my IT skills and management knowledge in a real project and helps me understand them better and showed me that i have a lot to learn in this fast paced field, which only gave me further motivation to study hard in the up coming years.

It also provides me with access to get in touch with routine in the lab, which helps me think about what to do after graduation. Personally, I like working in the lab where innovative and meaningful work can be done in a relatively personal rhythm.

Projet wise, there is still so much to be done as well as alot of room for technical growth;

- For one, it would be optimal if the auction server could adopt various auction mechanisms and strategies agents could choose from and be able to add them to the webserver for the user to choose from.

- Second, for now we are only capable of generating a competition description that consists of bundles of only one good and it would be great to develop the webserver even further down this road and have it accessible online even.

- Third, writing a guidline for others, interns specifically, to understand our 4 months worth of work and code and what each section does what in the code to clear up any confusion there might be.

References

[1] General Auction Handling System by Armen Inants

[2] Auction Theory © 2010, Elsevier Inc.

[3] Coordination non coopérative : Méthodes D'enchères by Philippe Caillou

[4] A Bidding Strategy in Combinatorial Auctions

[5] Bidding Strategies and their Impact on Revenues in Combinatorial Auctions by Na An, Wedad Elmaghraby, Pınar Keskinocak

[6] Simple Auctions with Performance Guarantees for Multi-Robot Task Allocation