Documentation - A* - "Multe autobuze"

Table of contents

Context

Summary

Given a list of **people** — each identified by a **name** and having a list of **station names** as **destinations** and a **budget**— and a list of **types of buses**, who have a corresponding **number**, a **ticket price**, the number of **minutes** buses of these types **leave the depots**, the **time** in minutes it takes to **reach its next destination** (moves at this constant time) and the list of **station names** the bus stops at, **compute the route each person has to take** so as only one person can be at one place at a time (only one person in a bus/in station), so everyone finishes their destinations and it takes **minimum time and money**.

Full text

▶ ...

Autobuzele merg în 2 sensuri, deci lista de stații (traseul) va fi parcursă și în sens opus, autobuzele plecând din ambele garajele din capete la fiecare TPLEC minute. Autobuzele când ajung la capătul de traseu opus

generării lor, dispar (generarea autobuzelor în capătul opus e independentă de ajungerea autobuzelor în acel capăt, venind din celălalt sens).

N oameni certați între ei care nu vor să se mai vadă niciodată, au toți de parcurs în ziua curentă un drum. Drumul unui om e dat ca listă de stații. Prima stație e cea din care începe drumul. Drumul omului se termină cu ultima locație pe care o are de vizitat după care omul dispare de pe hartă (nu îl mai luăm în considerare). Locațiile trebuie vizitate **exact în ordinea din listă**.O locație, **pentru a fi considerată vizitată, trebuie să îndeplinească condițiile: să fie prima nevizitată din lista omului și omul obligatoriu să coboare din autobuz acolo.** Dacă un om are în listă stațiile s1,s2,s3 și trece prin s3 și apoi prin s1, doar s1 se consideră vizitată, nu și s3. Va trebui să treacă prin stația s3 din nou: s3 se va considera vizitată doar daca ajunge în ea după ce a trecut prin s2.

Dacă un om a ajuns într-o stație și coboară din autobuz, poate urca imediat în alt autobuz, dacă următorul autobuz dorit a ajuns și el în același timp în stație. Urcările și coborârile se fac instant (deci nu iau timp).

Soluția va consta în a indica fiecărui om ce autobuze trebuie sa ia și până unde să se ducă cu ele astfel încât să treacă prin toate locațiile dorite și să nu se afle doi oameni în aceeași locație (stație sau autobuz) fiind certați. Pentru a rezolva asta e posibil să fie necesar să deplasăm un om cu o stație mai departe de destinație (iar apoi să se întoarcă în alt autobuz), dacă la destinație se află un alt om dintre cei N. **Deci dacă un om a ajuns într-o stație dar nu coboară, nu îl afectează dacă în stația respectivă așteaptă un alt om. Omul, dacă trebuie să ia un autobuz X și așteaptă în stație, obligatoriu se va urca în autobuzul X atunci când autobuzul sosește (nu are voie să lase un autobuz cu numărul X să treacă și apoi să urce în următorul cu numărul X).**

Fiecare om are o suma de bani și nu poate cumpăra mai multe bilete decât îi permite bugetul.

States and transitions

The states represent a change in a person's activity - either they were waiting and boarded a bus or they were travelling and unboarded the bus.

Nodurile-stări în graful problemei vor fi momentele de timp când un om își schimbă activitatea (așteaptă/merge cu autobuzul), mutările fiind, deci, aceste schimbări.

Cost

The cost is the sum of time and money each person spent to reach every destination they had.

Costul va fi suma tuturor timpilor parcurși și a banilor consumați pentru toți cei N oameni (vom considera costul unei mutari ca suma celor două măsuri, deoarece vrem și timpi cât mai mici dar și cât mai puțini bani consumați. Totuși trebuie să se memoreze separat pentru a fi indicate cu exactitate în afișarea drumului). Din momentul în care un om și-a terminat drumul, nu se mai adună nimic la cost pentru el.

Running the application

In the folder with the project, open a console and run:

```
python multeautobuze.py <folderinput> <folderoutput> <nsol> <timeout>
```

folderinput - path to the folder with the input files, needs to be a valid path

folderoutput - path to the output folder, if not available it will be created

nsol - number of required solutions from every algorithm besides optimized a* that returns only one solution

timeout - time in seconds after which the algorithm running should be stopped if it didn't finish, leave *0* for no timeout

Input files

Given input

input1 - input with **no finishing state**, not all destinations can be reached

input2 - an **initial state that is also final**, all persons have only one destination, the one they spawn at

input3 - input with **multiple solutions, no algorithms timeout**

input4 - input with **multiple solutions, some algorithms timeout**, **"euristica inadmisibila" doesn't give the solution with minimum cost**

Format

- Primul rând va fi ora la care toți oamenii sunt in prima stație din lista lor. Tot la această oră pleacă câte un autobuz din fiecare garaj (un garaj e un capăt de traseu, deci pleacă autobuze în ambele sensuri pe un traseu. Pe același rând va fi și ora la care trebuie să fi terminat toți oamenii traseul (nu avem voie să generăm stări după această oră).
- Următoarele rânduri conțin date pentru fiecare autobuz: numărul autobuzui, prețul biletului, intervalul de timp între două plecări din garaj, timpul între două stații consecutive, traseul indicat printr-un sir de forma statie_1,statie_2, ... statie_n.
- Un rand cu numărul de oameni, apoi fiecare om cu suma de bani pe care o are și traseul pe care îl dorește

```
HH:MM HH:MM # Start time and end time
BusNr Plei TPLECmin TDEPmin "Station 1","Station 2",...
    # P is the ticket price
    # TPLEC is the nr. of minutes buses leave the depots
    # TDEP is the nr. of minutes it takes the bus to reach the next station
    # The list of stations, encapsulated in "" and split by a single comma
.....
X oameni
# Number of people
PersonName Blei "Station 1","Station 2",...
    # B is the budget
    # The list of stations, encapsulated in "" and split by a single comma
....
```

Output file format

For each input a file with the name `<input file name>_output` will be created in the provided directory. In this file the algorith name will be displayed, then the heuristic used and finally the solutions, each separated.
Example:

```
                File "input_output"
================================================
Solutii pentru algoritmul breadth_first
================================================

Euristica folosita: euristica banala
________________________________________________
....
Lungimea drumului este: 6
Costul drumului este: 32
Numarul maxim de noduri in memorie: 323
Numarul total de noduri calculate: 445
Solutia a fost gasita in 0.7570152282714844
------------------------------------------------
Stopped because of timeout
------------------------------------------------
...

Euristica folosita: euristica admisibila 1
________________________________________________
.
.
.
```

Solution format

În fișierul de output se vor afișa drumurile-soluție In cadrul afisarii soutiei, pentru fiecare nod din drum:

- se va arata fiecare moment de timp în care s-a schimbat ceva, începând cu ora de la care pornesc toate deplasările. Schimbările se referă doar la urcatul sau coborâtul din autobuz, nu și faptul că un om a trecut printr-o stație (în care nu a coborât). Momentul de timp este reprezentat prin ora.
- La fiecare pas se va afișa starea fiecarui om (care încă nu și-a terminat drumul) cu următoarele tipuri de mesaje:
    - "Omul [nume-om] așteaptă în stația [nume-statie]" "Omul [nume-om] se deplasează cu autobuzul [numar-autobuz] de la statia [nume-statie] la statia [nume-statie] pe traseul [traseu]", unde zonele între parenteze drepte vor fi înlocuite cu informația cerută. Traseul este strict zona pe care o parcurge omul, și are forma "statie_1 -> statie_2 -> .... statie_n", iar stațiile între care se deplasează sunt cele între care se află acum, nu capetele traseului.
    - Omul [nume-om] a coborât în stația [nume-stație] din autobuzul [nr]
    - Omul [nume-om] a coborât în stația [nume-stație] din autobuzul [nr] și și-a terminat traseul
    - Omul [nume-om] a urcat în stația [nume-stație] in autobuzul [nr] pentru traseul [traseu]
    - După status se va afișa și bugetul, timpul total de mers cu autobuzul și timpul total de așteptare

Important! mesajul cu terminarea traseului ar trebui să fie ultimul pentru fiecare om, și acel om nu va mai fi menționat în nodurile următoare.

Observație: dacă un om coboară dintr-un autobuz, și urcă în același moment de timp în alt autobuz (e posibil când ambele autobuze au ajuns în stație în același timp), se vor afișa atât mesajul de coborâre cât și de urcare.

- Se va afișa și costul de până atunci: suma pentru toți oamenii a banilor consumați și a timpului petrecut p

```
0)
08:00
Omul Ionel a urcat în stația "Hotel Gogonel" in autobuzul 200 pentru traseul
"Hotel Gogonel"->"Soseaua papucilor"->"Piata 3 castraveti". Buget: 97lei. Timp
mers: 0min. Timp asteptare: 0min.
Omul Gigel așteaptă în stația "Dealul melcului schiop". Buget: 80lei. Timp mers:
0min. Timp asteptare: 0min.
Omul Costica așteaptă în stația "Strada Aricilor". Buget: 55lei. Timp mers: 0min.
Timp asteptare: 0min.
Cost pana acum: 3.
```

Heuristics

"Euristica banală"

A simple approach. If the **state is final** evidently the estimated cost to a final state is **0**, but if it **isn't final** it means there's at least one of the following actions that can take place:

- Person got down from a bus $\rightarrow$ **at least 1 minute passed**, because a person can't get down from a bus the moment they boarded it
- Person boarded a bus $\rightarrow$ **person spent money** on the ticket, also **has to unboard** at a future time $\rightarrow$ at least 1 minute to travel to the next station and get down

The actual cost it takes to reach a final state is more or equal to 1.

"Euristica admisibilă 1"

Each person **waiting** for a bus, that **hasn't finished** their destinations, has to **board at least one bus** and **travel to at least one station** to finish. Using this information we can imagine the **best possible case**, each of them boarded the bus with the minimum ticket price and the minimum travel time and has only one destination left. The **sum of these costs** for each waiting person is **less or equal** to the actual cost of reaching a final state. *Example*:

```
Buses:
200, 3 min move time, ticket price 1lei, stations "Station 1","Station 2"
300, 10 min move time, ticket price 1lei, stations "Station 1","Station 3"
=> Minimum ticket price: 1 lei
   Minimum travel time: 3min
```

```
At the start time
Person 1 waiting, location "Station 1" - budget 9lei, destinations "Station 3"
Person 2 waiting, location "Station 2" - budget 15lei, destinations "Station 1",
"Station 3"
Estimated cost of reaching final state: (1 + 3) + (1 + 3) = 8
Actual cost of reaching final state: 4 + 4 + 11 = 19

Explanation:
    Person 1 takes bus 200, pays 1lei and reaches "Station 2" in 3min, gets down
and finishes.
    Person 2 takes bus 200, pays 1lei and reaches "Station 1" in 3min, gets down.
Takes bus 300, pays 1lei and reaches "Station 3" in 10min, gets down and finishes.
```

"Euristica admisibilă 2"

The time it takes from one state to reach the final state is at least the time it takes the **person with the maximum number of destinations left to finish using only buses with the minimum travel time**. If every bus has the same move time, by the time the person with the maximum number of destinations left has finished, the rest have too. *Example*:

```
Buses:
200, 3 min move time, ticket price 1lei, stations "Station 1","Station 2"
300, 10 min move time, ticket price 1lei, stations "Station 1","Station 3"
=> Minimum travel time: 3min

At the start time
Person 1 waiting, location "Station 1" - budget 9lei, destinations "Station 3"
Person 2 waiting, location "Station 2" - budget 15lei, destinations "Station 1",
"Station 3"
Estimated cost of reaching final state: 2 * 3 = 6
Actual cost of reaching final state: 4 + 4 + 11 = 19

Explanation:
    Person 1 takes bus 200, pays 1lei and reaches "Station 2" in 3min, gets down
and finishes.
    Person 2 takes bus 200, pays 1lei and reaches "Station 1" in 3min, gets down.
Takes bus 300, pays 1lei and reaches "Station 3" in 10min, gets down and finishes.
```

"Euristica admisibilă 3"

Combining the heuristics stated above, each waiting person boards at least one bus, with the ticket price at least the minimum. The time it takes from one state to reach the final state is at least the time it takes the person with the maximum number of destinations left to finish using only buses with the minimum travel time. Doesn't have a different approach then the other two, but was made too see how a stronger heuristic compares to the others. *Example*:

```
Buses:
200, 3 min move time, ticket price 1lei, stations "Station 1","Station 2"
```

```
300, 10 min move time, ticket price 1lei, stations "Station 1","Station 3"
=> Minimum ticket price: 1 lei
   Minimum travel time: 3min

At the start time
Person 1 waiting, location "Station 1" - budget 9lei, destinations "Station 3"
Person 2 waiting, location "Station 2" - budget 15lei, destinations "Station 1",
"Station 3"
Estimated cost of reaching final state: 2 + 2 * 3 = 8
Actual cost of reaching final state: 4 + 4 + 11 = 19

Explanation:
    Person 1 takes bus 200, pays 1lei and reaches "Station 2" in 3min, gets down
and finishes.
    Person 2 takes bus 200, pays 1lei and reaches "Station 1" in 3min, gets down.
Takes bus 300, pays 1lei and reaches "Station 3" in 10min, gets down and finishes.
```

"Euristica neadmisibilă"

Assuming every person that hasn't finished, takes at least **one new bus with the maximum ticket price,** and **every person travels the maximum number of destinations with the maximum travel time**. *Example*:

```
Buses:
200, 3 min move time, ticket price 1lei, stations "Station 1","Station 2"
300, 10 min move time, ticket price 2lei, stations "Station 1","Station 3"
=> Maximum ticket price: 2lei
   Maximum travel time: 10min

At the start time
Person 1 waiting, location "Station 1" - budget 9lei, destinations "Station 3"
Person 2 waiting, location "Station 2" - budget 15lei, destinations "Station 1",
"Station 3"
Estimated cost of reaching final state: 2 * 2 * 10 + 2 * 2 = 44
Actual cost of reaching final state: 4 + 4 + 12 = 20

Explanation:
    Person 1 takes bus 200, pays 1lei and reaches "Station 2" in 3min, gets down
and finishes.
    Person 2 takes bus 200, pays 1lei and reaches "Station 1" in 3min, gets down.
Takes bus 300, pays 2lei and reaches "Station 3" in 10min, gets down and finishes.
```

Optimizations

Validation

Only inputs following the format stated above will work for the application, otherwise a message will be printed and execution will end. A correct formated input is then verified (function *Information.checkIfPossible*), **if the state is invalid** (two persons are at the same station at the start) execution stops and a message is printed.

Two more validation tests follow, both for **checking if the input has no solution**. First is the **money test**, if the person with the **lowest budget can't afford the lowest ticket price** it's impossible for them to reach any destination, so the execution is stopped. Second test is the **time test** if the **lowest move time of the buses is bigger then the duration of the day** (provided by the difference in between the end time and start time) no bus can move, nobody can reach any destination, so the execution stops.

Optimization

After verifying input format and validity, another question is raised. **If the input is already a finishing state?** With a function (*Node.isFinal*) **determine this before starting the algorithms**, write it in the output file and move to the next input.

For solving a "solution" graph is created, where the nodes represent a new state -- called *Information*, along with the cost it took reaching this point and the estimated cost to reach a solution.

The information is defined by :

- The list of schemas for the buses (the one given in the input)
- The list of the persons who haven't finished
- The list of all the buses on route
- A time in minutes, calculated by assuming start time is minute 0
- An action, either "up" for a person boarding a bus, "down" for a person unboarding or "finished" for a person who unboarded and finished.
- A list of time values, where an action could be triggered (new buses leave the depot or a bus moves to the next station)
- The person who triggered the event
- The bus who triggered the event
- Minimum ticket price and travel time, calculated at initialization of state and used in the heuristics

At some point a **state** could represent a **dead end**, meaning from that point it is **impossible to reach a final state**. Function *Information.stopGenerating* applies multiple tests to see if there's no possible action that can take place from this point forward.

Test1: if **all waiting persons** have reached a **budget that is lower then the minimum ticket price** nobody can board a bus, its a dead end.

Test2: if **all persons** have succesors where they **board all the types of buses** (in both directions) there's **no other possible state** that can be generated

Test3: a final state can't generate any more states

Time analization

Input 3 - minimum solution: length 6, cost 32

Type of algorithm	Type of heuristic	Time	Solution length	Solution cost	Max nodes in memory	Number of nodes generated
Breadth First	Banală	24.069	6	32	7843	11035

Type of algorithm	Type of heuristic	Time	Solution length	Solution cost	Max nodes in memory	Number of nodes generated
	Admisibilă 1	23.176	6	32	7843	11035
	Admisibilă 2	22.36	6	32	7843	11035
	Admisibilă 3	22.51	6	32	7843	11035
	Neadmisibilă	22.936	6	32	7843	11035
Depth First	Banală	0.208	26	221	10	100
	Admisibilă 1	0.210	26	221	10	100
	Admisibilă 2	0.209	26	221	10	100
	Admisibilă 3	0.211	26	221	10	100
	Neadmisibilă	0.226	26	221	10	100
Iterative Depth First	Banală	8.718	6	32	10	4415
	Admisibilă 1	8.718	6	32	10	4415
	Admisibilă 2	8.218	6	32	10	4415
	Admisibilă 3	8.238	6	32	10	4415
	Neadmisibilă	8.248	6	32	10	4415
A*	Banală	0.717	6	32	323	445
	Admisibilă 1	0.415	6	32	167	262
	Admisibilă 2	0.298	6	32	150	206
	Admisibilă 3	0.244	6	32	123	175
	Neadmisibilă	0.055	6	32	35	42
Optimized A*	Banală	0.409	6	32	220	247
	Admisibilă 1	0.260	6	32	142	166
	Admisibilă 2	0.212	6	32	122	146
	Admisibilă 3	0.185	6	32	109	130
	Neadmisibilă	0.053	6	32	43	42
IDA*	Banală	5.423	6	32	10	3613
	Admisibilă 1	2.471	6	32	10	1808
	Admisibilă 2	2.121	6	32	10	1590
	Admisibilă 3	1.371	6	32	10	1062

Type of algorithm	Type of heuristic	Time	Solution length	Solution cost	Max nodes in memory	Number of nodes generated
	Neadmisibilă	0.055	6	32	10	43

Input 4 - minimum solution: length 10, cost 47

Type of algorithm	Type of heuristic	Time	Solution length	Solution cost	Max nodes in memory	Number of nodes generated
Breadth First	Banală	> 25s				
	Admisibilă 1	> 25s				
	Admisibilă 2	> 25s				
	Admisibilă 3	> 25s				
	Neadmisibilă	> 25s				
Depth First	Banală	6.094	48	379	10	1477
	Admisibilă 1	6.243	48	379	10	1477
	Admisibilă 2	5.949	48	379	10	1477
	Admisibilă 3	5.888	48	379	10	1477
	Neadmisibilă	5.810	48	379	10	1477
Iterative Depth First	Banală	> 25s				
	Admisibilă 1	> 25s				
	Admisibilă 2	> 25s				
	Admisibilă 3	> 25s				
	Neadmisibilă	> 25s				
A*	Banală	6.460	10	47	2212	2864
	Admisibilă 1	3.226	10	47	1140	1532
	Admisibilă 2	2.001	10	47	890	1151
	Admisibilă 3	1.538	10	47	711	917
	Neadmisibilă	0.334	12	69	157	203
Optimized A*	Banală	2.226	10	47	927	1021
	Admisibilă 1	1.052	10	47	520	597
	Admisibilă 2	0.900	10	47	428	480
	Admisibilă 3	0.676	10	47	341	377

Type of algorithm	Type of heuristic	Time	Solution length	Solution cost	Max nodes in memory	Number of nodes generated
	Neadmisibilă	0.279	12	69	151	161
IDA*	Banală	> 25s				
	Admisibilă 1	19.561	10	47	10	12616
	Admisibilă 2	14.927	10	47	9	8090
	Admisibilă 3	9.110	10	47	9	5036
	Neadmisibilă	0.4560	10	47	9	304

Conclusions

A bad heuristic tested on a limited number of inputs can "trick" everyone, because it gives a much better time value and uses less memory. But as we can see for input 4, the "euristica neadmisibila" doesn't give the minimum solution, because the nodes that lead to the wanted solution are misslabeled and put at the end of the queue.

***Breadth first* is the weakest algorithm** provided. The succesors aren't sorted in any way, so in the worst case reaching a final state requires **iterating through all of the nodes**, which takes **a lot of time and also memory**, because for each node more succesors could be added.

***Depth first* is also weak**, it doesn't provide the minimum length solution and can **fill the memory with additional values because of recursion**, also there's the risk of maximum recursion depth to be reached.

A* is a good algorithm, and in the tables provided it is easily noticeble how **a good heuristic can improve both the time and memory used**. The algorithm modifes breadth first by sorting the nodes in the queue by cost. This leads to an average reduction of 98.19% for time.

Optimizing A* leads to an even better result, but for the disadvantage of computing only one solution. In the table for input 3 we can see that the total number of nodes generated is 11035, and with "euristica admisibila 3" we reduce the number of generated nodes to only 161, which is close to 98.54% less. Same with the time, a reduction of close to 98.91% from the time it takes breadth first to solve. Regarding A*, the optimization gives an average reduction of 53.10% for memory usage and 60.99% for time.

Ida* performs well for finding **multiple solutions close to each other in the graph**, it takes a longer time otherwise and uses recursion, but keeps the lowest number of nodes in memory.

Using the right algorithm we can achieve both a good time and memory usage. For the minimum solution the best to use is optimized A* with "euristica admisbila 3" and A* for multiple solutions. Don't get tricked by the beautiful values an invalid heuristic provides, it works if we are looking for a random solution, but not for finding solutions in the ascending order of cost and length.