**UNIVERSITETI I PRISHTINËS**

***FAKULTETI I SHKENCAVE MATEMATIKE-NATYRORE***

***Departamenti i Matematikës***

Logo

Description automatically generated

**Dokumentimi teknik**

***Lënda: Inteligjenca Artificiale  
Tema: Vehicle Routing Problem me Simulated Annealing***

***Profesori:***  ***Studenti:***

*Prof.Dr.Eliot Bytyci Leart Bllacaku Gentin Morina Donat Gosalci*

Përmbajtja

[1. Hyrje në problem 3](#_Toc135599000)

[2. Simulated Annealing 3](#_Toc135599001)

[3. Pseudokodi 5](#_Toc135599002)

[4. Rezultati 6](#_Toc135599003)

[Referencat 8](#_Toc135599004)

[Figura 1. Algoritmi SA. 4](#_Toc135598933)

[Figura 2. Pseudokodi. 6](#_Toc135598934)

[Figura 3. Rezultati. 7](#_Toc135598935)

# Hyrje në problem

Vehicle Routing Problem (VRP) është një nga problemet e optimizimit më të studiuara dhe konsiston në optimizimin e dorëzimit të mallrave tek klientët. Ky problem zë një vend të rëndësishëm në fushat e menaxhimit të shpërndarjes, logjistikës dhe transportit. Interesi për VRP është motivuar nga rëndësia ekonomike dhe vështirësia sfiduese [1].

Që nga përcaktimi i tij, VRP ka sjellë shumë variante me kushte shtesë ose përmirësime në problemin origjinal. Për t'u përballur me to, janë zhvilluar shumë qasje të ndryshme [1]. Megjithatë, pavarësisht numrit të lartë të algoritmeve që janë propozuar për ta zgjidhur, VRP mbetet një sfidë reale për t'u përballur me të. Kjo është veçanërisht e vërtetë kur madhësia e problemit rritet për shkak të natyrës së vështirë [2].

Një nga qasjet që mund të përdoren për të zgjidhur VRP është kërkimi lokal i cili është një metodë e pjesshme dhe e ndërprerë që përfshin eksplorimin e hapësirës së kërkimit lokalisht duke filluar nga një zgjidhje fillestare dhe duke e përmirësuar atë iterativisht. Për këtë qëllim, algoritmi lëviz nga zgjidhja në zgjidhje duke zbatuar ndryshime lokale [1] [2]. Avantazhi i kësaj teknike është që mund të gjendet një zgjidhje e arsyeshme në një hapësirë të gjerë kërkimi duke përdorur një sasi të vogël burimesh. Megjithatë, zgjidhja e gjetur nuk garanton të jetë optimale [2].

# Simulated Annealing

Simulated Annealing (SA) është një algoritëm optimizimi metaheuristik i frymëzuar nga procesi i pjekjes në metalurgji. Përdoret për zgjidhjen e problemeve të optimizimit kombinues ku qëllimi është gjetja e zgjidhjes më të mirë midis një numri të madh zgjidhjesh të mundshme [1] [2] [3].

Algoritmi e ka marrë emrin nga analogjia me procesin e pjekjes, i cili përfshin ngrohjen dhe ftohjen e një materiali për të zvogëluar defektet e tij dhe për të rritur qëndrueshmërinë e tij. Në mënyrë të ngjashme, SA fillon me një zgjidhje fillestare dhe eksploron në mënyrë të përsëritur hapësirën e kërkimit duke bërë ndryshime të vogla në zgjidhje. Këto ndryshime shpesh quhen "lëvizje" dhe mund të përfshijnë shkëmbimin, rrokullisjen ose modifikimin e elementeve në zgjidhje [3].

SA ruan një parametër "temperaturë" që kontrollon shkëmbimin eksplorim-shfrytëzim. Fillimisht, temperatura vendoset në një vlerë të lartë, duke i lejuar algoritmit të eksplorojë një gamë të gjerë zgjidhjesh, madje edhe ato që janë më të këqija se zgjidhja aktuale [1] [3]. Ndërsa algoritmi përparon, temperatura zvogëlohet gradualisht, duke çuar në një kërkim më të fokusuar rreth zonave premtuese.

Në çdo përsëritje, algoritmi vlerëson cilësinë e zgjidhjes aktuale duke përdorur një funksion objektiv, i cili i cakton një vlerë numerike çdo zgjidhjeje që tregon cilësinë e saj. Funksioni objektiv mund të bazohet në kritere të ndryshme, në varësi të problemit që zgjidhet [3].

Gjatë procesit të eksplorimit, SA pranon zgjidhje më të këqija me një probabilitet të caktuar të përcaktuar nga një "orar ftohje". Ky pranim probabilistik lejon që algoritmi t'i shpëtojë optimës lokale dhe të vazhdojë të eksplorojë hapësirën e kërkimit. Fillimisht, probabiliteti i pranimit është i lartë, por zvogëlohet me uljen e temperaturës, duke favorizuar zgjedhjen e zgjidhjeve më të mira.

Orari i ftohjes përcakton se si temperatura zvogëlohet me kalimin e kohës. Ekzistojnë strategji të ndryshme për përditësimin e temperaturës, si ftohja lineare, ftohja gjeometrike ose ftohja logaritmike. Zgjedhja e orarit të ftohjes varet nga karakteristikat e problemit dhe ekuilibri i dëshiruar midis eksplorimit dhe shfrytëzimit [3].

SA vazhdon përsëritjet derisa të plotësohet një kriter ndalimi, i cili mund të jetë një numër maksimal përsëritjesh, një cilësi specifike e synuar ose një afat kohor. Pasi të përfundojë algoritmi, zgjidhja më e mirë e gjetur gjatë kërkimit kthehet si rezultat përfundimtar [4].

Është e rëndësishme të theksohet se SA nuk garanton gjetjen e zgjidhjes optimale globalisht, por është në gjendje të eksplorojë një hapësirë ​​të madhe kërkimi në mënyrë efikase dhe shpesh prodhon zgjidhje me cilësi të lartë. Është veçanërisht i dobishëm kur kemi të bëjmë me probleme komplekse optimizimi ku algoritmet deterministe mund të ngecin në optimën lokale [2].

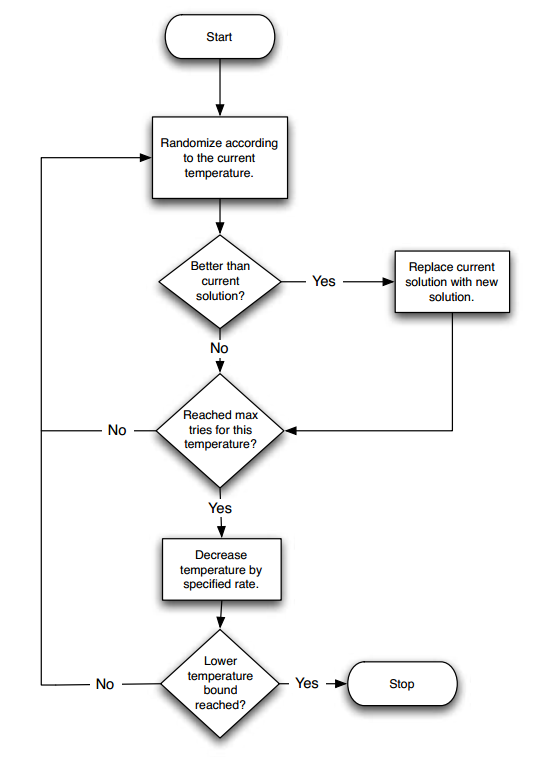


Figura 1. Algoritmi SA.

# Pseudokodi

Më poshtë gjeni një shpjegim hap pas hapi të pseudokodit për algoritmin e implementuar nga ne, Simulated Annealing (SA):

1. Funksioni **solve()** inicializon zgjidhjen aktuale, koston aktuale, zgjidhjen më të mirë, koston më të mirë dhe temperaturën fillestare.
2. Algoritmi hyn në një lak që funksionon për një numër të caktuar përsëritjesh **(NUM\_ITERATIONS)**.
3. Brenda qarkut, krijohet një zgjidhje e re duke thirrur funksionin **generateNeighborSolution().** Ky funksion gjeneron një zgjidhje fqinje duke shkëmbyer në mënyrë të rastësishme dy klientë brenda një rruge automjeti të zgjedhur rastësisht.
4. Kostoja e zgjidhjes së re llogaritet duke thirrur funksionin **accountTotalDistance().**
5. Probabiliteti i pranimit të zgjidhjes së re llogaritet duke thirrur funksionin **AcceptanceProbability()**, i cili krahason koston aktuale dhe koston e re me temperaturën aktuale.
6. Nëse probabiliteti i pranimit është më i madh se një numër i krijuar rastësisht midis 0 dhe 1, zgjidhja e re pranohet si zgjidhja aktuale dhe kostoja aktuale përditësohet.
7. Nëse kostoja e zgjidhjes së re është më e ulët se kostoja më e mirë e gjetur deri më tani, zgjidhja e re bëhet zgjidhja më e mirë dhe kostoja më e mirë përditësohet.
8. Temperatura zvogëlohet duke e shumëzuar me shpejtësinë e ftohjes **(COOLING\_RATE).**
9. Cikli vazhdon derisa të plotësohet numri i specifikuar i përsëritjeve.
10. Së fundi, zgjidhja më e mirë kthehet si rezultat i funksionit solve().
11. Funksioni **AcceptanceProbability()** llogarit probabilitetin e pranimit të një zgjidhjeje më të keqe bazuar në koston aktuale, koston e re dhe temperaturën. Nëse kostoja e re është më e ulët se kostoja aktuale, probabiliteti vendoset në 1.0. Përndryshe, llogaritet duke përdorur formulën: **e^((Kostoja aktuale - Kostoja e re) / temperatura)**, ku **e** është baza e logaritmit natyror.

Algoritmi i SA eksploron në mënyrë të përsëritur hapësirën e zgjidhjes, duke lejuar lëvizje të rastësishme përpjetë (duke pranuar zgjidhje më të këqija) bazuar në probabilitetin e pranimit, i cili varet nga temperatura. Ndërsa temperatura zvogëlohet gjatë përsëritjeve, algoritmi gradualisht zvogëlon probabilitetin e pranimit të zgjidhjeve më të këqija, duke u fokusuar në përmirësimin e cilësisë së përgjithshme të zgjidhjes. Procesi vazhdon derisa të plotësohet kushti i përfundimit (numri i specifikuar i përsëritjeve), duke rezultuar në një zgjidhje të optimizuar.

A picture containing text, screenshot, software

Description automatically generated

Figura 2. Pseudokodi.

# Rezultati

Kodi ynë implementon problemin e Vehicle Routing (VRP) duke përdorur algoritmin e Simulated Annealing (SA). Synimi është gjetja e një zgjidhje të optimizuar për një problem rrumbullakimi ku një flotë automjetesh duhet të shërbejë një grup klientësh, duke minimizuar distancën totale të udhëtuar.

Kodi inicializon një matricë distancash që paraqet distancat midis lokacioneve të ndryshme (klientë dhe depo). Gjithashtu përcakton porositë për secilin klient, kapacitetin e automjeteve, numrin e automjeteve, temperaturën fillestare, shkallën e ftohjes dhe numrin e iteracioneve për algoritmin SA.

Metoda "main" inicializon një instancë të klasës "VehicleRoutingProblemSA" dhe kryen hapat e mëposhtëm:

* Llogarit numrin total të porosive për t'u dorëzuar.
* Përcakton numrin e automjeteve që nevojiten për të plotësuar porositë.
* Nëse nuk ka mjaftueshëm automjete, shfaq një mesazh.

Në të kundërt, vazhdon të zgjidhë problemet duke përdorur algoritmin SA.

Metoda "solve" është thelbi i implementimit të algoritmit SA. Ajo inicializon zgjidhjen aktuale, koston aktuale, zgjidhjen më të mirë dhe koston më të mirë. Pastaj kryen iteracione nëpërmjet një numri të caktuar të iteracioneve, gjeneron zgjidhje të reja fqinje, vlerëson koston e tyre dhe i pranon ose i refuzon ato bazuar në probabilitetin e pranimit të përcaktuar nga temperatura aktuale.

Metoda "generateInitialSolution" gjeneron një zgjidhje fillestare duke caktuar klientët në mënyrë të rastësishme tek automjetet, duke siguruar që kapaciteti i automjetit të mos tejkalohet. Metoda "generateNeighborSolution" gjeneron një zgjidhje fqinje të re duke ndërruar dy klientë në mënyrë të rastësishme brenda një rruge.

Metoda "calculateTotalDistance" llogarit distancën totale të udhëtuar për një zgjidhje të caktuar duke mbledhur distancat midis klientëve të ndjekur dhe depo. Metoda "calculateLoad" llogarit ngarkesën totale (numrin e porosive) për një rruge të caktuar.

Metoda "acceptanceProbability" përcakton probabilitetin e pranimit të një zgjidhje të re bazuar në koston aktuale, koston e re dhe temperaturën. Kostot më të ulëta pranohen gjithmonë, ndërsa kostot më të larta mund të pranohen me një probabilitet që zvogëlohet sipas uljes së temperaturës.

Metoda "printSolution" printon zgjidhjen e optimizuar, duke përfshirë distancën totale të udhëtuar dhe rrugët për secilin automjet.

Për të marrë rezultate specifike për këtë kod, duhet ta ekzekutojmë. Output-i do të shfaqë zgjidhjen e optimizuar, distancën totale të udhëtuar dhe rrugët për secilin automjet. Në vijim jepet konzola me rezultatin pas ekzekutimit të kodit.

A screen shot of a computer program

Description automatically generated with low confidence

Figura 3. Rezultati.

# Referencat

|  |  |
| --- | --- |
| [1] | Z. X. Li Dai, "Simulated annealing algorithm for finding the spanning directed trees with many leaves," [Online]. Available: https://ieeexplore.ieee.org/document/5974942. |
| [2] | R. V. T. P. S. Shatabdi Chakrabarty, "A Simulated Annealing optimization technique to obtain uniform dose distribution in gamma irradiators," 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0969806X23002049. |
| [3] | I. C. C. D. M. C. Ş. S. C. R. P. Alexandru Vasile a b, "Simulated annealing algorithms used for microstructural design of composites," [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S2214785323027414. |
| [4] | T. B. ,. B. C. ,. R. S. Shyam Marjit, "Simulated annealing aided genetic algorithm for gene selection from microarray data," 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0010482523003190. |