Cuprins

Diploma project summary

Introduction

The microgrids represent the most recent used solution for the rise of the auto-sustainability and reliability of the future distribution electric grids. A contained microgrid has in componence renewable energy sources, a storage system for energy and consumers. The assurance of an adequate management of energy for the microgrid is a difficult problem. There exist different approaches for the solving of this problem.

In this paper there will be presented the development, the implementation and the testing of the algorithm Simulated Annealing used for solving the problem of daily planning.

Theoretical fundamentals

A microgrid is an electro-energetic system of reduced sizes which closes one or more groups of distributed generation that can function independently of the electro-energetic national system or connected to it.

Classically, the electric energy is transferred from the transport grid to the distributed grid and in this process, there are produced losses of power on the chain of grid elements, respectively, through a large number of lines (km) and transformers. The development of the low power sources functioning either on conventional fuel, either using renewable sources, allowed the increase of the number of generators which are present in the distributed grid. The existence of some electrical energy sources in an area of the distribution grid lead to the idea of an architecture of a microgrid. A microgrid is used because:

- The cost of the energy can be reduced (comparing to the received energy, from the main electro-energetic system)
- The reliability and the quality of the energy can be increased
- There can be a rise in efficiency and a reduction in the particulate matter emissions
- It can be the only option if the new or modernized transport infrastructure cannot be developed with a reduced cost or a reduced time

The features of a microgrid

- The peak value of the electrical energy consumption: 1kW – 100 MW
- Consumption of thermal energy: 0.5 MJ/h – 1000 MJ/h
- The number of supplied consumers: 1 – 50.000
- The type of consumers: residential, commercial or industrial
- The geographic spread: from a house until 10 kms
- The mixt functioning: the microgrid can be configured to switch between contained functioning and un-contained, depending on the state of the public grid
- The contained functioning: the microgrid works independently of the public grid
- The tension level: JT or MT, AC or DC
- Architecture: radial or lopped with one or more generators

The Arhitecture of a microgrid

A microgrid can contain a part of the distribution systems of MT or JT and aggregated consumers derived of one or more GD units. From the point of view of the operational, a microgrid can function being interconnected with the main electro-energetic system through a common connection point, having the possibility of containing itself if there is the case of some defections that can appear in the distribution electrical grid or the transport one.

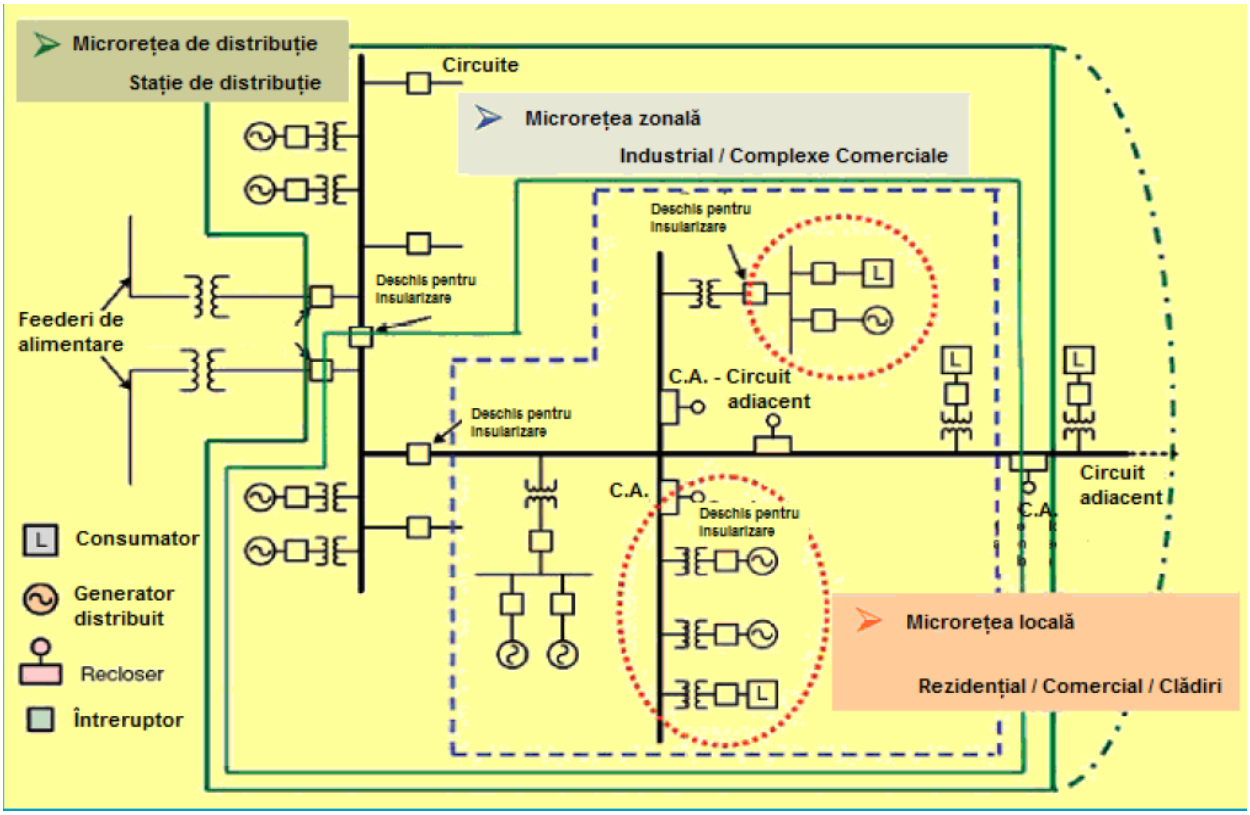


Figure 1. Illustration of Microgrids

While it is connected physically to the main grid, the operation and microgrid control mode can change form a dependent of the main grid mode to an independent of main grid mode, depending on the interchanged power between the microgrid and the main electrical grid.

In Figure 1 there are very revealing presented 3 types of microgrids: the distribution microgrid / industrial (which belong to a distribution company), an area microgrid with singular utility/ multifunctional of industrial type or commercial and a local microgrid meant for the residential sector or small buildings.

Microgrids functioning modes
Connected to the public electrical grid

This functioning mode is a normal mode because, for the assurance of some electrical parameters according to the quality requests of electrical energy, there it is needed the interconnected functioning.

Considering the frequency, this one can be kept very close to the nominal value of 50 Hz, just in conditions of interconnected functioning.

Interconnected functionality offers the necessary conditions to the access of the cheapest sources of electrical energy. Microgrid consumers can choose to use the electrical energy sources inside the microgrid or for acquiring electrical energy form the public grid.

Contained mode, with functionality, independent of the public electrical grid

When in the public grid there occurs an incident, which leads to the interruption of electrical energy supply, the microgrid must be capable of ensure the secondary supply of the main consumers of electrical energy, localized inside the microgrid. The correct functioning in these conditions suppose the existence of a local hub, but also of tune systems for the contained functioning.

The Simulated Annealing algorithm
Introduction

Simulated Annealing is a probabilistic method proposed by Kirkpatrick, Gellet, Vecchi (1983), and Cerny (1985). The idea of this algorithm comes from the analogy between looking for a solution to an optimization problem and the evolution of the states of a solid subjected to a heat treatment that begins by a sudden and continuous heating with a slow cooling. At an elevated temperature, the solid particles (in the liquid phase) are arranged randomly. As the temperature decreases, the particles tend to get in increasingly low energy states. If the temperature drops slowly enough, then for each temperature value, the solid is allowed to reach the so-called thermal balance state.

The operating principle

The basic elements of simulated annealing (SA) are the following:

1) A finite set S.
2) A real-valued cost function J defined on S. Let $S^* \subset S$ be the set of global minima of the function J, assumed to be a proper subset of S
3) For each $i \in S$, a set $S(i) \subset S - \{i\}$, called the set of neighbors of i.
4) For every i, a collection of positive coefficients q_{ij}, $j \in S(i)$, such that $\sum_{j \in S(i)} q_{ij} = 1$. It is assumed that $j \in S(i)$ if and only if $i \in S(j)$
5) A nonincreasing function $T: N \rightarrow (0, \infty)$, called the cooling schedule. Here N is the set of positive integers, and T(t) is called the temperature at time t.
6) An initial "state" $x(0) \in S$

Given the above elements, the SA algorithm consists of a discrete-time inhomogeneous Markov chain x(t), whose evolution we now describe. If the current state x(t) is equal to i, choose a neighbor j of i at random; the probability that any particular $j \in S(i)$ is selected is equal to q_{ij}. Once j is chosen, the next state x(t + 1) is determined as follows:

- If $J(j) \leq J(i)$, then $x(t + 1) = j$
- If $J(j) > J(i)$, then $x(t + 1) = j$ with probability $\exp[-(J(j) - J(i)) / T(t)]$, $x(t + 1) = i$ otherwise.

Formally, $P[x(t + 1) = j | x(t) = i]$

$$= q_{ij} \exp\left[-\frac{1}{T(t)} max\{0, J(j) - J(i)\}\right] \quad (1)$$

if $j \neq i$, $j \in S(i)$.

If $j \neq i$ and $j \notin S(i)$, then $P[x(t + 1) = j \mid x(t) = i] = 0$.

The rationale behind the SA algorithm is best understood by considering a homogeneous Markov chain x_T(t), in which the temperature T(t) is held at a constant value T. Let us assume that the Markov chain x_T(t) is irreducible and aperiodic and that $q_{ij} = q_{ji}$ for all i, j. Then x_T(t) is a reversible Markov chain, and its invariant probability distribution is given by:

$$\pi_T(\mathrm{i}) = \frac{1}{Z_T}\exp\left[-\frac{J(i)}{T}\right],\ \mathrm{i} \in \mathrm{S} \qquad (2)$$

Where Z_T is a normalizing constant. It is then evident that as $T \downarrow 0$, the probability distribution π_T is concentrated on the set S^* of global minima of J. **.** This latter property remains valid if the condition $q_{ij} = q_{ji}$ is relaxed (Faigle and Kern, 1989).

The probability distribution (2), known as the Gibbs distribution, plays an important role in statistical mechanics. In fact, statistical physicists have been interested in generating a sample element of S, drawn according to the probability distribution π_T. This is accomplished by simulating the Markov chain x_T(t) until it reaches equilibrium, and this method is known as the Metropolis algorithm. In the optimization context, we can generate an optimal element of S with high probability if we produce a random sample according to the distribution π_T, with T very small. One difficulty with this approach is that when T is very small, the time it takes for the Markov chain x_T(t) to reach equilibrium can be excessive. The SA algorithm tries to remedy this drawback by using a slowly decreasing "cooling schedule" T(t).

The SA algorithm can also be viewed as a local search algorithm in which there are occasional "upward" moves that lead to a cost increase. One hopes that such upward moves will help escape from local minima.

Theoretical Fundamentals
Microgrid Description

The schematic diagram of the proposed MG with 230V/50Hz is presented in Fig. 2. The characteristics of the three DERs and the storage system used are the following:

1) 12 - 250 Wp photovoltaic panels, manufactured by ET Solar (ET-P660250WW); the SMA Sunny-Boy 3600 inverter, a grid-following unit that converts sunlight into electricity; the PV modules are connected in series;
2) a geothermal generator with a maximum power output of 3kW;
3) a biomass generator with a maximum power output of 3kW;
4) storage units: 8 - 250Ah, 12V VRLA batteries; the Sunny Island 6.0H master inverter, a bidirectional grid-forming unit, that can set the grid voltage and frequency and is used for the charging/discharging of the batteries.

The communication between the presented units and the master PC is realized by an RS-485 network.

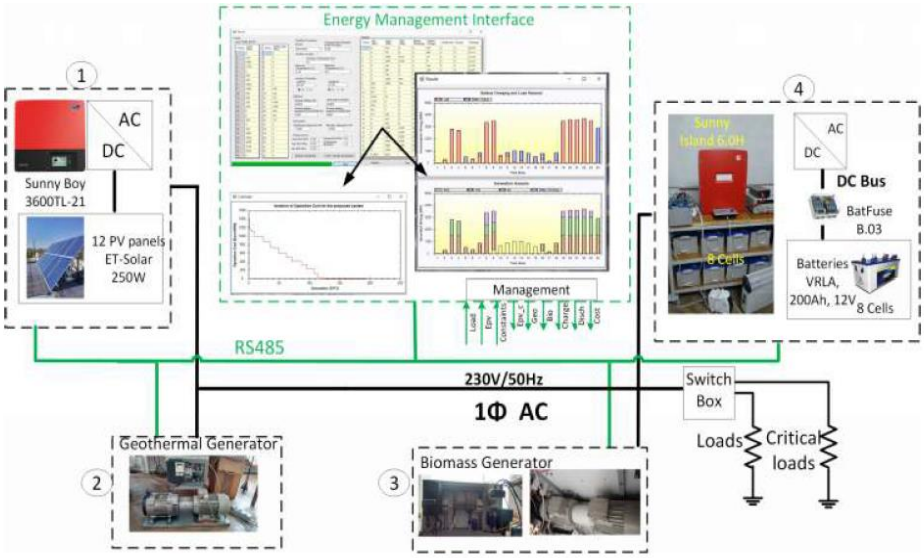


Figure 2. The proposed microgrid

The organization chart of the simulated annealing algorithm

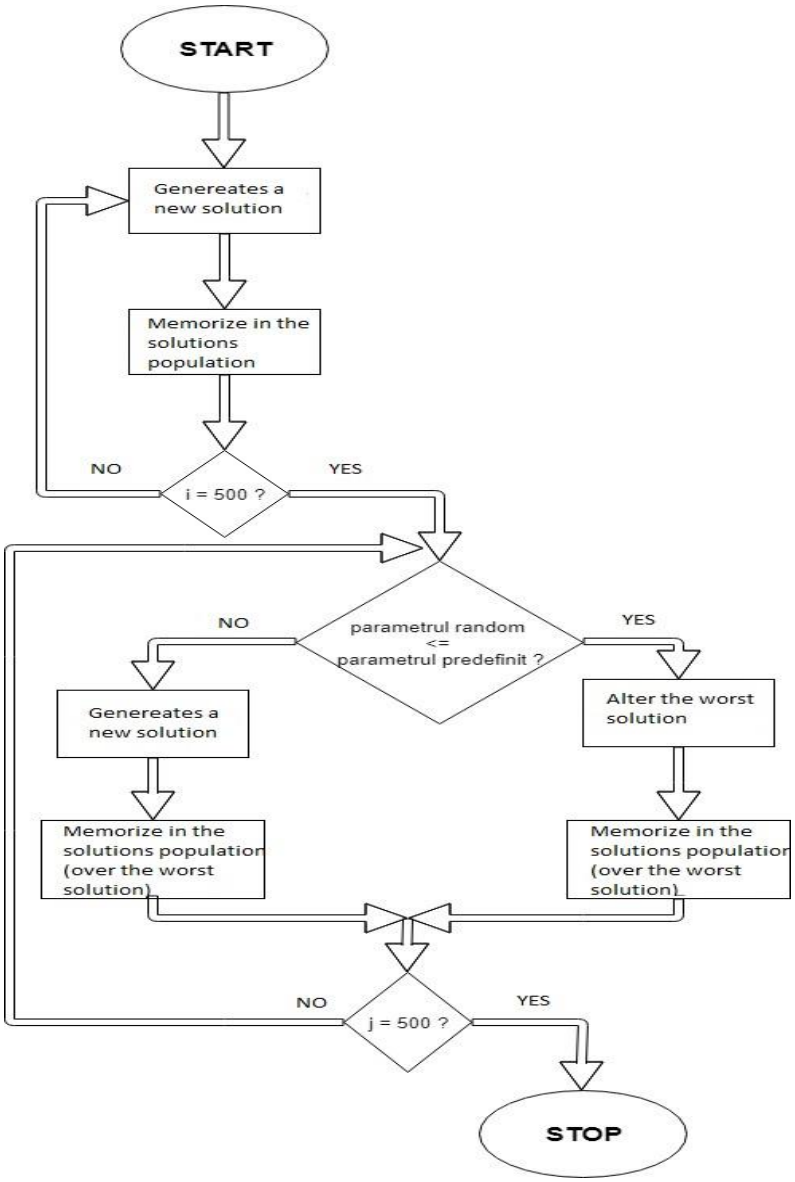


Figure 3. The organization chart of the simulated annealing algorithm

Experimental Results

- Optimal cost calculated by the Simulated Annealing algorithm for January: 247.05 €
- Optimal cost calculated by the Simulated Annealing algorithm for February: 45.25 €
- Optimal cost calculated by the Simulated Annealing algorithm for March: 30.5643 €
- Optimal cost calculated by the Simulated Annealing algorithm for April: -689.65 €
- Optimal cost calculated by the Simulated Annealing algorithm for May: -1585.35 €
- Optimal cost calculated by the Simulated Annealing algorithm for June: -1437.25 €
- Optimal cost calculated by the Simulated Annealing algorithm for July: -2452.75 €
- Optimal cost calculated by the Simulated Annealing algorithm for August: -1884.15 €
- Optimal cost calculated by the Simulated Annealing algorithm for September: -1209.8 €
- Optimal cost calculated by the Simulated Annealing algorithm for October: -84.8 €
- Optimal cost calculated by the Simulated Annealing algorithm for November: 56.40 €
- Optimal cost calculated by the Simulated Annealing algorithm for December: 220.40 €

Conclusions

Table 1. Costs for the three processes used to solve the problem of daily planning of an isolated microgrid

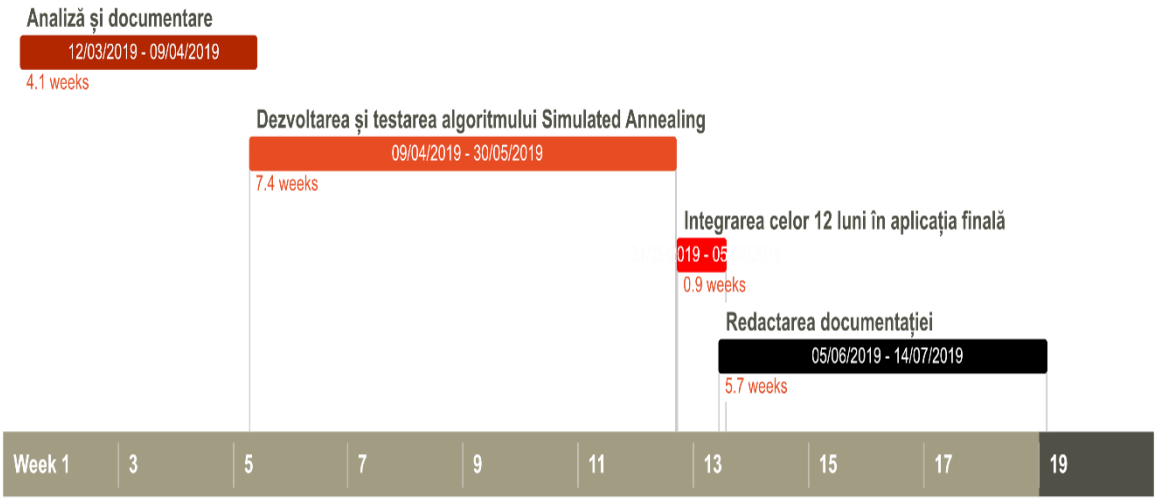
Month	Harmony Search Optimization Algorithm (€)	Particle Swarm Optimization Algorithm (€)	Simulated Annealing Algorithm (€)
January	6.3847	6.391	247.05
Februaie	6.38715	6.3882	45.25
March	6.14945	6.13845	30.5643
April	5.98865	6.12471	-689.65
May	5.63315	5.614	-1585.35
June	5.4915	5.4811	-1437.25
July	5.48825	5.47914	-2452.75
August	5.6911	5.84621	-1884.15
September	6.04065	6.032	-1209.8
October	6.3674	6.489	-84.8
November	6.3632	6.365	56.40
December	6.694	6.681	220.40

It can be seen from Table 1 that the version of the Simulated Annealing algorithm developed, implemented and tested in this diploma paper has a much higher cost in some months of the year than in the other methods used. Instead, in April, May, June, August, September and October the cost ,after executing the Simulated Annealing algorithm, is negative.

The results obtained using the Simulated Annealing algorithm show us that it is a viable solution to solve the problem of daily planning of an isolated microgrid. The load demand is satisfied by the renewable energy sources used.

The advantage of this algorithm is its execution time. The Simulated Annealing algorithm makes all the calculations necessary in less than an hour. Instead, the disadvantage of this version is that besides loading the available batteries, it also produces an excess of energy.

1 Planificarea activității



2 Stadiul actual

Microrețelele reprezintă cea mai recentă soluție utilizată pentru creșterea autosustenabilității și fiabilității viitoarelor rețele electrice de distribuție. O microrețea izolată dispune de surse de energie regenerabile, un sistem de stocare a energiei și consumatori. Asigurarea unui management al energiei adecvat pentru microrețele este o problemă dificilă. Există abordări diferite pentru rezolvarea acestei probleme. În acest capitol se vor prezenta trei algoritmi folosiți în managementul energiei pentru microrețele, respectiv câteva idei despre algoritmul folosit pentru realizarea lucrării de diplomă.

Primul algoritm folosit pentru rezolvarea problemei planificării zilnice este Harmony Search Optimization. Acest procedeu a fost dezvoltat de Geem și colaboratorii în [1], având aceeași structură de bază ca și alți algoritmi metaeuristici de căutare, cu caracteristici cum ar fi păstrarea istoriei vectorilor anteriori, variind rata lor de adaptare în timpul procesului de căutare (similar algoritmului Simulated Annealing) și luând în considerare mai mulți vectori simultan. Ceea ce îl face diferit este noua metodă de generare a soluției: noul vector este generat de toți vectorii existenți, algoritmul nu necesită modele de probabilitate pentru a estima o distribuire a unor noi valori promițătoare și noile valori sunt selectate din memorie, precum și eventualele intervale de valori [2].

Harmony Search a fost inspirat din procesul în care muzicienii o utilizează pentru a obține o armonie plăcută. În acest caz, calitatea estetică a muzicii este înlocuită cu funcția obiectiv. Deoarece pitch-ul fiecărui instrument influențează estetica generală a muzicii, ele sunt reprezentate de variabilele de decizie a problemei de optimizare, formând împreună vectorul de soluție. În procesul lor de improvizație muzicienii au trei opțiuni: să folosească un pitch din memorie, să adapteze ușor un pitch din memorie sau să utilizeze un pitch aleatoriu din intervalul posibil. Prin urmare, în procesul de optimizare, o variabilă de decizie poate primi o valoare din memoria armonică, poate primi o valoare din memoria armonică care a fost ușor ajustată sau poate primi o valoare aleatorie din intervalul posibil definit la începutul algoritmului de optimizare. Există trei parametri care influențează performanța algoritmului: rata de luare în considerare a armoniei, rata de ajustare a pitch-ului și lățimea de bandă. Probabilitatea alegerii unei armonii din memoria armonică este dată de către rata de luare în considerare a armoniei. Rata de ajustare a pitch-ului reprezintă probabilitatea ajustării unei armonii din memoria armonică. Lățimea de bandă este folosit pentru a obține un echilibru între un proces local de optimizare și unul global. După ce variabilele de decizie primesc noi valori și se formează un nou vector de soluție posibil, se evaluează folosind funcția obiectiv. Dacă noul vector de soluție îndeplinește toate constrângerile și este mai bun decât cel mai rău vector de soluție din memoria armonică, atunci acesta din urmă va fi înlocuit. În Figura 1 sunt prezentați pașii folosiți în algoritmul Harmony Search Optimization. Algoritmul se termină atunci când se validează condiția de stop, de obicei fiind numărul maxim de iterații.

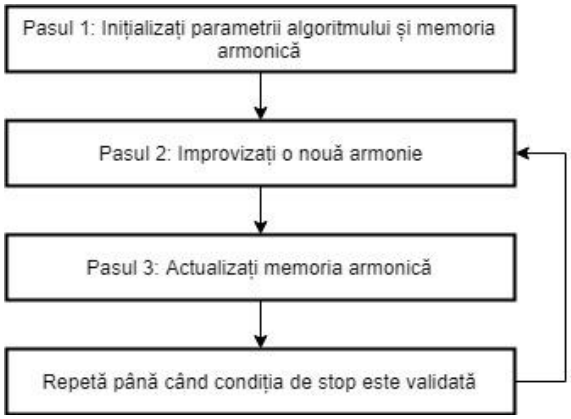


Figura 1. Pașii utilizați de algoritmul Harmony Search Optimization

Rezultatele obținute prin intermediul acestui algoritm arată că este o soluție viabilă pentru programarea zilnică a unei microrețele izolate. Cererea de sarcină este satisfăcută de energiile regenerabile utilizate și starea de încărcare din baterii este în limitele predefinite. În plus, se încarcă bateriile disponibile fără a produce o cantitate mai mare de energie care nu poate fi stocată.

A doua procedură utilizată este Particle Swarm Optimization. Dezvoltat de Kennedy și Eberhart în [3], Particle Swarm Optimization este o căutare stocastică bazată pe populație, fiind inspirat din comportamentul natural al unui stol de păsări sau al unui banc de pești pentru căutarea hranei lor. Similar cu alți algoritmi de căutare metaeuristici, Particle Swarm Optimization începe cu o populație generată aleatoriu de soluții posibile și converge în mod ideal spre soluția optimă globală a problemei. Spre deosebire de alte tehnici de optimizare, algoritmul Particle Swarm Optimization adaptează fiecare dintre soluțiile sale posibile bazate pe cel mai bun rezultat al acestora, precum și pe cel mai bun rezultat general al întregii populații. Parametrii utilizați pentru generarea de noi soluții variază de la o versiune a algoritmului la alta [4].

Algoritmul Particle Swarm Optimization încearcă să imite comportamentul animalelor în căutarea lor pentru hrană. Stolul de păsări sau bancul de pești este reprezentat de către populație, fiecare particulă preluând rolul unui individ. Funcția obiectiv reprezintă sursa de hrană. Dimensiunea problemei este dată de către numărul de variabile de decizie. În căutarea hranei animalele își adaptează poziția atât individual, cât și colectiv. Prin urmare, în procesul de optimizare, atât cea mai bună poziție a particulei cât și cea mai bună poziție a întregii populații sunt luate în considerare atunci când se generează noi soluții posibile. Algoritmul începe cu o mulțime inițială de particule generate aleatoriu în spațiul de căutare. Particulele sunt inițializate fiecare cu o poziție aleatorie și o viteză aleatorie. Pe măsură ce se mișcă populația, fiecare particulă este actualizată cu o nouă poziție și o nouă viteză. Viteza nouă a particulelor se calculează pe baza vitezei sale curente, a celei mai bune poziții și a celei mai bune poziții a mulțimii. Poziția nouă se calculează utilizând poziția curentă și viteza nouă calculată. Când se calculează noua viteză și poziție pentru fiecare particulă se utilizează trei parametri: greutatea inerției, greutatea cognitivă și greutatea socială. Greutatea inerției reprezintă cantitatea de influență pe care o are viteza curentă asupra celei următoare. Greutatea cognitivă și greutatea socială reprezintă influența pe care poziția cea mai bună a particulei și poziția cea mai cunoscută a mulțimi o au în calcularea noii poziții a particulelor. Cea mai bună poziție atinsă de fiecare particulă, precum și de întreaga mulțime sunt actualizate cu fiecare iterație. În Figura 2 putem observa pașii utilizați de către algoritm. De obicei, când se atinge numărul maxim de iterații, algoritmul Particle Swarm Optimization se termină.

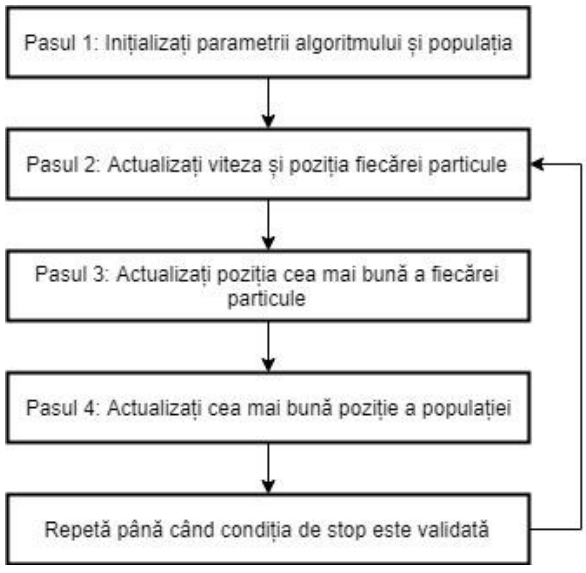


Figura 2. Pașii utilizați de algoritmul Particle Swarm Optimization

Rezultatele obținute prin utilizarea algoritmului Particle Swarm Optimization arată că folosirea acesteia pentru programarea zilnică a unei microrețele izolate este o soluție viabilă. Nu numai că rezultatele sunt comparabile cu cele obținute folosind alți algoritmi de optimizare metaeuristică, dar și timpul de calcul este mai scurt. Elementele funcției microrețelei în cadrul parametrilor definiți și cerințele energetice sunt asigurate de surse regenerabile de energie.

Al treilea procedeu folosit pentru rezolvarea acestei probleme este algoritmul genetic. Algoritmii genetici sunt tehnici adaptive de căutare euristică, bazate pe principiile geneticii și ale selecției naturale, enunțate de Darwin ("supraviețuiește cel care e cel mai bine adaptat"). Mecanismul este similar procesului biologic al evoluției. Acest proces posedă o trăsătură prin care numai speciile care se adaptează mai bine la mediu sunt capabile să supraviețuiască și să evolueze peste generații, în timp ce acelea mai puțin adaptate nu reușesc să supraviețuiască și cu timpul dispar, ca urmare a selecției naturale. Probabilitatea ca specia să supraviețuiască și să evolueze peste generații devine cu atât mai mare cu cât gradul de adaptare crește, ceea ce în termeni de optimizare înseamnă că soluția se apropie de optim.

Un algoritm genetic este un model informatic care emulează modelul biologic evoluționist pentru a rezolva probleme de optimizare ori căutare. Acesta cuprinde un set de elemente individuale reprezentate sub forma unor șiruri binare (populația) și un set de operatori de natură biologică definiți asupra populației. Cu ajutorul operatorilor, algoritmii genetici manipulează cele mai promițătoare șiruri, evaluate conform unei funcții obiectiv, căutând soluții mai bune. Algoritmii genetici au început să fie recunoscuți ca tehnici de optimizare odată cu lucrările lui John Holland.

Ca aplicații practice, algoritmii genetici sunt cel mai adesea utilizați în rezolvarea problemelor de optimizare, planificare ori căutare. Condiția esențială pentru succesul unei aplicații cu agenți inteligenți este ca problema de rezolvat să nu ceară obținerea soluției optime, ci să fie suficientă și o soluție apropiată de optim.

A patra metodă folosită în programarea zilnică a unei microrețele izolate este algoritmul Simulated Annealing, fiind și procedeul folosit pentru realizarea acestei lucrări de diplomă. Este o tehnică probabilistică pentru aproximarea optimului global al unei funcții date. Mai exact, este o metodă euristică de a aproxima optimizarea globală într-un spațiu de căutare mare pentru o problemă de optimizare. Este adesea folosit atunci când spațiul de căutare este discret. Pentru problemele în care găsirea unui optim global aproximativ este mai importantă decât găsirea unui optim local precis într-o anumită perioadă de timp, Simulated Annealing poate fi preferabilă alternativelor, cum ar fi coborârea gradientului.

Numele și inspirația provin din călirea în metalurgie, o tehnică care implică încălzirea și răcirea controlată a unui material pentru a mări dimensiunea cristalelor sale și a reduce defectele lor. Ambele sunt atribuite materialului, depinzând de energia liberă termodinamică. Încălzirea și răcirea materialului afectează atât temperatura cât și energia liberă termodinamică. Acest algoritm poate fi folosit pentru a găsi o aproximare a unui minim global pentru o funcție cu un număr mare de variabile. Această noțiune de răcire lentă implementată în Simulated Annealing este interpretată ca o scădere lentă a probabilității de acceptare a soluțiilor mai slabe pe măsură ce spațiul soluției este explorat. Acceptarea soluțiilor mai slabe este o proprietate fundamentală a metaeuristici, deoarece permite o căutare mai amplă a soluției optime globale. În general, algoritmii Simualted Annealing funcționează după cum urmează: la fiecare etapă, algoritmul selectează aleatoriu o soluție apropiată de cea actuală, măsoară calitatea sa și apoi decide să se mute la ea sau să rămână cu soluția actuală pe baza oricăreia dintre cele două probabilități între care alege pe baza faptului că noua soluție este mai bună sau mai rea decât cea actuală. În timpul căutării, temperatura este scăzută treptat de la o valoare pozitivă inițială la zero și afectează cele două probabilități: la fiecare etapă, probabilitatea de a trece la o soluție mai bună este fie menținută la 1, fie este schimbată spre o valoare pozitivă. Pe de altă parte, probabilitatea de a trece la o soluție mai rea nouă este treptat schimbată spre zero.

3 Fundamentare teoretică

3.1 Microrețeaua

3.1.1 Introducere

O microrețea este un sistem electroenergetic de dimensiuni mici ce include una sau mai multe grupuri de generare distribuită care pot funcționa independent de sistemul electroenergetic național sau conectat cu acesta.

În mod clasic, energia electrică este transferată dinspre rețeaua de transport spre rețeaua de distribuție, astfel că se produc pierderi de putere pe lanțul de elemente de rețea, respectiv printr-un număr mare de linii (exprimat în km) și transformatoare. Dezvoltarea surselor de putere mică, funcționând fie pe combustibil convențional, fie folosind surse regenerabile, a permis creșterea numărului de generatoare prezente în rețeaua de distribuție. Existența unor surse de energie electrică într-o zonă de rețea de distribuție a condus la ideea de arhitectură de microrețea. O microrețea se utilizează deoarece:

- Costul energiei se poate micșora (comparativ cu energia primită din sistemul electroenergetic principal)
- Fiabilitatea și calitatea energiei se pot îmbunătăți
- Poate crește eficiența și se reduc emisiile de noxe
- Poate fi singura opțiune dacă infrastructura de transport nouă sau modernizată nu poate fi dezvoltată într-un timp sau cu un cost eficace

3.1.2 Scurt istoric

În perioada de început a dezvoltării sistemelor electroenergetice (1880 – 1910), acestea aveau o structură simplă de microrețea. O singură centrală electrică alimenta o zonă de consum. Ulterior, ideea de funcționare izolată, sub forma unei microrețele, a început să dispară în perioada 1910 – 1950 prin politica de interconectare a rețelelor electrice, datorită avantajelor pe care acestea le prezintă. În prezent, ideea de microrețea a început să prezinte un nou sens ca urmare a dezvoltării de noi tehnologii, existența restricțiilor privind construirea de noi elemente în rețelele de transport și distribuție, aspectelor legate de mediu, cerințelor legate de fiabilitate.

Societatea modernă se bazează pe un sistem de furnizare a energiei electrice de înaltă fiabilitate. Problemele actuale privind disponibilitatea energiei primare, îmbătrânirea infrastructurii de transport și distribuție a rețelelor electrice, necesitatea de a instala surse noi de producție (cum ar fi sursele regenerabile) și vânzarea energiei electrice prin piețele angro constituie o provocare pentru operatorii de sistem în ceea ce privește securitatea, siguranța și calitatea. De aceea sunt necesare investiții importante pentru dezvoltarea și modernizarea infrastructurii electrice, iar cel mai eficient mod pentru a răspunde cerințelor sociale va fi încorporarea de soluții inovatoare, tehnologii și arhitecturi de rețele.

Viitoarele rețele electrice vor trebui să se adapteze la schimbările tehnologice, să răspundă valorilor societății privind mediul înconjurător și aspectul comercial. Astfel, securitatea sistemului, siguranța, mediul înconjurător, calitatea energiei electrice, costul furnizării și eficiența energetică sunt evaluate într-o manieră nouă, ca răspuns la schimbarea cerințelor de pe piață. Tehnologia trebuie să demonstreze siguranță, rezistență și eficiență. La nivelul distribuției, noile cerințe pentru dezvoltare sunt:

- Rețelele de distribuție trebuie să prezinte accesibilitate pentru producerea distribuită (GD) și sursele de energie regenerabile (SER), care pot fi auto-dispecerizate sau dispecerizate prin intermediul unui dispecer local

- Rețelele de distribuție care permit managementul consumului local și care interacționează cu utilizatorii finali prin sisteme de contorizare inteligente

Rețelele de distribuție încep să se transforme din rețele pasive în rețele active în sensul în care luarea unei decizii și controlul sunt distribuite, iar puterea circulă bidirecțional. Acest tip de rețele cu participarea generării distribuite, a surselor de energie regenerabile și a dispozitivelor de stocare, oferă soluții pentru noi tipuri de echipamente și servicii, fiecare din acestea fiind necesar să respecte standardele și protocoalele comune. Funcția unei rețele de distribuție active este să interconecteze în mod eficient sursele de putere și consumatorii, permițându-le ambelor părți să decidă care este cel mai bun mod de funcționare în timp real. Determinarea circulației de puteri, controlul tensiunii și sistemele de protecții necesită tehnologii de cost competitiv și sisteme de comunicare noi, care încorporează tehnologii comunicaționale și informatice.

Crearea rețelelor de distribuție active vor permite în mod radical dezvoltarea de noi concepte. Probabil că cel mai promițător concept este acela de microrețele. Microrețelele cuprind sisteme de distribuție de joasă tensiune (LV) cu surse distribuite de energie (DERs) cum ar fi microturbinele, pile cu combustibil, celule fotoelectrice, precum și dispozitive de stocare (baterii cu condensatoare de dimensiuni mari, baterii de stocare de dimensiuni mici), respectiv consumatori controlabili, oferind posibilități de control considerabile în ceea ce privește funcționarea rețelei electrice.

Aceste sisteme sunt conectate la rețeaua de distribuție de medie tensiune, dar pot funcționa și izolat de rețeaua principală în cazul unor defecte în rețea. Din punctul de vedere al consumatorilor, microrețelele asigură atât energie termică cât și energie electrică, iar în plus îmbunătățesc siguranța în funcționare la nivel local, reduc emisiile de noxe, îmbunătățesc calitatea energiei prin controlul tensiunii și reducerea golurilor de tensiune, pot oferi costuri reduse pentru furnizarea energiei electrice sau termice. Din punctul de vedere al rețelelor, o microrețea poate fi privită ca o entitate controlabilă în cadrul sistemului energetic care poate să funcționeze ca o mică sursă de putere sau ca servicii auxiliare, sprijinind rețeaua principală.

Potențialul economic cheie al aplicațiilor GD la premisele de client constă în oportunitatea de utilizare locală a căldurii evacuate de la conversia combustibilului primar în electricitate. În ultimii ani au existat progrese remarcabile privind dezvoltarea de aplicații de cogenerare de mică putere. Aceste sisteme se așteaptă să joace un rol important în microrețelele din țările nordice. Pe de altă parte, sistemele PV se anticipează că se vor dezvolta semnificativ în țările cu climat însorit. Aplicații de microcogenerare și PV au un potențial de creștere a eficienței totale a utilizării surselor de energie primară și în consecință, reduce substanțial emisiile de carbon și noxe, constituind un alt beneficiu important în eforturile omenirii de a combate schimbarea climatică.

Din punctul de vedere al rețelei electrice, aplicațiile GD pot reduce necesitatea de noi investiții în rețelele electrice de distribuție și de transport. Generatoarele distribuite amplasate în apropierea consumatorilor vor decongestiona căile de alimentare și vor conduce la reducerea pierderilor de putere și posibilitatea de rezervare a unor elemente din rețelele principale. Microrețelele pot, de asemenea, să asigure servicii de sistem în cazul unor probleme ce apar în rețeaua electrică principală.

În consecință, microrețelele sunt subiectul unei cercetări intense și activități de propagare în America, Japonia, Europa, Canada pentru a furniza soluțiile eficiente și a demonstra conceptele de funcționare ale microrețelelor.

3.1.3 Caracteristici ale unei microrețele

- Valoare de vârf a consumului de energie electrică: 1kW – 100 MW
- Consum de energie termică: 0.5 MJ/h – 1000 MJ/h
- Numărul de consumatori alimentați: 1 – 50.000
- Tipul consumatorilor: rezidențial, comercial sau industrial
- Întinderea geografică: de la o casă până la 10 kmp

- Funcționarea mixtă: microrețeaua poate fi configurată pentru a comuta între funcționarea "izolată" și "ne-izolată" în funcție de starea rețelei publice
- Funcționarea izolată: microrețeaua funcționează independent de rețeaua publică
- Nivelul tensiunii: JT sau MT, AC sau DC
- Arhitectura: radială sau buclată cu una sau mai multe generatoare

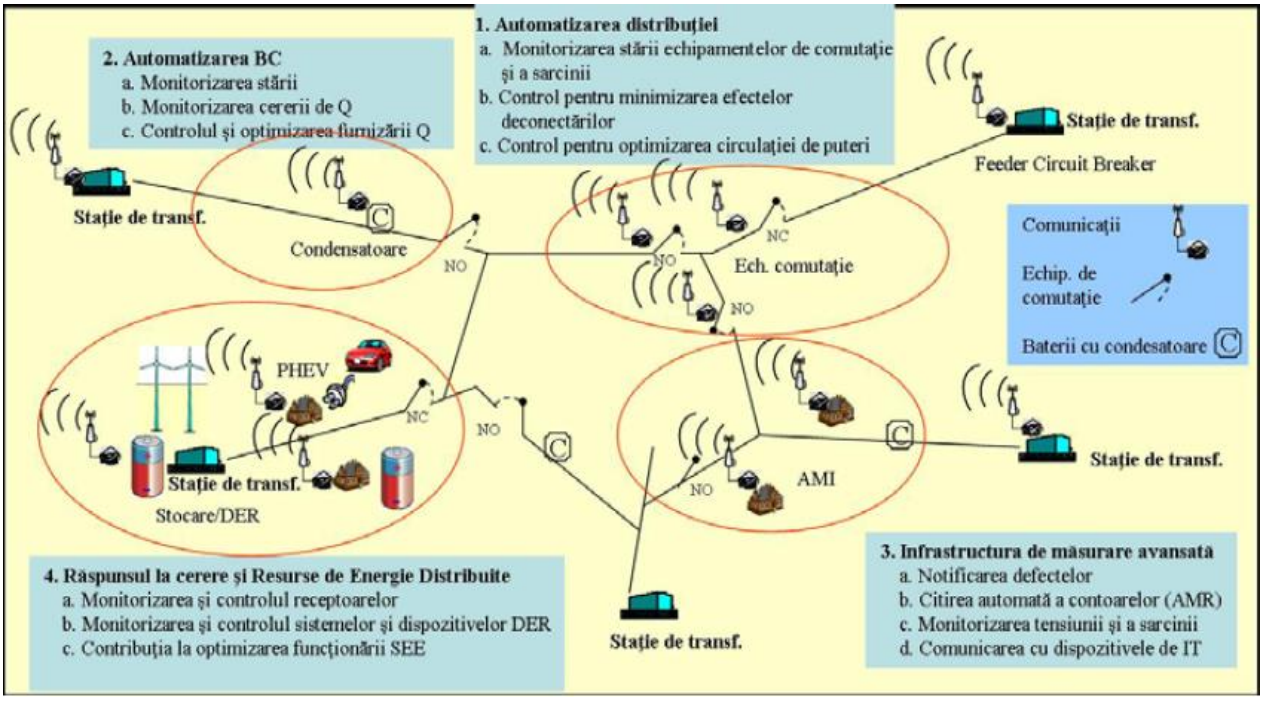


Figura 3. Funcțiile unei microrețele [16]

Pe lângă infrastructura clasică a unei rețele electrice, ce cuprinde linii, transformatoare, sisteme de protecții și automatizări, o microrețea include consumatori inteligenți generatoare distribuite, sisteme de detectare a defectelor avansate, echipamente de comutație inteligente, o infrastructură de măsurare avansată, căi de alimentare de rezervă, precum și un sistem de monitorizare și control ce cuprinde produse informatice și cu ajutorul cărora se pot îndeplini următoarele obiective:

- Izolarea microrețelei în cazul producerii unui blackout în SEN
- Optimizarea costului energiei electrice, comparativ cu cel obținut pe piața de energie electrică, folosind resursele proprii, atât generatoarele electrice și sursele de stocare, precum și consumatorii activi
- Îmbunătățirea fiabilității și calității energiei electrice prin posibilitatea de reconfigurare a rețelei electrice
- Creșterea eficienței și reducerea emisiilor de substanțe poluante prin integrarea surselor regenerabile de energie, precum panourile fotovoltaice, micro-hidrocentralele, turbinele eoliene

O microrețea se poate dezvolta în cadrul unei rețele electrice de distribuție (RED), de medie tensiune sau de joasă tensiune.

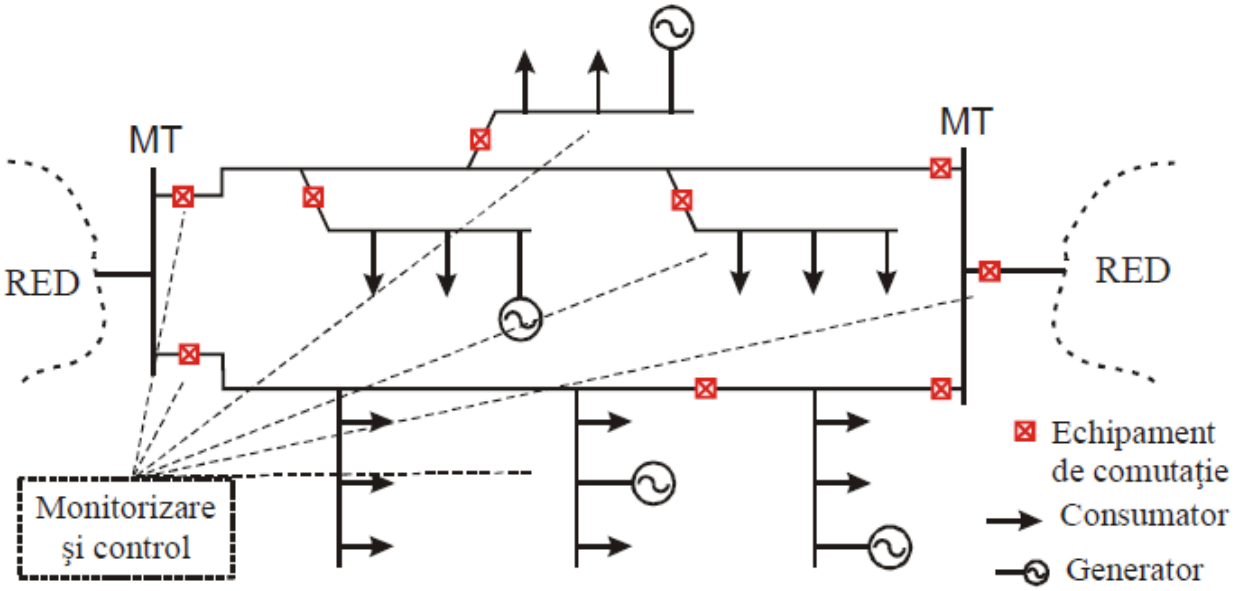


Figura 4. Reprezentarea de bază a unei microrețele într-o rețea de distribuție [16]

În cazul zonelor izolate, microrețelele pot constitui singura opțiune din punct de vedere tehnic sau economic. În plus, alimentarea cu energie electrică la tensiune continuă poate reprezenta o soluție mult mai eficientă decât alimentarea la tensiune alternativă. O atenție deosebită trebuie acordată tensiunii continue și electronicii de putere pentru conectarea la rețeaua electrică a unor surse care produc energie electrică la tensiune continuă (C.C).

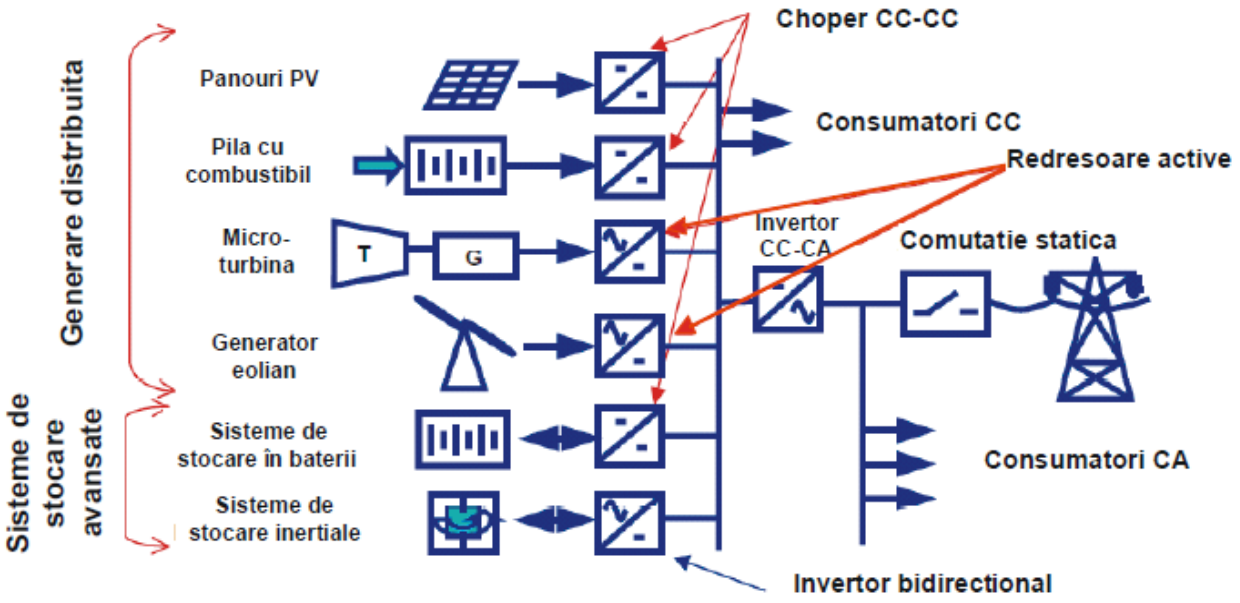


Figura 5. Conectarea la rețeaua electrică a surselor de energie electrică [16]

3.1.4 Arhitectura Microrețelelor

O microrețea poate cuprinde o parte din sistemele de distribuție de MT și JT și consumatori agregați deserviți de unul sau mai multe unități de GD. Din punct de vedere al operaționalului, o microrețea poate funcționa interconectat cu sistemul electroenergetic principal prin intermediul

unui punct comun de conectare (PCC), având astfel posibilitatea de a se izola în cazul unor defecte ce apar în rețeaua electrică de distribuție sau transport.

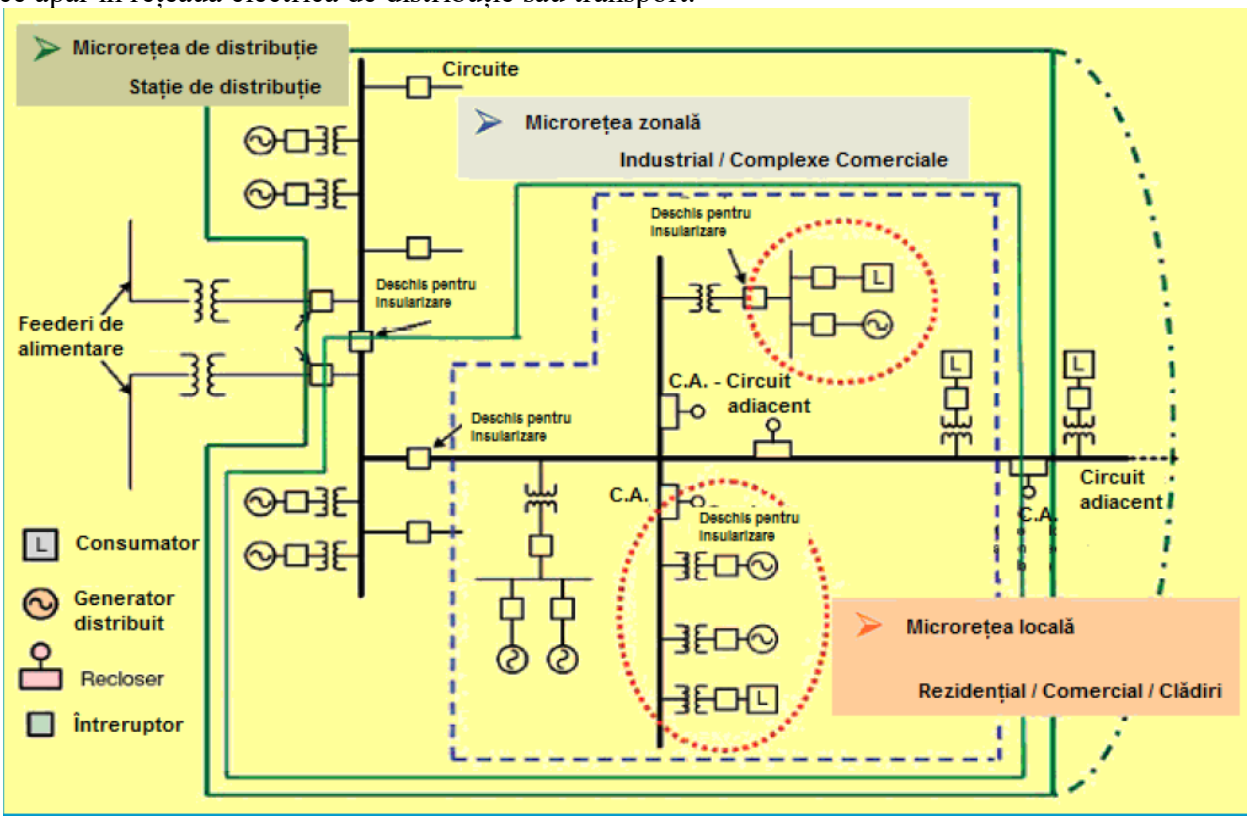


Figura 6. Ilustrarea microrețelelor [16]

În timp ce este fizic conectată la rețeaua principală, modul de operare și control al microrețelei poate să se schimbe de la mod dependent de rețeaua principală la mod independent de rețeaua principală, în funcție de puterea interschimbată între microrețea și rețeaua electrică principală.

În Figura 6 se prezintă sugestiv 3 tipuri de microrețele și anume: microrețea de distribuție / industrială (care aparține unei companii de distribuție), microrețea zonală cu utilitate singular / multifuncțională de tip industrial sau comercial și microrețea locală destinată sectorului rezidențial sau clădirilor mici.

3.1.4.1 Microrețele de distribuție

Accesul la microrețele poate facilita pe scară largă utilizarea SRE și / sau încorpora sisteme de cogenerare (CHP) în rețeaua de distribuție, amortizând în același timp fluctuațiile de energie ale rețelei principale. Microrețelele pot fi formate dintr-o parte sau de către toate fiderele care pleacă dintr-o stație de distribuție, fiind condusă de un operator de distribuție. Utilizarea unui număr mare de generatoare distribuite localizate aproape de centrele de consum, o microrețea industrială poate asigura echilibrul local între puterile active generate și consumate, respectiv pot contribui la evitarea unor congestii pe toate fiderele de medie tensiune. La nivelul companiei de distribuție ce are în proprietate microrețeaua, turbine hidro de putere mică, celule fotoelectrice, grupurile eoliene și sistemele de biomasă sunt câteva din sursele de energie alternativă care pot fi utilizate cu emisii reduse de noxe. O microrețea de utilitate publică poate să fie deconectată de la rețeaua principală în timpul perioadelor programate de întreținere a fiderelor de înaltă tensiune și stațiilor într-un mod controlat. Izolarea planificată a microrețelei limitează durata întreruperilor în alimentarea consumatorilor. O microrețea de utilitate publică poate să ofere și servicii de sistem cum ar fi furnizarea de energie reactivă și creșterea calității energiei electrice.

De asemenea, câteva tehnologii GD pot asigura putere reactivă dispecerizabilă care să compenseze puterea reactivă a sarcinii locale și să mențină nivelul de tensiune. Folosirea surselor de cogenerare (CHP), microrețeaua de utilitate publică poate să ofere energie termică din procesul de producere al energiei electrice sub formă de căldură sau apă fierbinte (sau abur) pentru uz casnic. Conceptul de CHP în cadrul microrețelei este aplicat prin amplasarea optimă a surselor CHP acolo unde echipamentele termice / electrice cresc complet eficiența instalației și reduc consumul de combustibil.

3.1.4.2 Microrețele zonale – industriale și comerciale

Consumatorii comerciali și industriali de energie electrică sunt definiți prin clase de importanță, care se referă la gradul de calitate al energiei electrice. Un consumator critic poate să nu tolereze întreruperile în alimentarea cu energie electrică. O microrețea poate fi folosită cu ușurință pentru a alimenta mai multe tipuri de consumatori industriali sau comerciali, de exemplu: campus universitar, centru comercial, instalații industriale etc. Strategiile de management avansat al sarcinii din cadrul microrețelelor constau într-un control distribuit și automat care să prevină întreruperile neplanificate și deci să contribuie la îmbunătățirea calității energiei electrice prin creșterea independenței microrețelei de rețeaua electrică principală.

Utilizarea conceptului de microrețea, cu un nivel distinct de fiabilitate și calitate a energiei electrice poate fi definit pe baza unei clasificări a consumatorilor și diferențierea serviciilor pentru utilizatori multipli. Clasificarea consumatorilor și controlul acestora în cadrul unei microrețele poate, de asemenea, contribui la aplatizarea curbei de sarcină fie în mod izolat, fie în mod conectat la rețea. O microrețea comercială sau industrială se poate izola atunci când calitatea energiei electrice din rețeaua principală nu satisface cerințele impuse și poate afecta inclusiv calitatea energiei electrice asigurate de microrețea. Funcționarea independentă față de rețeaua principală a unei microrețele comerciale / industriale poate fi planificată, de exemplu, la vârf de sarcină când prețul energiei electrice absorbite din rețeaua principală este ridicat. O microrețea poate alimenta un consumator rezidențial mic, adică un grup de case orășenești. Microrețeaua rezidențială asigură un sistem de furnizare a energiei electrice convenabilă și eficientă, fiind particularizată în funcție de cerințele consumatorilor și generatoarelor distribuite utilizate. Generația de panouri solare și microturbine în cogenerare constituie surse distributive atractive pentru aplicațiile rezidențiale și clădirile comerciale. Sursele PV pot fi încorporate în structura clădirii. Proprietarii clădirii pot beneficia de bună corelare între vârful curbei de sarcină și intensitatea soarelui pentru producerea de energie solară. Modulele la scară redusă de microturbine oferă surse de cogenerare controlabile și eficiente de energie electrică și căldură, cu zgomot mic care pot fi instalate individual în apartamente sau birouri, acolo unde consumul de energie electrică se realizează la o eficiență ridicată. Bazându-ne pe accesul la microrețea, consumul de energie termică și electrică total al rețelelor locale este controlat prin putere adecvată și strategii de management, cu rol de a personaliza prețul energiei și de a limita impactul fluctuațiilor de energie ale surselor intermitente și schimbările bruște ale consumului în rețea.

3.1.4.3 Microrețele locale

Electrificarea comunităților retrase și a zonelor neintegrate în țările dezvoltate și a insulelor geografice este o mare prioritate pentru companiile de utilități din toată lumea. Câteva țări au cercetat adoptarea unui concept de producere de energie descentralizată, microrețeaua, pentru furnizarea de energie în zone izolate. Cererile de energie ale zonelor neintegrate pot fi satisfăcute prin instalarea regenerabilelor și alternativelor DER pentru a forma rețele izolate și microrețele autonome care furnizează energie electrică și eventual căldură sau apă caldă clienților locali sau comerciali.

În funcție de caracteristicile geografice ale unei zone retrase și disponibilitatea resurselor de diverse tipuri cum ar fi: microturbine, mori de vânt, celule fotovoltaice și turbine pe gaz cu emisii

mici, pot fi folosite. O deosebire majoră în modelul microrețelelor depărtate este aceea că producerea trebuie fie dimensionată pentru a servi întregii saricini cu un nivel adecvat al capacității de rezervă. În plus, dispersarea încărcării și marile diferențe între încărcarea minimă și maximă a microrețelei fac din tehnologia selecției, a mărimii DER, un lucru competitiv. Următoarele metode sunt sugerate pentru a realiza echilibrul de energie pe termen scurt sau pe termen lung a microrețelelor retrase menite să învingă fluctuațiile de putere introduse de producerea intermitentă și sarcinii variabile:

- Participare avansată a puterii și angajamentul unităților printr-un set de mai multe surse de generare pentru a selecta combinația potrivită a DER în funcție de variația sarcinii
- Utilizarea mărimii optime a unităților de energie
- Control avansat al sarcinii

3.1.5 Moduri de funcționare a microrețelelor

3.1.5.1 Modul conectat la rețeaua electrică publică

Acest mod de funcționare este un mod normal deoarece, pentru asigurarea unor parametrii electrici în conformitate cu cerințele de calitate a energiei electrice, este necesară funcționarea interconectată.

Din punct de vedere al frecvenței, aceasta poate fi menținută foarte aproape de valoarea nominală de 50 Hz, doar în condițiile funcționării interconectate.

Funcționarea interconectată asigură condițiile necesare accesului la cele mai ieftine surse de energie electrică. Consumatorii din microrețea pot să opteze pentru a folosi sursele de energie electrică din cadrul microrețelei sau pentru a cumpăra energie electrică din rețeaua publică.

3.1.5.2 Modul izolat, cu funcționare independentă de rețeaua electrică publică

În cazul în care în rețeaua publică se produce un incident care conduce la întreruperea alimentării cu energie electrică, microrețeaua trebuie să fie capabilă să asigure alimentarea de rezervă a principalilor consumatori de energie electrică localizați în cadrul microrețelei. Funcționarea corespunzătoare în aceste condiții presupune existența unui dispecer local, dar și a unor sisteme de reglaj pentru funcționarea izolată.

3.2 Algoritmul Simulated Annealing

3.2.1 Introducere

Simulated Annealing este o metodă probabilistică propusă de către Kirkpatrick, Gellet, Vecchi (1983) și Cerny (1985). Ideea acestui algoritm provine de la analogia dintre căutarea soluției unei probleme de optimizare și evoluția stărilor unui solid supus unui tratament termic care începe printr-o încălzire bruscă și continuă cu o răcire lentă. La o temperatură ridicată particulele solidului (aflat în faza lichidă) se aranjează aleatoriu. Pe măsură ce temperatura scade, particulele tind să ajungă în stări cu energie din ce în ce mai mică. Dacă temperatura scade suficient de lent, atunci pentru fiecare valoare a temperaturii, solidului îi este permis să atingă starea de așa-numită echilibru termic.

3.2.2 Principiul de funcționare

Elementele principale ale algoritmului Simulated Annealing sunt următoarele:

7) O mulțime finită S
8) O funcție de cost reală J definită pe S. Fie $S^* \subset$ S mulțimea minimelor globale ale funcției J, presupus a fi o submulțime corespunzătoare a lui S
9) Pentru fiecare i ϵ S, o mulțime S(i) ⊂ S – {i}, numit mulțimea vecinilor lui i
10) Pentru fiecare i, o colecție de coeficienții pozitivi q_{ij}, j ϵ S(i), astfel încât $\sum_{j \epsilon S(i)} q_{ij}$ = 1. Se presupune că j ϵ S(i) dacă și numai dacă i ϵ S(j)
11) O funcție necrescătoare T: N → (0, ∞), denumit program de răcire. N este mulțimea numerelor întregi pozitive, iar T(t) este temperatura la timpul t.
12) O "stare" inițială x(0) ϵ S

Având în vedere elementele de mai sus, algoritmul SA constă dintr-un lanț Markov neomogen discret în timp x(t). Dacă starea curentă x(t) este egală cu i, se alege un vecin j al lui i la întâmplare. Probabilitatea pentru oricare j ϵ S(i) să fie selectat este egală cu q_{ij}. Odată ce j este ales, următoarea stare x(t + 1) este determinată astfel:

- Dacă J(j) ≤ J(i), atunci x(t + 1) = j
- Dacă J(j) > J(i), atunci x(t + 1) = j cu probabilitatea exp[- (J(j) – J(i)) / T(t)], altfel x(t + 1) = i

În teorie, P[x(t + 1) = j|x(t) = i]

$$= q_{ij}\exp\left[-\frac{1}{T(t)}\, max\{0, J(j) - J(i)\}\right] \qquad (1)$$

dacă j ≠ i, j ϵ S(i).

Dacă j ≠ i și j ∉ S(i), atunci P[x(t + 1) = j | x(t) = i] = 0.

Rațiunea din spatele algoritmului Simulated Annealing este cel mai bine înțeleasă prin luarea în considerare a unui lanț Markov neomogen x_T(t), în care temperatura T(t) este menținută la o valoare constantă T. Să presupunem că lanțul Markov x_T(t) este ireductibil și aperiodic și că $q_{ij} = q_{ji}$ pentru fiecare i, j. Atunci x_T(t) este un lanț Markov reversibil și distribuția sa invariantă de probabilitate este dată de către:

$$\pi_T(i) = \frac{1}{Z_T}\exp\left[-\frac{J(i)}{T}\right],\ i \epsilon S \qquad (2)$$

Unde Z_T este o constantă de normalizare. Este evident că atunci când T ↓ 0, distribuția de probabilități π_T este concentrată pe mulțimea S^* a minimelor globale al lui J. Această ultimă proprietate rămâne validă daca condiția că $q_{ij} = q_{ji}$ este îndeplinită. (Faigle și Kern, 1989).

Distribuția probabilităților (2), cunoscut sub numele de distribuția Gibbs, joacă un rol foarte important în mecanica statistică. De fapt, fizicienii statisticieni au fost interesați să genereze un element eșantion de S, desenat în funcție de distribuția de probabilități π_T. Aceasta se realizează simulând lanțul Markov x_T(t) până când ajunge la echilibru, metoda fiind cunoscută sub numele de algoritmul Metropolis (1953). În contextul optimizării, se poate genera un element optim de S cu o probabilitate mare dacă se va produce un eșantion aleatoriu în funcție de distribuția π_T, cu o temperatură (T) foarte mică. O dificultate în această abordare este că atunci când T este foarte mic, timpul necesar lanțului Markov pentru a ajunge la echilibru poate fi excesiv. Algoritmul Simulated Annealing încearcă să remedieze acest dezavantaj utilizând un "program de răcire" lent T(t).

Procedeul Simulated Annealing poate fi privit ca un algoritm de căutare locală în care (spre deosebire de algoritmul de căutare locală determinist) există mișcări ocazionale "în sus" care conduc la o creștere a costului. Se dorește ca astfel de mișcări ascendente să ajute la evitarea minimelor locale.

3.2.3 Analiza convergenței

În acest subcapitol se vor prezenta performanțele algoritmului Simulated Annealing. De acum înainte, se va presupune că pentru un anumit T (și prin urmare pentru toate temperaturile), lanțul Markov x_T(t) este ireductibil și aperiodic. Se va spune că procedeul Simulated Annealing converge dacă $lim_{t \to \infty}$ P[x(t) ϵ S^*] = 1. Un volum echitabil de muncă a fost utilizat în găsirea condițiilor necesare și suficiente pentru convergență. Rezultatul principal, datorat lui Hajek, este prezentat în continuare, urmând câteva definiții.

Teorema 1 (Hajek, 1988): Spunem că starea i comunică cu S^* la o înălțime h dacă există un drum în S (fiecare element al căii fiind un vecin al elementului precedent) care începe de la i și se termină la un element al lui S^* și astfel încât cea mai mare valoare a lui J de-a lungul drumului este J(i) + h. Fie d^* cel mai mic număr astfel încât fiecare i ϵ S comunică cu S^* la înălțimea d^*. Atunci, procedeul SA converge dacă și numai dacă $lim_{t \to \infty}$ T(t) = 0 și

$$\sum_{t=1}^{\infty} \exp[-\frac{d^*}{T(t)}] = \infty. \quad (3)$$

Cel mai popular program de răcire (cel puțin în teorie) este sub forma:

$$T(t) = \frac{d}{\log t}, \quad (4)$$

unde d este o constantă pozitivă. Teorema 1 afirmă că SA converge dacă și numai dacă d ≥ d^*. Constanta d^* este o măsură a dificultății pentru x(t) pentru a evita minimele locale și a trece de la o stare neoptimală la S^*. Suntem interesați în primul rând de problemele în care d^* > 0, care vor fi asumate de acum încolo. Astfel de probleme au minimele locale care nu sunt optime. Câteva aspecte ale teoremei 1 sunt furnizate de următorul argument. Se ia în considerare un minim local al cărui "drum" este d^*. Procedeul SA realizează un număr infinit de încercări de a trece de minimul local, iar probabilitatea de succes al fiecărei încercări este de ordinul $\exp[-\frac{d^*}{T(t)}]$. Atunci condiția (3) argumentează (datorită lemei Borel-Cantelli) că un număr infinit de astfel de încercări vor fi îndeplinite. Într-adevăr, teorema lui Hajek, se realizează prin estimarea statisticilor timpilor de ieșire din anumite vecinătăți ale minimelor locale.

Fie π_T(i, t) = P[x(t) = i]. Dacă T(t) scade foarte încet, ca în cazul (4), atunci comportamentul lui x(t) pe intervale de timp destul de lungi, este de așteptat ca diferența dintre $\pi_{T(t)}(i)$ and π_T(i, t) să fie mică în orice moment. Într-adevăr, una dintre primele dovezi de convergență s-a bazat pe această idee, deși rezultatele au fost mai puțin clare decât în Teorema 1.

Pentru a obține o mai mare intuiție cu privire la interpretarea Teoremei 1, vom continua legătura dintre SA și familia corespunzătoare de lanțuri omogene Markov. Pentru acest scop, se va considera programul de răcire $T(t) = \frac{d}{\log t}$. În general, statisticile despre lanțul Markov x(t) din cadrul unui program variabil lent de răcire T(t) rămân aproape neschimbate dacă se utilizează un program de răcire aferent, în care temperatura este menținută constant pentru perioade destul de lungi de timp. În cazul nostru, programul $T(t) = \frac{d}{\log t}$ poate fi aproximat astfel. Fie t_1 = 1 și $t_{k+1} = t_k$ + exp(kd). Atunci fie $\hat{T}(t) = \frac{1}{k}$, pentru $t_k \leq t < t_{k+1}$. Se consideră segmentul $[t_k, t_{k+1}]$ al programului din bucăți constante $\hat{T}$(t). În acest caz se confruntă cu un lanț reversibil omogen $x_{\frac{1}{k}}$(t), iar în continuare vom aprofunda cât de repede poate să ajungă în starea de echilibru.

Se dorește studiul convergenței lanțului $x_{\frac{1}{k}}$(t). Valorile proprii ale matricei de probabilitate de tranziție sunt reale. Timpul de relaxare este determinată de a doua cea mai mare valoare proprie λ_2, existând estimări bune cel puțin în limita k → ∞ (exemplu: Chiang și Chow, 1988). În particular, dacă costul funcției J are global minim unic, timpul de relaxare este aproximat de exp(kd^*).

Destul de interesant este că d^* este egală cu aceeași constantă definită în Teorema 1, deși acest lucru este departe de adevăr. Aceasta oferă o altă interpretare a condiției de convergență d ≥ d^* pentru programul $\hat{T}$(t). Dacă d < d^*, atunci la fiecare temperatură $\frac{1}{k}$ se folosește $x_{\frac{1}{k}}$(t) pentru o fracție neglijabilă al timpului de relaxare, deși nu este suficientă pentru π(i, t) de a se apropria de π_T(i). Pe de altă parte, dacă d > d^*, atunci intervalul [t_k, t_{k+1}] corespunde timpilor de relaxare al lui $x_{\frac{1}{k}}$(t) cu ajutorul exp[k(d^* - d)], ce implică ca π(i, t_{k+1}) să fie foarte aproape de $\pi_{\frac{1}{k}}$(i), cu k → ∞.

Se poate urmări, de asemenea, această aproximare prin programe constante în mod direct, fără a introduce valori proprii de estimare. Ideea principală este că la o temperatura joasă, statisticile unui lanț omogen pot fi limitate cu exactitate prin vizionarea lui ca un lanț Markov perturbat și prin utilizarea limitelor de abatere destul de mari.

Convergența algoritmului Simulated Annealing (în sensul definit anterior) este o proprietate încurajatoare, dar nu este suficientă pentru a fi un algoritm potrivit. Este nevoie să se știe viteza de convergență. Se poate demonstra că pentru oricare program T(t) = $\frac{d}{\log t}$ și pentru fiecare t

$$\max_{x(0)} \mathrm{P}[x(t) \notin S^* \mid x(0) \geq \frac{A}{t^a}, \qquad (5)$$

unde A și a sunt constante pozitive ce depind de funcția J și structura vecinătății. Dacă se dorește ca x(t) să fie în afara lui S^* cu o probabilitate mai mică decât ϵ, avem nevoie ca t ≥ $\left(\frac{A}{\epsilon}\right)^{\frac{1}{a}}$.

Pentru o perspectivă mai practică, în timp ce se execută algoritmul, trebuie ținută evidența celei mai bune stări i întâlnite până acum și costul J(i) asociat. Dacă algoritmul trebuie să fie executat de t^* pași de timp, fără a se ține cont de valoarea P(x(t^*) ∉ S^*). Mai degrabă, interesul se axează pe probabilitatea că nici o stare a lui S^* să fie vizitată pe perioada de execuție a algoritmului. Având în vedere un program de răcire de forma T(t) = $\frac{d}{\log t}$, unde d > d^*, poate fi arătat că această probabilitate este cel puțin $\frac{A}{(t^*)^a}$ pentru câteva constante pozitive A și a. Ea dispare cu cât t → ∞. Pe de altă parte, dacă temperatura este fixată la oricare valoare pozitivă (sau chiar la infinit, care corespunde unei plimbări aleatorii), probabilitatea ca nici o stare din S^* să fie vizitată în t^* unități de timp, este cel mult $Be^{\frac{-t^*}{b}}$ pentru constantele pozitive corespunzătoare B și b. Deci, de foarte multe ori, performanța unei plimbări aleatorii pare să fie mai bună decât garanțiile de performanță ale procedeului SA. Punctul cheie este că analizele de mai sus, bazate pe un timp asimptotic, implică constante extrem de mari și sunt în mare măsură irelevante. Într-adevăr, constantele din estimările de mai sus sunt adesea mult mai mari decât cardinalitatea spațiului S. În special, analiza de mai sus nu poate stabili că algoritmul SA este întotdeauna cea mai bună soluție pentru o căutare exhaustivă.

3.2.4 Comportamentul în practică

În ciuda lipsei unei justificări teoretice riguroase a vitezei sale de convergență, cercetătorii au utilizat acest procedeu pe o scară largă în diferite aplicații în ultimul deceniu. Există numeroase programe de răcire ce influențează calitatea soluțiilor obținute. În van Laarhoven și Aarts (1987), autorii compară trei programe diferite de răcire pentru problema de partiționare a graficelor și au observat că particularitatea soluției găsite de diferitele programe de răcire pot să difere cu până la 10 procente. O altă observație este că timpul de execuție poate fi exagerat de mare pentru anumite probleme.

Bohachevsky, Johnson și Stein (1986) au propus o procedură SA "generalizată" pentru probleme de optimizare continuă și au aplicat metoda lor la o problemă optimă de proiectare. Mulți cercetători au considerat Simulated Annealing ca un instrument pentru dezvoltarea unor modele experimentale optime. Exemple recente sunt descrise în Currin și colaboratorii săi (1991), Meyer

și Nachtsheim (1988) și Sacks și Schiller (1988). Variante ale algoritmului bazate pe ideile lui Bayesian au fost propuse de Laud, Berliner și Goel (1989), dar și de van Laarhoven și colaboratorii săi (1989).

În ansamblu, SA este un algoritm probabilistic de aproximare general aplicabil și ușor de implementat, fiind capabil să producă soluții bune pentru o problemă de optimizare, chiar dacă nu se înțelege bine structura problemei. Cu toate acestea, se consideră că sunt necesare mai multe cercetări, atât teoretice, cât și experimentale, pentru a evalua în continuare potențialul metodei.

3.2.5 Aplicații ale procedeului Simulated Annealing

- Pentru rețele neuronale:
    1) Minimizarea funcției de energie în cazul învățării supervizate la rețelele feedforward. În acest caz sunt utilizați pentru minimizarea unor funcții cu multe minime locale și sunt definite pe un domeniu continuu.
    2) Determinarea stării de energie minimă la rețelele recurente stohastice (modelul Hopfield pentru rezolvarea problemelor de optimizare sau mașina Boltzmann utilizată în rezolvarea problemelor de asociere). În acest caz sunt utilizați pentru determinarea unei configurații de cost minim dintr-un spațiu discret.
- Alte aplicații:
    1) Prelucrarea imaginilor (eliminarea zgomotului și segmentare).
    2) Rezolvarea problemelor de rutare.
    3) Modelarea structurilor amorfe.
    4) Analiza datelor obținute prin investigații bazate pe difracție cu raze X sau rezonanța magnetică nucleară.

3.3 Mediul de dezvoltare Visual Studio

Microsoft Visual Studio este un mediu de dezvoltare integrat (IDE) de la Microsoft. Este folosit pentru a dezvolta programe de calculator, precum și site-uri web, aplicații web, servicii web și aplicații mobile. Visual Studio utilizează platforme de dezvoltare software Microsoft: Windows API, Windows Forms, Windows Presentation Foundation, Windows Store și Microsoft Silverlight. Poate produce atât cod nativ, cât și cod gestionat.

Visual Studio include un editor de cod care suportă IntelliSense (componenta de completare a codului), precum și refactorizarea codului. Debuggerul integrat funcționează atât ca un debugger la nivel de sursă, cât și ca un depanator la nivel de mașină. Alte instrumente încorporate în acest IDE sunt: designerul de formulare pentru construirea aplicațiilor GUI, designerul web, designerul de clasă și designerul de schemă de baze de date. Acesta acceptă plugin-uri care îmbunătățesc funcționalitatea la aproape toate nivelele, inclusiv adăugarea de suport pentru sistemele de control al sursei (precum Subversion și Git) și adăugarea de noi seturi de instrumente, cum ar fi editorii și designerii vizuali pentru domeniul specific al limbajelor sau alte instrumente ce ajută la procesul de dezvoltare software (cum ar fi clientul Team Foundation Server: Team Explorer).

Acest IDE suportă 36 de limbaje de programare diferite și permite editorului de cod și depanatorului să suporte (în grade diferite) aproape orice limbaj de programare, cu condiția să existe un serviciu specific limbajului. Limbajele de programare încorporate sunt: C, C++, C++/CLI, Visual Basic .NET, C#, F#, Java Script, TypeScript, XML, XSLT, HTML și CSS. Oferă suport și pentru alte limbaje precum Phyton, Ruby, Node.js. Pentru limbajul de programare Java, Microsoft nu mai oferă suport.

Ediția cea mai utilizată este ediția comunitară, fiind disponibilă în mod gratuit. Sloganul pentru versiunea Visual Studio Community este "IDE gratuit și complet, pentru studenți, dezvoltatori open-source și dezvoltatori individuali".

În acest moment, Microsoft oferă suport pentru versiunea Visual Studio 2019.

Figura 7. Logo Microsoft Visual Studio [14]

3.4 Controlul versiunilor de dezvoltare software GitHub/Git

3.4.1 GitHub

Github este o companie americană care oferă găzduire pentru controlul versiunilor de dezvoltare software folosind Git. Ea este o filială a companiei Microsoft. GitHub a fost achiziționat de către Microsoft în 2018 pentru suma de 7,5 miliarde de dolari. GitHub oferă toate funcționalitățile distribuite de control al versiunilor și managementul codului sursă (MCS) ale Git-ului, precum și adăugarea propriilor caracteristici. În plus, oferă mai multe funcții de colaborare, cum ar fi urmărirea erorilor, solicitările de caracteristici, gestionarea sarcinilor și documentația pentru fiecare proiect.

Această companie oferă conturi de utilizator gratuite, profesionale și de întreprindere. Conturile gratuite GitHub sunt utilizate în mod obișnuit pentru a găzdui proiecte open-source. Începând cu Ianuarie 2019, GitHub oferă depozite private nelimitate pentru toate planurile, inclusiv pentru conturile gratuite.

Figura 8. Logo GitHub [12]

3.4.2 Git

Git este un sistem distribuit de control al versiunii pentru urmărirea schimbărilor în codul sursă în timpul dezvoltării software-ului. Acesta este conceput pentru coordonarea muncii între programatori, dar poate fi folosit și pentru urmărirea modificărilor în orice set de fișiere. Obiectivele sale includ viteza, integritatea datelor și susținerea fluxurilor de lucru distribuite neliniar.

A fost creat de Linus Torvalds în 2005 pentru dezvoltarea nucleului Linux, împreună cu alți dezvoltatori de nuclee. Ca și în cazul celorlalte sisteme distribuite pentru controlul versiunii și spre deosebire de majoritatea sistemelor client-server, fiecare director Git de pe fiecare calculator este un depozit complet cu o istorie completă și abilități complete de urmărire a versiunilor, fără a se ține cont de accesul la rețea sau de un server central.

Figura 9. Logo Git [13]

4 Implementarea soluției adoptate

În acest capitol se prezintă soluția adoptată pentru întocmirea lucrării de diplomă. Capitolul se împarte în subcapitole pentru a se distinge mult mai bine toate aspectele abordate.

4.1 Descrierea microrețelei

Diagrama schematică a microrețelei propuse cu 230V/50Hz este prezentată în Figura 10 [10]. Caracteristicile sistemului de stocare și a celor trei surse de energie distribuită utilizate sunt următoarele:

5) Panouri fotovoltaice de 12 – 250 Wp, fabricate de către ET Solar (ET-P660250WW), invertorul SMA Sunny-Boy 3600, o unitate ce urmărește transformarea razelor de soare în energie electrică. Modulele fotovoltaice sunt conectate în serie.
6) Un generator geotermal cu o putere maximă de ieșire de 3 kW.
7) Un generator de biomasă cu o putere maximă de ieșire de 3 kW.
8) Unități de stocare: 8 – 250 Ah, 12 V baterii VRLA, invertorul Sunny Island 6.0H. o unitate bidirecțională de formare a rețelei (setează voltajul și frecvența rețelei, fiind folosită pentru încărcarea și descărcarea bateriilor).

Comunicarea dintre componentele prezentate mai sus și PC este realizată cu ajutorul protocolului RS-485.

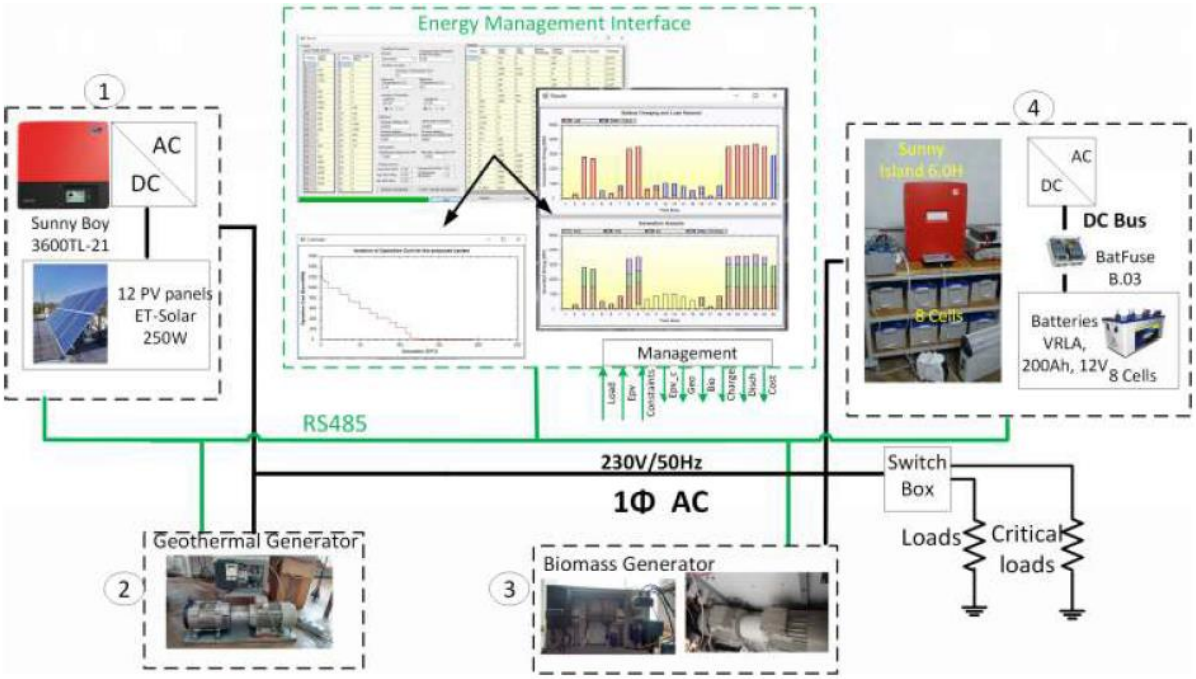


Figura 10. Microrețeaua propusă

4.2 Formularea matematică

Scopul programării zilnice este de a obține costul minim de operare al microrețelei, asigurându-se în același timp că cerințele energetice sunt îndeplinite chiar și în momentele critice.

Această problemă de optimizare poate fi reprezentată printr-o funcție obiectiv și mai multe constrângeri.

Funcția obiectiv al sistemului propus este reprezentată prin ecuația (6). Ea depinde de costul de operare al fiecărei unități conectate.

$$f_{min}=\sum_{t=0}^{23}\begin{pmatrix}E_{PV}(t)*\ C_{PV}+\ E_{GEO}(t)*\ C_{GEO}+\ E_{BIO}(t)*\ C_{BIO}+\\ E_{BAT}(t)*\ C_{BAT}+\ E_{Excess}(t)*\ C_{Excess}\end{pmatrix} \quad (6)$$

Unde E_{PV}, E_{GEO} și E_{BIO} reprezintă energia produsă de soare, geotermală și biomasă, iar E_{BAT} este energia încărcată / descărcată pentru baterii. E_{Excess} reprezintă surplusul de energie produs. Acest exces de energie nu este stocată în baterii, dar nici nu este utilizată. C_{PV}, C_{GEO}, C_{BIO}, C_{BAT}, și C_{Excess} sunt coeficienții de cost. În Tabelul 1 sunt definiți coeficienții de cost pentru fiecare energie utilizată.

Tabel 1. Valoarea coeficienților de cost

Coeficientul de cost	**€ / kWh**
C_{PV}	0.05
C_{GEO}	0.25
C_{BIO}	0.3
C_{BAT}	0.55
C_{Excess}	1.5

Mai mult decât atât, problema de optimizare prezentată este supusă la șapte constrângeri liniare. Acestea trebuie să fie validate în fiecare oră.

- Echilibru energetic al microrețelei în intervale:

$$E_{PV}(t) + E_{GEO}(t) + E_{BIO}(t) + E_{BAT}(t) + E_{Excess}(t) - E_{Load}(t) = 0 \quad (7)$$

- Constrângerile panourilor fotovoltaice, energiei geotermale și de biomasă:

$$0 \le E_{PV}(t) \le E_{PV\ limit}(t) \quad (8)$$

$$0 \le E_{GEO}(t) \le E_{GEOmax} \quad (9)$$

$$0 \le E_{BIO}(t) \le E_{BIOmax} \quad (10)$$

- Valorile minime și maxime de încărcare / descărcare:

$$-3500\ \text{Wh} \le E_{BAT}(t) \le 1000\ \text{Wh} \quad (11)$$

- Limitele sistemului E_S:

$$E_S(t) = E_S(t-1) - E_{BAT}(t) \quad (12)$$

$$0 \le E_S(t) \le E_{S\_limit} \quad (13)$$

$$E_{S\_limit} = 24000\ \text{Wh} \quad (14)$$

- Starea inițială de încărcare:

$$E_{S0} = 20 \text{ kWh} \quad (15)$$

Satisfacerea constrângerilor asigură faptul că cererea de sarcină este acoperită în fiecare etapă în timp ce se utilizează elementele microrețelei în parametrii specificați. Scopul problemei de optimizare este de a minimiza funcția obiectiv, obținând astfel costul minim de operare al microrețelei.

Pentru microrețeaua propusă și pentru funcționarea ei sunt utilizate următoarele variabile de decizie: $E_{PV}(t)$, $E_{GEO}(t)$, $E_{BIO}(t)$, $E_{BAT}(t)$ și $E_{Excess}(t)$. Energia generată de fiecare sursă de energie regenerabilă (SER) , precum și energia în exces se calculează la fiecare oră. E_{Load} reprezintă energia electrică cerută la fiecare oră de către consumator. Prin utilizarea algoritmului pentru programarea zilnică și luând în considerare intervalul orar de timp, obținem 120 de variabile de decizie. Aceasta reprezintă dimensiunea problemei.

Limitele maxime și minime ale fiecărei variabile de decizie sunt stabilite utilizând ecuațiile de constrângere ale panourilor fotovoltaice, ale generatorului geotermal și de biomasă, precum și ale bateriilor (8) – (11). Panourile fotovoltaice sunt diferite de celelalte SER-uri în ceea ce privește domeniul lor. Toate SER-urile se caracterizează printr-o limită inferioară constantă (0 Wh), dar limitele superioare ale panourilor fotovoltaice, $E_{PV\ limit}(t)$, depind de amplasarea microrețelei, precum și de condițiile meteorologice. Limitele stabilite în algoritm se bazează pe datele meteorologice statistice pentru locația dată a microrețelei. Generatoarele geotermale și de biomasă au o limită superioară constantă pentru generarea de energie. Limitele de încărcare și descărcare ale bateriilor sunt, de asemenea, constante.

Ecuația (6) oferă funcția obiectiv. Generarea de soluții inițiale, dar și noi, este supusă unor constrângeri de egalitate și de inegalitate. Acestea sunt reprezentate de ecuațiile (7), (13) și (15).

În Tabelul 2 și Tabelul 3 sunt prezentate limitele superioare ale panourilor fotovoltaice pentru fiecare lună din an, respectiv pentru fiecare oră. Valorile energiei electrice ce sunt cerute la fiecare oră de către consumator sunt definite în Tabelul 4.

Tabel 2. Limitele superioare ale panourilor fotovoltaice

Ora	Ianuarie	Februarie	Martie	Aprilie	Mai	Iunie
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	359	866	1080
7	0	0	366	1000	1503	1660
8	257	523	1033	1675	2128	2214
9	720	1141	1695	2290	2667	2685
10	1114	1668	2237	2760	3057	3022
11	1347	2001	2569	3023	3258	3195
12	1370	2074	2639	3047	3248	3186
13	1176	1876	2437	2829	3030	2996
14	811	1445	1997	2399	2628	2648
15	357	865	1390	1815	2090	2177
16	0	256	719	1157	1477	1633
17	0	0	97	513	858	1072
18	0	0	0	0	297	544
19	0	0	0	0	0	93
20	0	0	0	0	0	0

21	0	0	0	0	0	0
22	0	0	0	0	0	0
23	0	0	0	0	0	0

Tabel 3. Limitele superioare ale panourilor fotovoltaice (continuare)

Ora	Iulie	August	Septem-brie	Octombrie	Noiembrie	Decembrie
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	1035	785	422	111	0	0
7	1593	1337	924	524	181	0
8	2132	1885	1442	972	538	300
9	2595	2366	1906	1383	878	643
10	2933	2727	2256	1688	1135	917
11	3116	2928	2446	1840	1257	1059
12	3125	2949	2452	1815	1221	1038
13	2961	2789	2273	1617	1033	859
14	2640	2465	1934	1279	733	562
15	2198	2016	1481	857	380	217
16	1679	1491	974	418	44	0
17	1136	946	478	29	0	0
18	618	438	48	0	0	0
19	169	10	0	0	0	0
20	0	0	0	0	0	0
21	0	0	0	0	0	0
22	0	0	0	0	0	0
23	0	0	0	0	0	0

Tabel 4. Valorile energiei electrice ce sunt cerute la fiecare oră de către consumator

Ora	E_{Load}
0	0
1	250
2	2800
3	2700
4	0
5	300
6	600
7	3400
8	3500
9	600
10	600
11	100

12	200
13	0
14	0
15	300
16	0
17	200
18	3500
19	3600
20	3600
21	3700
22	3500
23	0

4.3 Organigrama algoritmului Simulated Annealing

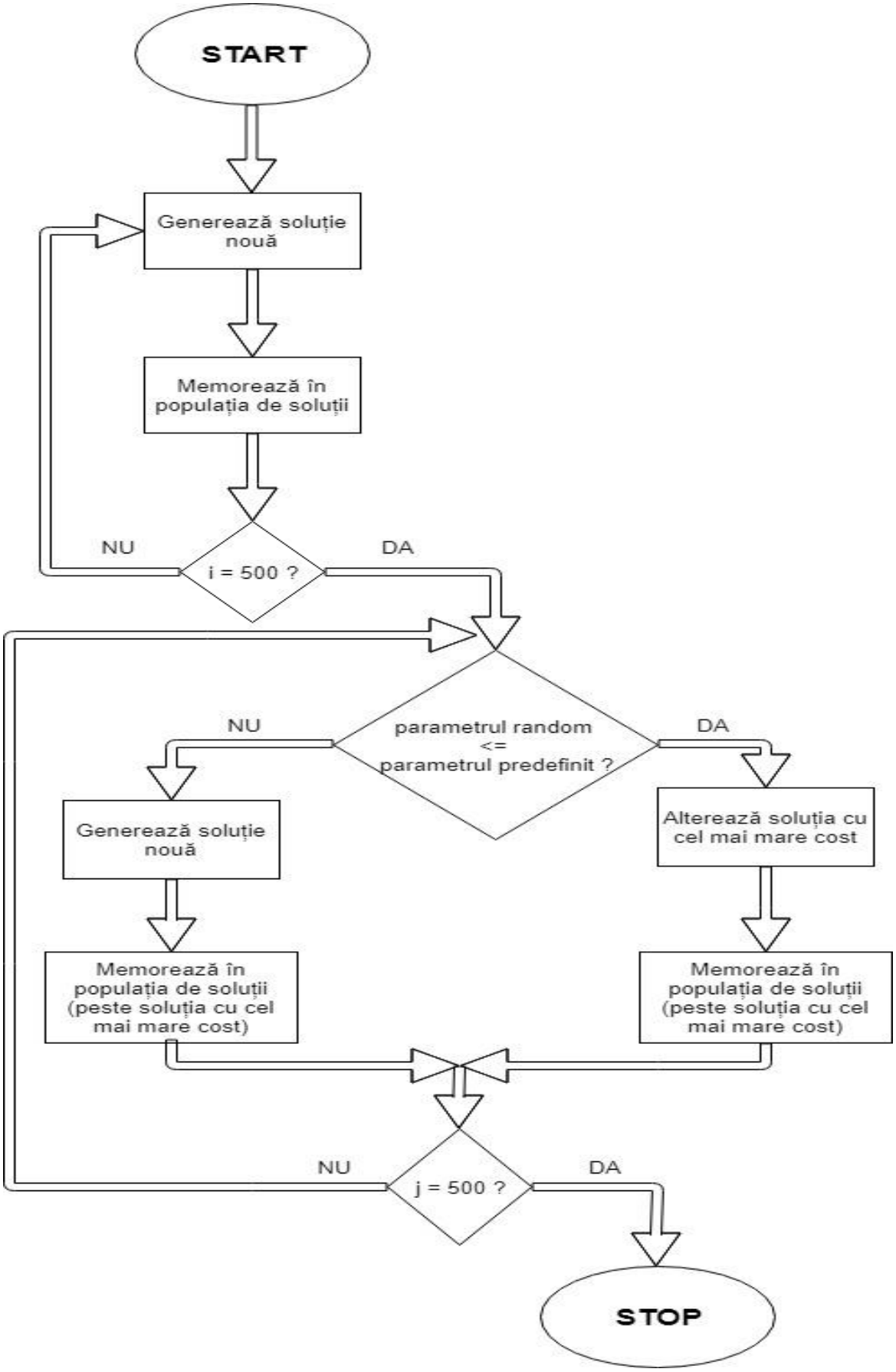


Figura 11. Organigrama algoritmului Simulated Annealing

4.4 Implementarea în limbajul de programare C

Algoritmul Simulated Annealing, procedeul folosit pentru realizarea proiectului de diplomă, a fost dezvoltat, implementat și testat în limbajul de programare C, cu ajutorul IDE-ului Microsoft Visual Studio.

În Figura 12 s-a definit în aplicație energiile și costurile necesare pentru rezolvarea problemei planificării zilnice. Cu ajutorul acestor variabile constante definite la începutul programului s-a evitat introducerea unor erori în implementarea funcțiilor ce vor urma descrise in acest subcapitol. Aceste valori sunt preluate din Tabelul 1 și din ecuațiile (8), (9), (10), (11) și (14).

```
8     //defineuri ptr min/max E
9     #define EPv_Min 0
10    #define EGeo_Min 0
11    #define EBio_Min 0
12    #define EBat_Min -3500 //Energy_Bat_Max = Ebat incarcare
13    #define Es_Min 0
14    #define EGeo_Max 1500
15    #define EBio_Max 1500
16    #define EBat_Max 1000
17    #define ES_Limit 24000 //limita sistemului de energie
18
19    //defineuri ptr costuri
20    #define CPv 0.05
21    #define CGeo 0.25
22    #define CBio 0.3
23    #define CBat 0.55
24    #define CExcess 1.5
```

Figura 12. Definirea în program a energiilor și costurilor

În Figura 13 s-a definit în program limitele superioare ale panourilor fotovoltaice pentru fiecare lună din an. Acești vectori sunt folosiți de funcția “randEPv” din program pentru a genera o valoare aleatorie respectând constrângerea (8) și luna specifică, dar și la alterarea soluției din populația inițială. Valorile sunt preluate din Tabelul 2 și Tabelul 3.

```
//luna decembrie
const unsigned int EPv_Max_Decembrie[24] = { 0, 0, 0, 0, 0, 0, 0, 0, 300, 643, 917, 1059, 1038, 859, 562,
217, 0, 0, 0, 0, 0, 0, 0, 0 };//EPV MAX ptr cele 24 ore
//luna ianuarie
const unsigned int EPv_Max_Ianuarie[24] = { 0, 0, 0, 0, 0, 0, 0, 0, 257, 720, 1114, 1347, 1370,
1176, 811, 357, 0 ,0 ,0 ,0 ,0 ,0 ,0 };//EPV MAX ptr cele 24 ore
//luna februarie
const unsigned int EPv_Max_Februarie[24] = { 0, 0, 0, 0, 0, 0, 0, 0, 523, 1141, 1668, 2001, 2074,
1876, 1445, 865, 256, 0, 0, 0, 0, 0, 0, 0 };
//luna martie
const unsigned int EPv_Max_Martie[24] = { 0, 0, 0, 0, 0, 0, 0, 366, 1033, 1695, 2237, 2569, 2639,
2437, 1997, 1390, 719, 97, 0, 0, 0, 0, 0, 0 };
//luna aprilie
const unsigned int EPv_Max_Aprilie[24] = { 0, 0, 0, 0, 0, 0, 359, 1000, 1675, 2290, 2760, 3023,
3047, 2829, 2399, 1815, 1157, 513, 0, 0, 0, 0, 0, 0 };
//luna mai
const unsigned int EPv_Max_Mai[24] = { 0, 0, 0, 0, 0, 0, 866, 1503, 2128, 2667, 3057, 3258, 3248,
3030, 2628, 2090, 1477, 858, 297, 0, 0, 0, 0, 0 };
//luna iunie
const unsigned int EPv_Max_Iunie[24] = { 0, 0, 0, 0, 0, 0, 1080, 1660, 2214, 2685, 3022, 3195,
3186, 2996, 2648, 2177, 1633, 1072, 544, 93, 0, 0, 0, 0 };
//luna iulie
const unsigned int EPv_Max_Iulie[24] = { 0, 0, 0, 0, 0, 0, 1035, 1593, 2132, 2595, 2933, 3116,
3125, 2961, 2640, 2198, 1679, 1136, 618, 169, 0, 0, 0, 0 };
//luna august
const unsigned int EPv_Max_August[24] = { 0, 0, 0, 0, 0, 0, 785, 1337, 1885, 2366, 2727, 2928,
2949, 2789, 2465, 2016, 1491, 946, 438, 10, 0, 0, 0, 0 };
//luna septembrie
const unsigned int EPv_Max_Septembrie[24] = { 0, 0, 0, 0, 0, 0, 422, 924, 1442, 1906, 2256, 2446, 2452, 2273,
1934, 1481, 974, 478, 48, 0, 0, 0, 0, 0 };
//luna octombrie
const unsigned int EPv_Max_Octombrie[24] = { 0, 0, 0, 0, 0, 0, 111, 524, 972, 1383, 1688, 1840, 1815, 1617,
1279, 857, 418, 29, 0, 0, 0, 0, 0, 0 };
//luna noiembrie
const unsigned int EPv_Max_Noiembrie[24] = { 0, 0, 0, 0, 0, 0, 0, 181, 538, 878, 1135, 1257, 1221, 1033,
733, 380, 44, 0, 0, 0, 0, 0, 0, 0 };
```

Figura 13. Limitele superioare ale panourilor fotovoltaice

În Figura 14 s-a definit energia electrică cerută la fiecare oră de către consumator. Aceste valori sunt preluate din Tabelul 4.

```
const unsigned int ELoad[24] = { 0, 250, 2800, 2700, 0, 300, 600, 3400, 3500, 600, 600, 100, 200, 0,
0, 300, 0, 200, 3500, 3600, 3600, 3700, 3500, 0 };//Eload ptr cele 24ore
```

Figura 14. Energia electrică cerută la fiecare oră de către consumator

```
unsigned int randEGeo()
{
    int random;
    random = rand() % ((EGeo_Max + 1 - EGeo_Min) + EGeo_Min);
    return random;
}

unsigned int randEBio()
{
    int random;
    random = rand() % ((EBio_Max + 1 - EBio_Min) + EBio_Min);
    return random;
}
```

Figura 15. Funcțiile ce realizează o valoare aleatorie, respectând constrângerile pentru energia geotermală și de biomasă

În Figura 16 s-a realizat o funcție ce produce o valoare aleatorie pentru energia fotovoltaică, respectând constrângerea (8) și limitele superioare ale panourilor fotovoltaice în funcție de lună.

Limitele superioare ale panourilor fotovoltaice sunt preluate din vectorii definiți în Figura 13.

```
98   unsigned int randEPv(unsigned int hour, unsigned int month)
99   {
100      int random;
101      if (month == 1)
102      {
103          random = rand() % ((EPv_Max_Ianuarie[hour] + 1 - EPv_Min) + EPv_Min);
104      }
105
106      else if (month == 2) { ... }
110
111      else if (month == 3) { ... }
115
116      else if (month == 4) { ... }
120
121      else if (month == 5) { ... }
125
126      else if (month == 6) { ... }
130
131      else if (month == 7) { ... }
135
136      else if (month == 8) { ... }
140
141      else if (month == 9) { ... }
145
146      else if (month == 10) { ... }
150
151      else if (month == 11) { ... }
155
156      else if (month == 12) { ... }
160
161      return random;
162  }
```

Figura 16. Funcție ce realizează o valoare aleatorie, respectând constrângerea pentru energia fotovoltaică

În Figura 17 s-a realizat o funcție ce produce o valoare aleatorie în intervalul [0,1], utilizată de către algoritmul Simulated Annealing pentru rezolvarea problemei planificării zilnice. Mai exact, în funcție de această valoare aleatorie programul alege ce caz să urmeze în continuare pentru a respecta principiul algoritmului.

```
164  double randParameter()
165  {
166      double random;
167      random = (double)rand() / (double)((unsigned)RAND_MAX + 1);
168      return random;
169  }
```

Figura 17. Funcție ce produce o valoare aleatorie utilizată în procedeul Simulated Annealing

În Figura 18 s-a implementat ecuația (6). Ecuație ce realizează costul total al unei soluții.

```
187  double calculateTotalCosts(unsigned int EPv[], unsigned int EGeo[], unsigned int EBio[], int EBat[], unsigned int EExcess[])
188  {
189      double sum = 0.0;
190      for (unsigned int hour = 0; hour < 24; hour++)
191      {
192          sum = sum + ((EPv[hour] * CPv) + (EGeo[hour] * CGeo) + (EBio[hour] * CBio) + (EBat[hour] * CBat) +
193              (EExcess[hour] * CExcess));
194      }
195      return sum;
196  }
```

Figura 18. Implementarea ecuației (6)

În Figura 19 s-a realizat o funcție ce generează o soluție nouă respectând ecuațiile (7), (11), (12), (13) și (14). Funcția creează o soluție pe baza funcțiilor definite în Figura 15 și în Figura 16.

Aceste soluții sunt memorate într-o structură de date pentru a putea fi prelucrate mai târziu în program de către procedeul Simulated Annealing. Structura de date definită se poate observa în Figura 20.

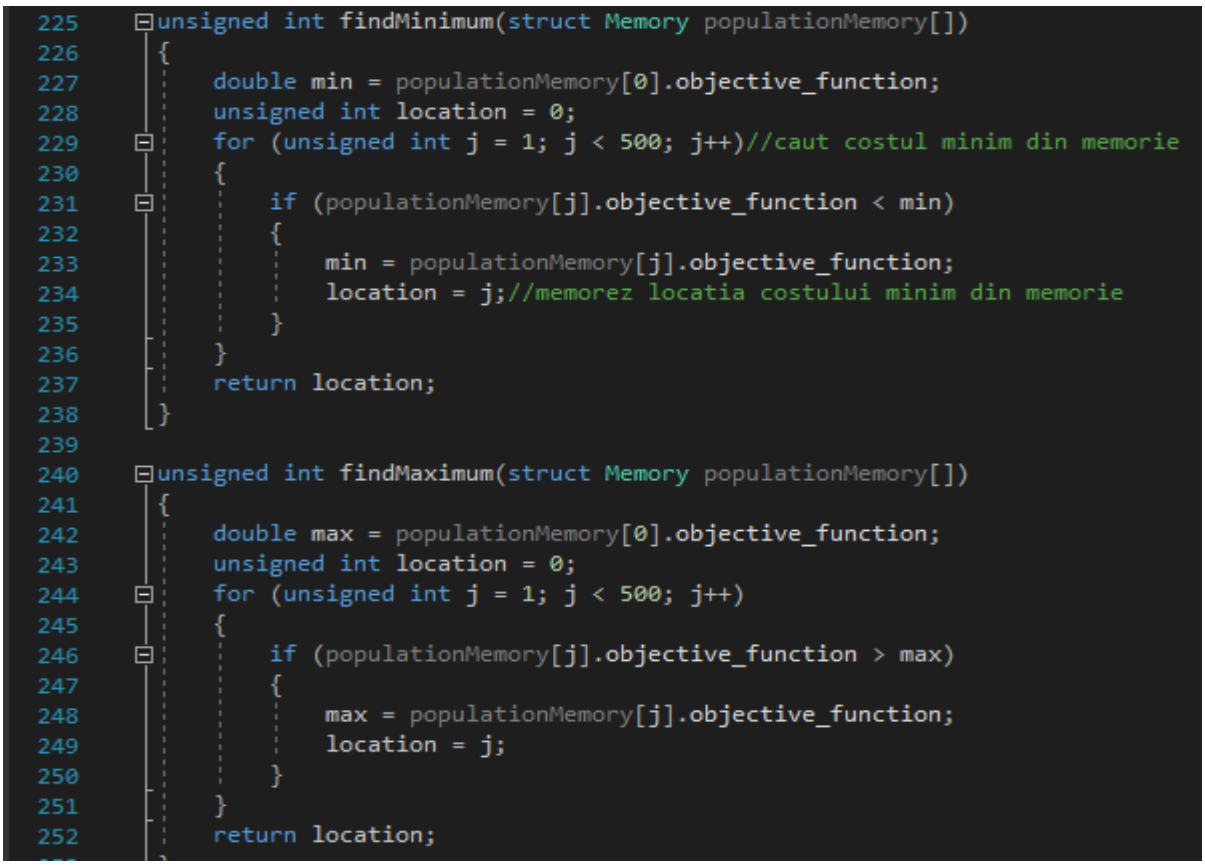
```
198  void generateSolution(unsigned int month)
199  {
200      Es[0] = 20000;
201      for (unsigned int hour = 0; hour < 24; hour++)
202      {
203          EPv[hour] = randEPv(hour, month);
204          EGeo[hour] = randEGeo();
205          EBio[hour] = randEBio();
206
207          EBat[hour] = ELoad[hour] - (EPv[hour] + EGeo[hour] + EBio[hour] + EExcess[hour]);
208
209          EExcess[hour] = 0;
210          if (EBat[hour] < EBat_Min)
211          {
212              EExcess[hour] = (EBat[hour] - EBat_Min) * (-1);//ptr a avea valoare pozitiva
213              EBat[hour] = EBat_Min;
214          }
215          if (EBat[hour] > EBat_Max)
216          {
217              EExcess[hour] = EBat[hour] - EBat_Max;
218              EBat[hour] = EBat_Max;
219          }
220
221          Es[hour] = Es[hour - 1] - EBat[hour];
222      }
223  }
```

Figura 19. Funcție ce realizează o nouă soluție

```
27  struct Memory
28  {
29      double objective_function;
30      unsigned int EPv[24];
31      unsigned int EGeo[24];
32      unsigned int EBio[24];
33      unsigned int ELoad[24];
34      unsigned int EExcess[24];
35      int EBat[24];
36      int Es[24];
37  };
```

Figura 20. Structura de date definită în program

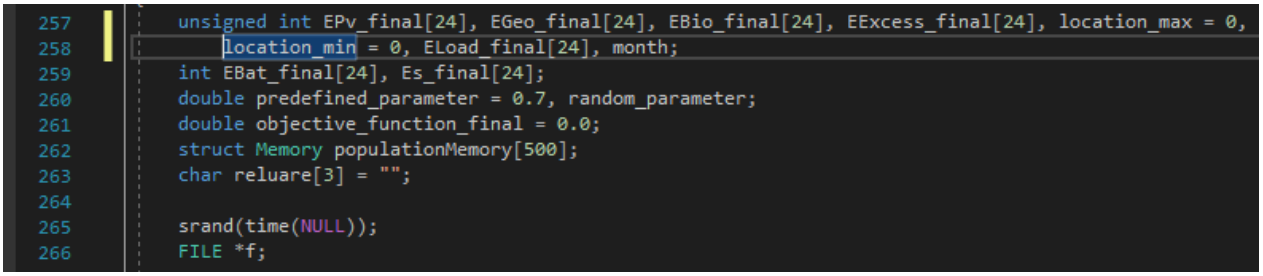
În Figura 21 s-au implementat 2 funcții ce caută soluția cu cost minim sau maxim din populația de soluții. Funcția “findMaximum” se utilizează în această aplicație pentru a găsi soluția cu cel mai mare cost din memorie, pentru ca mai apoi această soluție să fie alterată pentru a se găsi o valoare mult mai eficientă. Funcția “findMinimum” se folosește pentru a căuta soluția cea mai bună (cel mai mic cost) din populația de soluții.



```
225 unsigned int findMinimum(struct Memory populationMemory[])
226 {
227     double min = populationMemory[0].objective_function;
228     unsigned int location = 0;
229     for (unsigned int j = 1; j < 500; j++)//caut costul minim din memorie
230     {
231         if (populationMemory[j].objective_function < min)
232         {
233             min = populationMemory[j].objective_function;
234             location = j;//memorez locatia costului minim din memorie
235         }
236     }
237     return location;
238 }
239
240 unsigned int findMaximum(struct Memory populationMemory[])
241 {
242     double max = populationMemory[0].objective_function;
243     unsigned int location = 0;
244     for (unsigned int j = 1; j < 500; j++)
245     {
246         if (populationMemory[j].objective_function > max)
247         {
248             max = populationMemory[j].objective_function;
249             location = j;
250         }
251     }
252     return location;
```

Figura 21. Funcții ce caută soluția cu cost minim/maxim din memorie

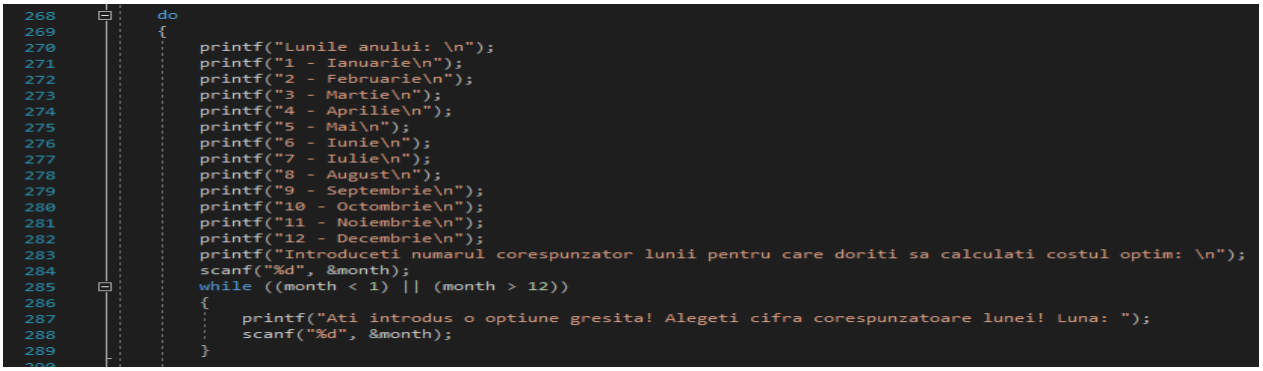
În Figura 22 sunt definite variabilele locale și inițializarea funcției predefinite rand() cu ajutorul orei locale. Funcția rand() se definește în acest mod pentru a avea întotdeauna o valoare aleatorie ce nu s-a mai repetat în execuțiile precedente.



```
257     unsigned int EPv_final[24], EGeo_final[24], EBio_final[24], EExcess_final[24], location_max = 0,
258         location_min = 0, ELoad_final[24], month;
259     int EBat_final[24], Es_final[24];
260     double predefined_parameter = 0.7, random_parameter;
261     double objective_function_final = 0.0;
262     struct Memory populationMemory[500];
263     char reluare[3] = "";
264
265     srand(time(NULL));
266     FILE *f;
```

Figura 22. Variabile locale

În Figura 23 s-a realizat interacțiunea cu utilizatorul.



```
268     do
269     {
270         printf("Lunile anului: \n");
271         printf("1 - Ianuarie\n");
272         printf("2 - Februarie\n");
273         printf("3 - Martie\n");
274         printf("4 - Aprilie\n");
275         printf("5 - Mai\n");
276         printf("6 - Iunie\n");
277         printf("7 - Iulie\n");
278         printf("8 - August\n");
279         printf("9 - Septembrie\n");
280         printf("10 - Octombrie\n");
281         printf("11 - Noiembrie\n");
282         printf("12 - Decembrie\n");
283         printf("Introduceti numarul corespunzator lunii pentru care doriti sa calculati costul optim: \n");
284         scanf("%d", &month);
285         while ((month < 1) || (month > 12))
286         {
287             printf("Ati introdus o optiune gresita! Alegeti cifra corespunzatoare lunei! Luna: ");
288             scanf("%d", &month);
289         }
```

Figura 23. Interacțiunea cu utilizatorul

În Figura 24 s-a realizat popularea memoriei cu 500 de soluții valide. Pe baza acestor soluții inițiale se efectuează algoritmul Simulated Annealing.

```
292         printf("Incep sa populez memoria cu 500 de solutii valide.\n");
293         printf("Am ajuns pe la iteratia: ");
294         for (unsigned int i = 0; i < 500; i++)//se populeaza cu 500 de solutii
295         {
296             printf("%d\n", i);
297             generateSolution(month);
298             //se copiaza in memorie 500 de solutii posibile(populatia initiala)
299             objective_function = calculateTotalCosts(EPv, EGeo, EBio, EBat, EExcess);
300             populationMemory[i].objective_function = objective_function;
301             memcpy(populationMemory[i].EPv, EPv, sizeof(EPv));
302             memcpy(populationMemory[i].EGeo, EGeo, sizeof(EGeo));
303             memcpy(populationMemory[i].EBio, EBio, sizeof(EBio));
304             memcpy(populationMemory[i].ELoad, ELoad, sizeof(ELoad));
305             memcpy(populationMemory[i].EBat, EBat, sizeof(EBat));
306             memcpy(populationMemory[i].Es, Es, sizeof(Es));
307             memcpy(populationMemory[i].EExcess, EExcess, sizeof(EExcess));
308         }
```

Figura 24. Popularea memoriei cu 500 de soluții valide

În Figura 25, Figura 26, Figura 27 și Figura 28 s-a implementat primul caz folosit de către procedeul Simulated Annealing pentru rezolvarea problemei planificării zilnice. Pentru a se executa primul caz, parametrul "random_parameter" din aplicație (fiind un parametru ce primește o valoare aleatorie din intervalul [0,1]) trebuie să fie mai mic sau cel mult egală cu parametrul predefinit (în aplicație parametrul predefinit este egală cu valoarea 0.7).

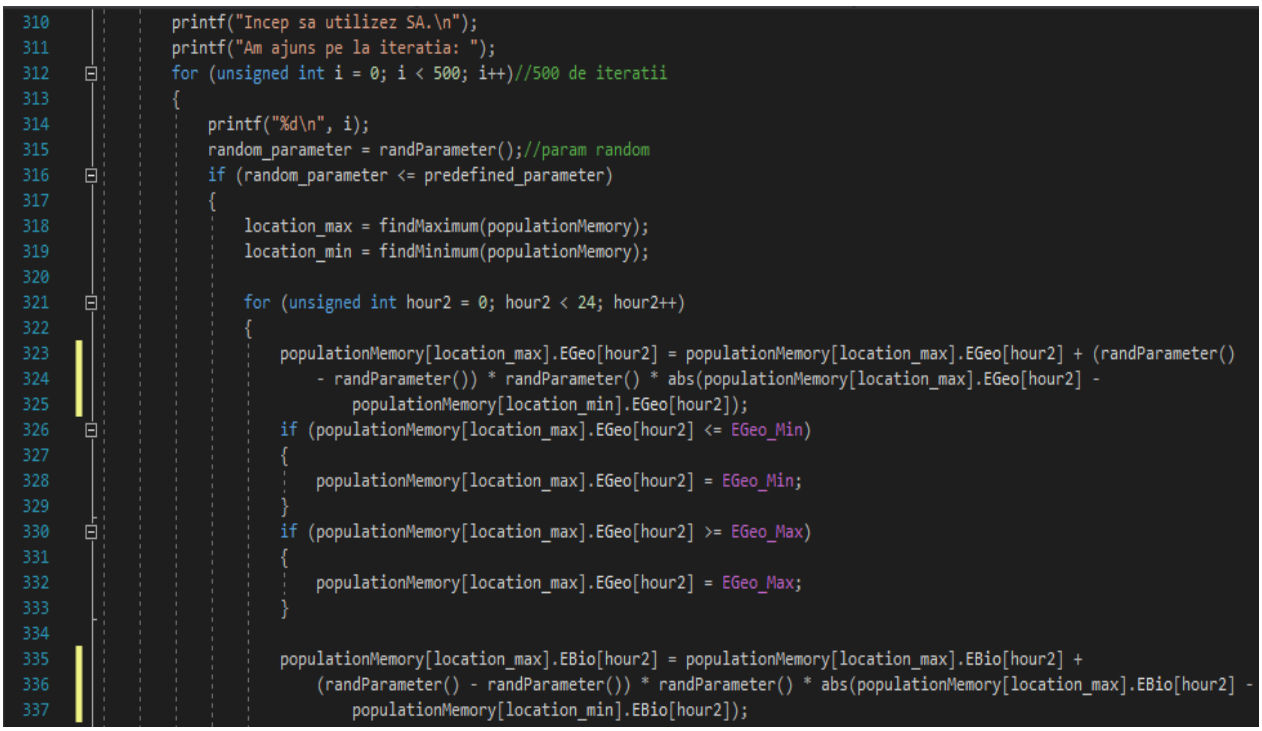
Primul caz presupune alterarea celei mai costisitoare soluții din populația inițială, pentru a forma o nouă soluție în memorie. Alterarea soluției se realizează cu ajutorul următoarei ecuații:

$$sol_i = sol_i + (random_1 - random_2) * random_3 * \text{abs}(\text{max} - \text{min}) \qquad (16)$$

Unde:

- Random1, random2 și random3 sunt valori aleatorii din intervalul [0,1]
- sol_i reprezintă soluția ce este alterată
- Max reprezintă soluția cu cel mai mare cost din memorie
- Min reprezintă soluția u cel mai mic cost din memorie

Variabilele de decizie EGeo (energia generată de către generatorul geotermal), EBio (energia generată de către generatorul de biomasă) și EPv (energia generată de către panourile fotovoltaice) sunt alterate folosind ecuația (16). Desigur că și celelalte variabile ce depind de EGeo, EBio și Epv sunt alterate și ele în urma acestui proces.

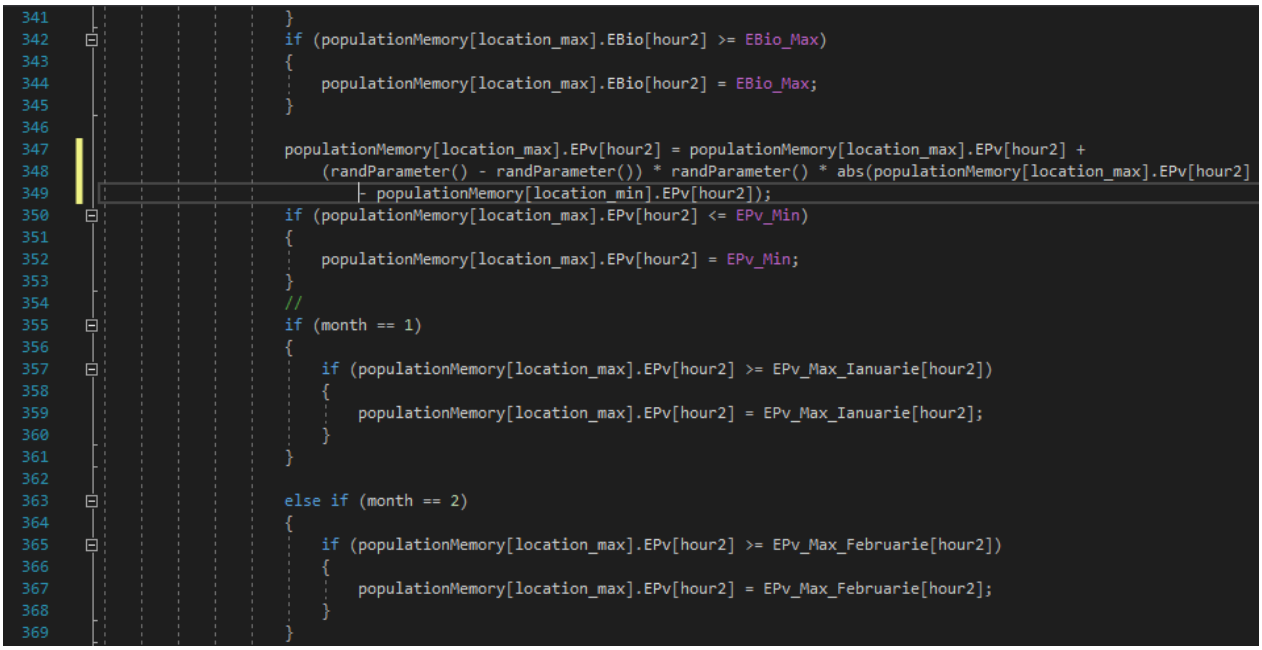


```
printf("Incep sa utilizez SA.\n");
printf("Am ajuns pe la iteratia: ");
for (unsigned int i = 0; i < 500; i++)//500 de iteratii
{
    printf("%d\n", i);
    random_parameter = randParameter();//param random
    if (random_parameter <= predefined_parameter)
    {
        location_max = findMaximum(populationMemory);
        location_min = findMinimum(populationMemory);

        for (unsigned int hour2 = 0; hour2 < 24; hour2++)
        {
            populationMemory[location_max].EGeo[hour2] = populationMemory[location_max].EGeo[hour2] + (randParameter()
                - randParameter()) * randParameter() * abs(populationMemory[location_max].EGeo[hour2] -
                    populationMemory[location_min].EGeo[hour2]);
            if (populationMemory[location_max].EGeo[hour2] <= EGeo_Min)
            {
                populationMemory[location_max].EGeo[hour2] = EGeo_Min;
            }
            if (populationMemory[location_max].EGeo[hour2] >= EGeo_Max)
            {
                populationMemory[location_max].EGeo[hour2] = EGeo_Max;
            }

            populationMemory[location_max].EBio[hour2] = populationMemory[location_max].EBio[hour2] +
                (randParameter() - randParameter()) * randParameter() * abs(populationMemory[location_max].EBio[hour2] -
                    populationMemory[location_min].EBio[hour2]);
```

Figura 25. Primul caz al algoritmului Simulated Annealing (I)



```
            }
            if (populationMemory[location_max].EBio[hour2] >= EBio_Max)
            {
                populationMemory[location_max].EBio[hour2] = EBio_Max;
            }

            populationMemory[location_max].EPv[hour2] = populationMemory[location_max].EPv[hour2] +
                (randParameter() - randParameter()) * randParameter() * abs(populationMemory[location_max].EPv[hour2]
                    - populationMemory[location_min].EPv[hour2]);
            if (populationMemory[location_max].EPv[hour2] <= EPv_Min)
            {
                populationMemory[location_max].EPv[hour2] = EPv_Min;
            }
            //
            if (month == 1)
            {
                if (populationMemory[location_max].EPv[hour2] >= EPv_Max_Ianuarie[hour2])
                {
                    populationMemory[location_max].EPv[hour2] = EPv_Max_Ianuarie[hour2];
                }
            }

            else if (month == 2)
            {
                if (populationMemory[location_max].EPv[hour2] >= EPv_Max_Februarie[hour2])
                {
                    populationMemory[location_max].EPv[hour2] = EPv_Max_Februarie[hour2];
                }
            }
```

Figura 26. Primul caz al algoritmului Simulated Annealing (II)

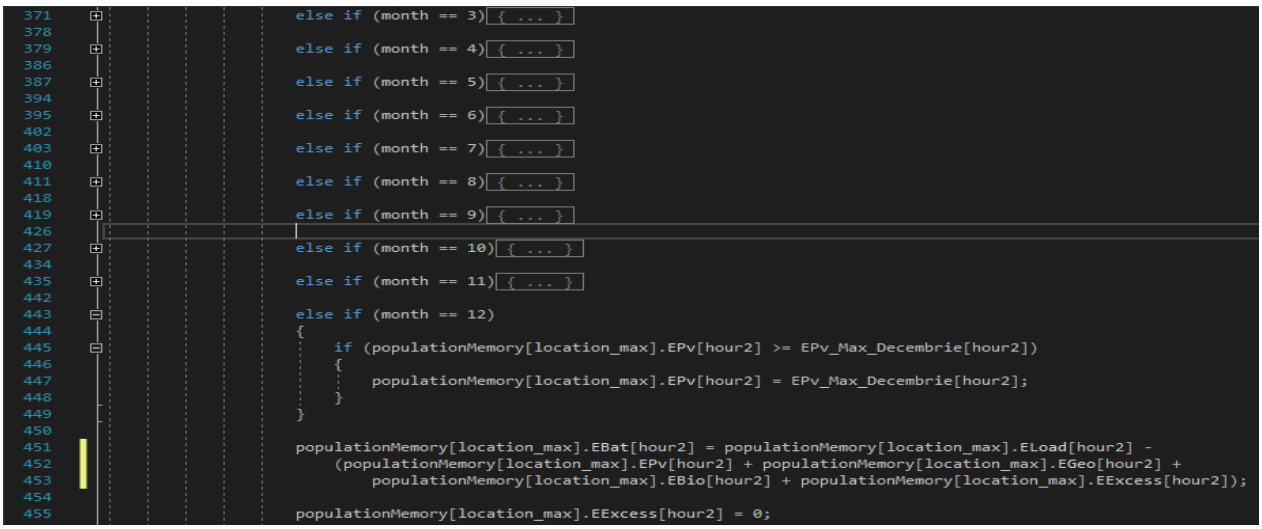


Figura 27. Primul caz al algoritmului Simulated Annealing (III)

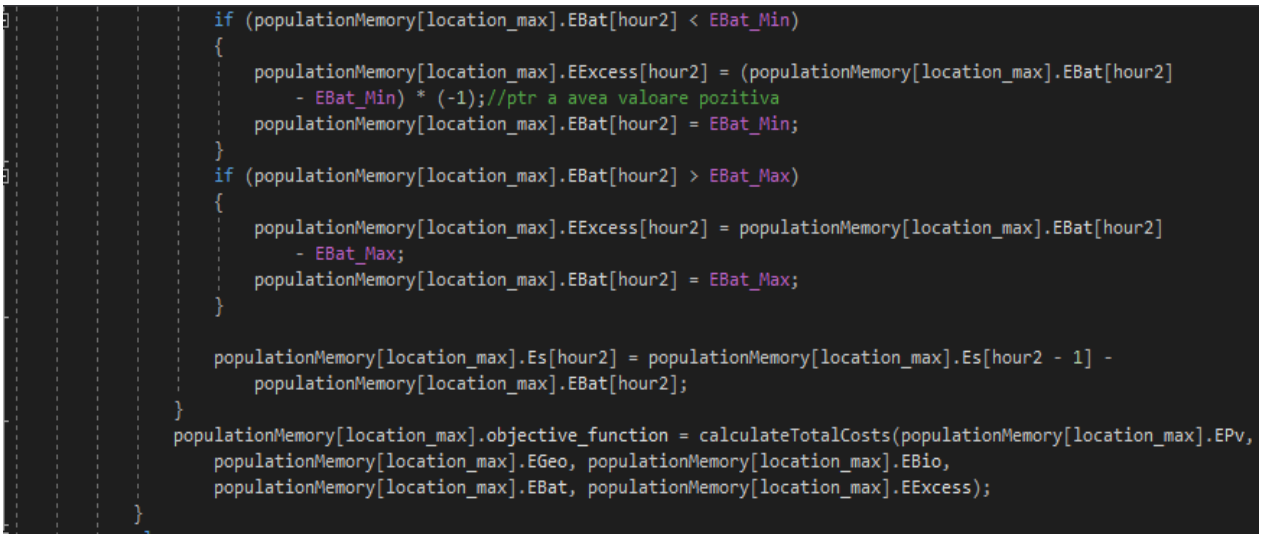


Figura 28. Primul caz al algoritmului Simulated Annealing (IV)

În Figura 29 s-a implementat al doilea caz folosit de către procedeul Simulated Annealing. Pentru a se executa al doilea caz, parametrul "random_parameter" din program trebuie să fie strict mai mare de valoarea 0.7, această valoare fiind parametrul predefinit ("predefined_parameter" din aplicație).

Al doilea caz constă în suprascrierea celei mai costisitoare soluții din memorie cu o nouă soluție generată conform funcției definite în Figura 18.

```
461         {
462             location_max = findMaximum(populationMemory);
463             generateSolution(month);
464             objective_function = calculateTotalCosts(EPv, EGeo, EBio, EBat, EExcess);
465             if (objective_function < populationMemory[location_max].objective_function)//s
466             {
467                 populationMemory[location_max].objective_function = objective_function;
468                 memcpy(populationMemory[location_max].EPv, EPv, sizeof(EPv));
469                 memcpy(populationMemory[location_max].EGeo, EGeo, sizeof(EGeo));
470                 memcpy(populationMemory[location_max].EBio, EBio, sizeof(EBio));
471                 memcpy(populationMemory[location_max].ELoad, ELoad, sizeof(ELoad));
472                 memcpy(populationMemory[location_max].EBat, EBat, sizeof(EBat));
473                 memcpy(populationMemory[location_max].Es, Es, sizeof(Es));
474                 memcpy(populationMemory[location_max].EExcess, EExcess, sizeof(EExcess));
475             }
476         }
477     }
```

Figura 29. Al doilea caz al procedeului Simulated Annealing

În Figura 30 s-a implementat afișarea rezultatului final la consolă, dar și memorarea ei în fișierul “rezultat.txt”.

```
491         f = fopen("rezultat.txt", "w");
492         if (f == NULL)
493         {
494             printf("Eroare la deschierea fisierului rezultat.txt!");
495             exit(1);
496         }
497
498         //Se afiseaza rezultatul final
499         printf("Rezultatul final pentru luna %d: \n", month);
500         fprintf(f, "Rezultatul final pentru luna %d: \n", month);
501         printf("Cost: %f\n", objective_function_final);
502         fprintf(f, "Cost: %f\n", objective_function_final);
503         printf("Hour:|\tEPv:|\tEGeo:|\tEBio:|\tEBat:|\tEExcess:|\tEs:|\tELoad:|\n");
504         fprintf(f, "Hour:|\tEPv:|\tEGeo:|\tEBio:|\tEBat:|\tEExcess:|\tEs:|\tELoad:|\n");
505         for (unsigned int i = 0; i < 24; i++)
506         {
507             printf("%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t", i + 1, EPv_final[i], EGeo_final[i], EBio_final[i],
508                 EBat_final[i], EExcess_final[i], Es_final[i], ELoad_final[i]);
509             printf("\n");
510             fprintf(f, "%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t%d|\t", i + 1, EPv_final[i], EGeo_final[i], EBio_final[i],
511                 EBat_final[i], EExcess_final[i], Es_final[i], ELoad_final[i]);
512             fprintf(f, "\n");
513         }
514
515         fclose(f);
```

Figura 30. Afișarea rezultatului la consolă și memorarea în fișier

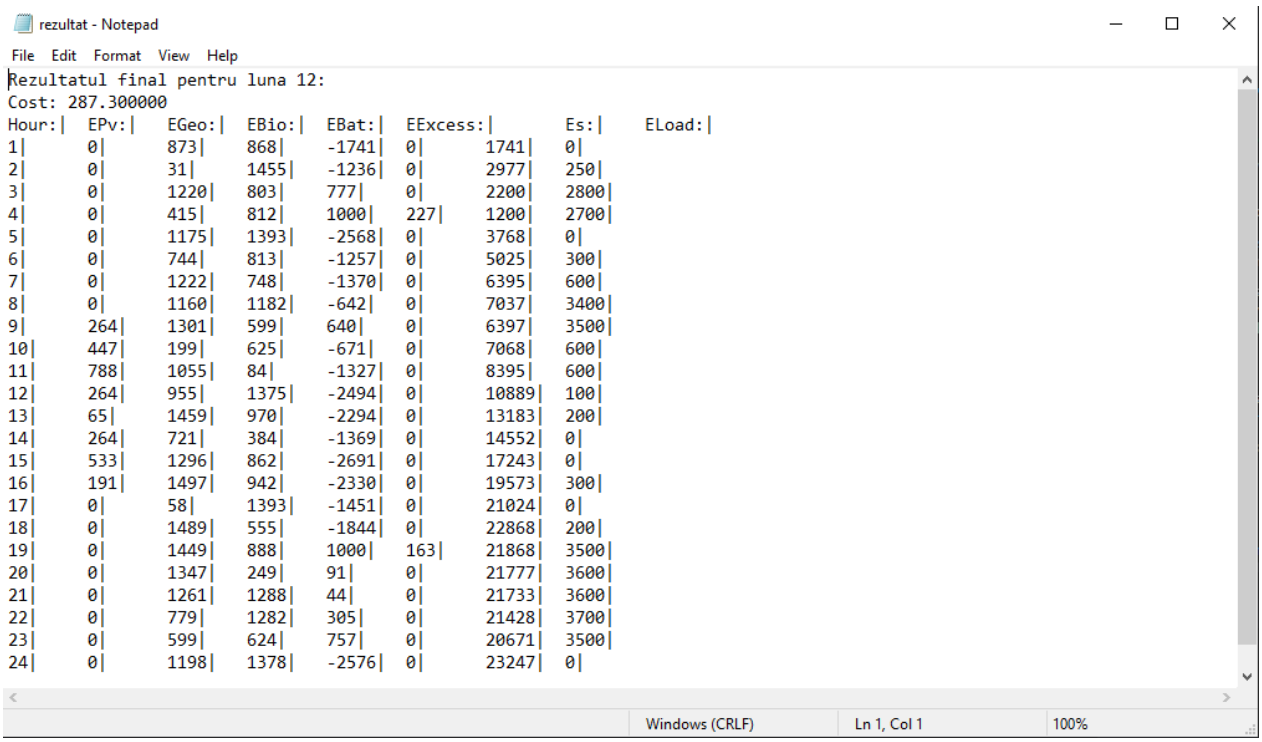


Figura 31. Exemplu de memorare a rezultatului în fișierul “rezultat.txt”

În Figura 32 am implementat un nou caz de interacțiune cu utilizatorul. Acesta este întrebat de către aplicație dacă dorește să realizeze o nouă operațiune de calculare a costului minim pentru o altă lună a anului.

```
514         printf("\n Doriti sa reluati operatiunea pentru alta luna? (Introduceti: da/nu): ");
515         scanf("%s", reluare);
516         while ((strcmp(reluare, "da") != 0) && (strcmp(reluare, "nu") != 0))
517         {
518             printf("Ati introdus o optiune gresita! Doriti sa reluati operatiunea pentru alta luna? (Introduceti: da/nu): ");
519             scanf("%s", reluare);
520         }
521     } while (strcmp(reluare, "da") == 0);
522     return (0);
523 }
```

Figura 32. Interacțiunea cu utilizatorul (II)

5 Rezultate experimentale

În urma procesului de dezvoltare și testare a algoritmului Simulated Annealing, cu ajutorul IDE-ului Microsoft Visual Studio, s-au obținut rezultatele prezentate mai jos.

5.1 Meniul pentru alegerea lunii

În Figura 33 este prezentat un meniu cu 12 opțiuni disponibile, reprezentând cele 12 luni ale anului. Utilizatorul poate selecta din lista respectivă luna pentru care algoritmul Simulated Annealing să realizeze costul optim de operare al microrețelei. Acest meniu a fost implementat pentru a avea o interacțiune mult mai prietenoasă cu utilizatorul.

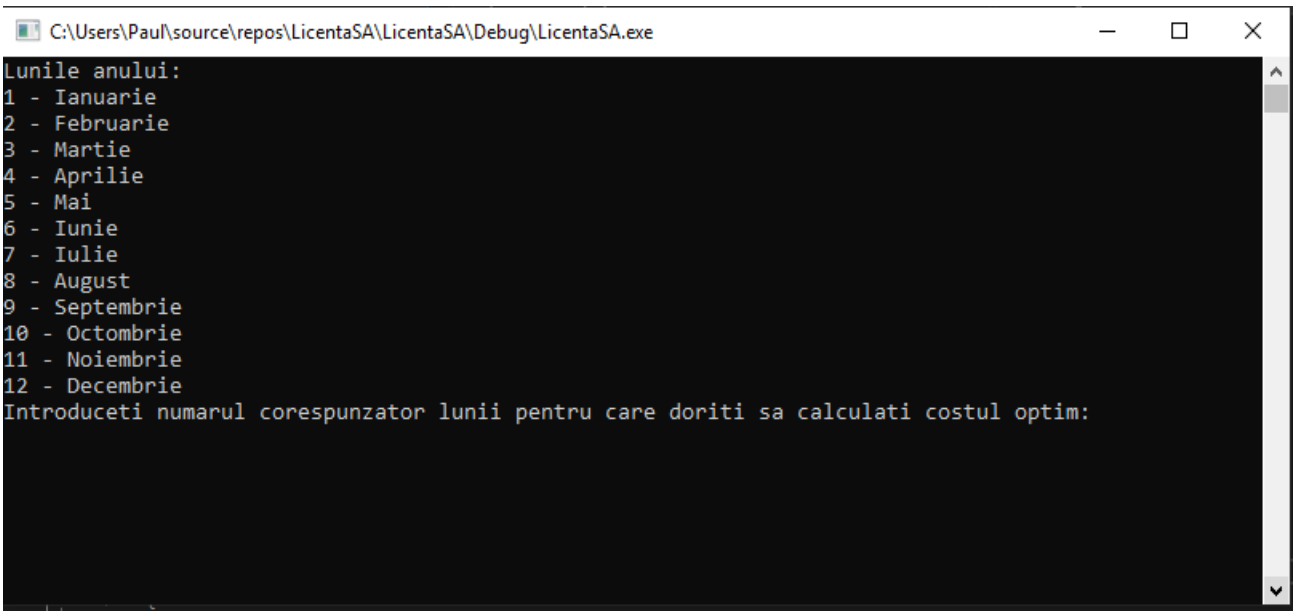


Figura 33. Meniul pentru alegerea lunii

În cazul în care utilizatorul introduce o altă opțiune care nu este specificată în lista meniului, aplicația o să îl atenționeze că a introdus o opțiune greșită și așteaptă ca acesta să introducă o opțiune validă.

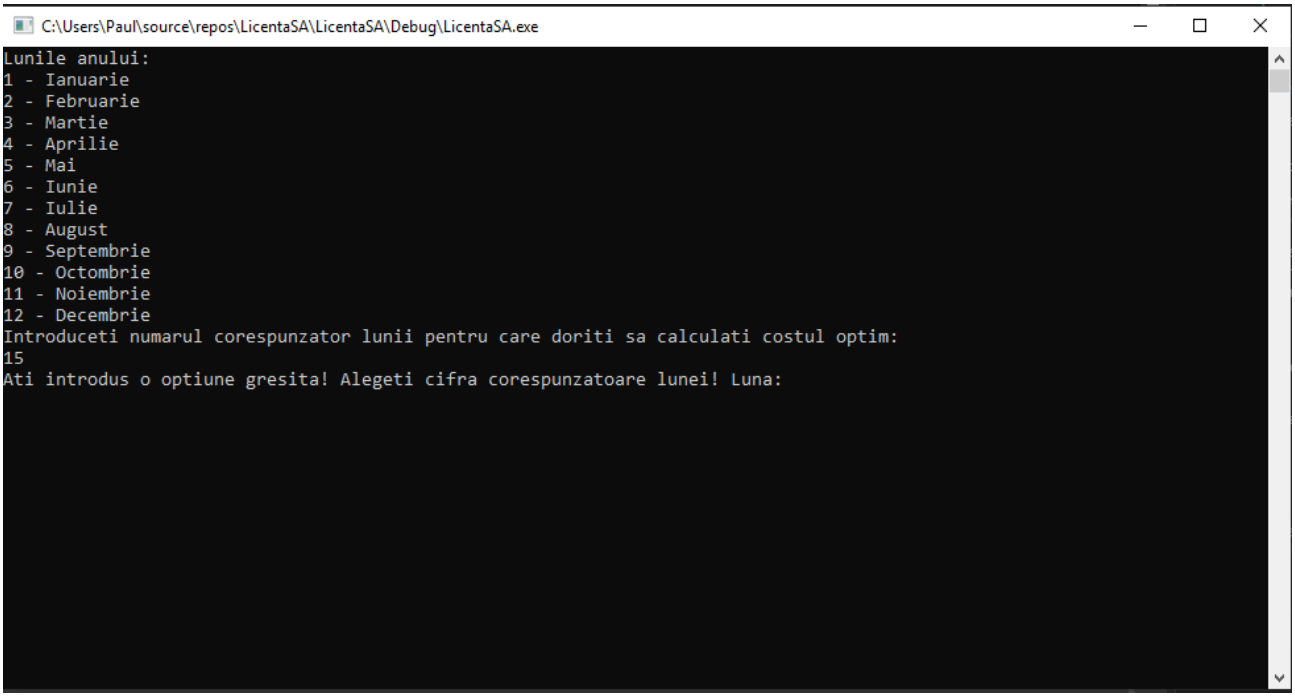


Figura 34. Cazul în care se introduce o opțiune inexistentă

5.2 Luna Ianuarie

Cost: 247.05 €.

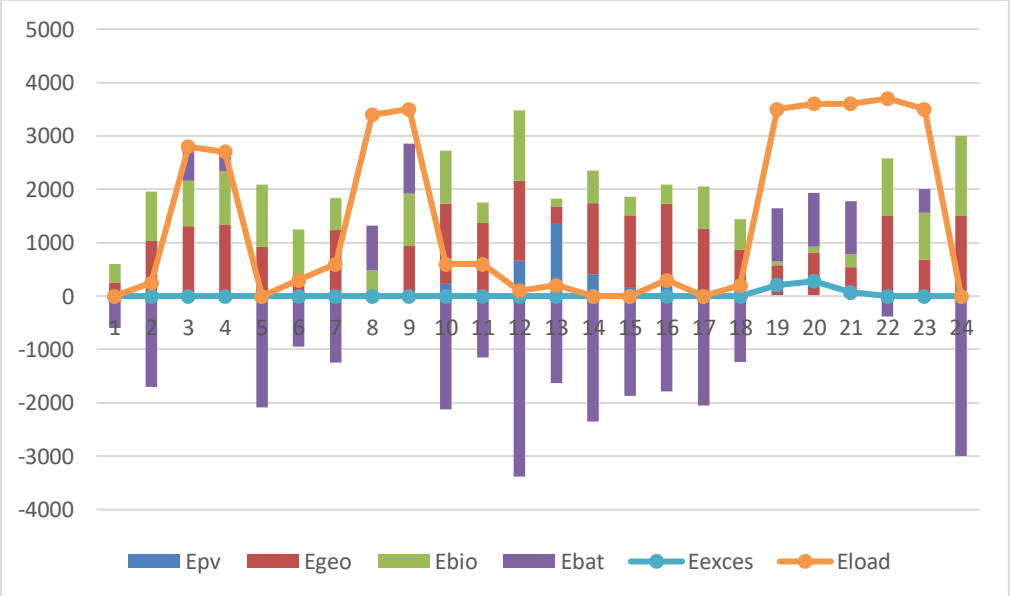


Figura 35. Rezultatele managementului energiei pentru luna Ianuarie

5.3 Luna Februarie

Cost: 45.25 €.

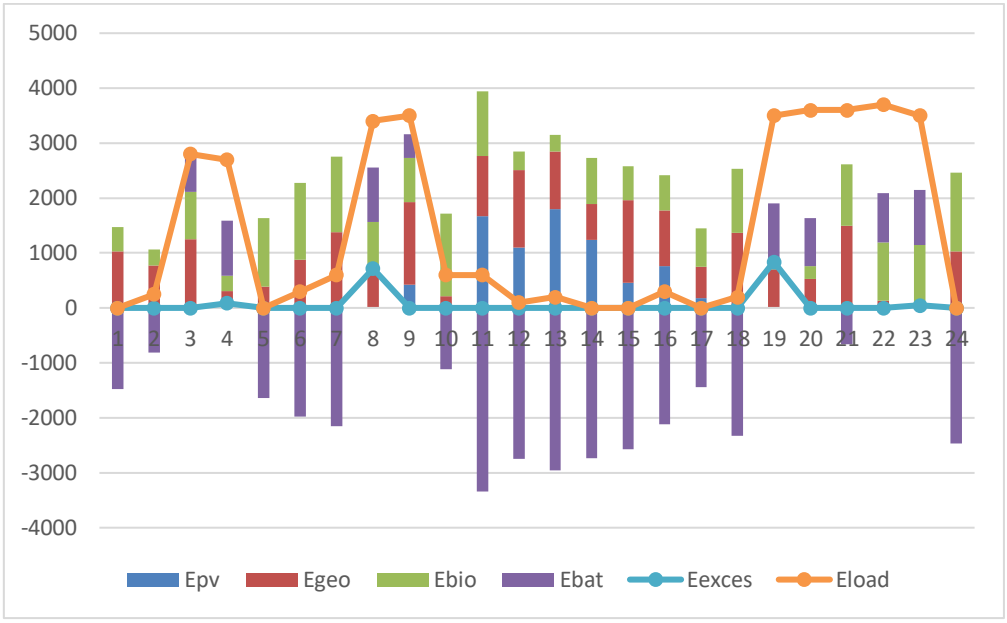


Figura 36. Rezultatele managementului energiei pentru luna Februarie

5.4 Luna Martie

Cost: 30.5643 €.

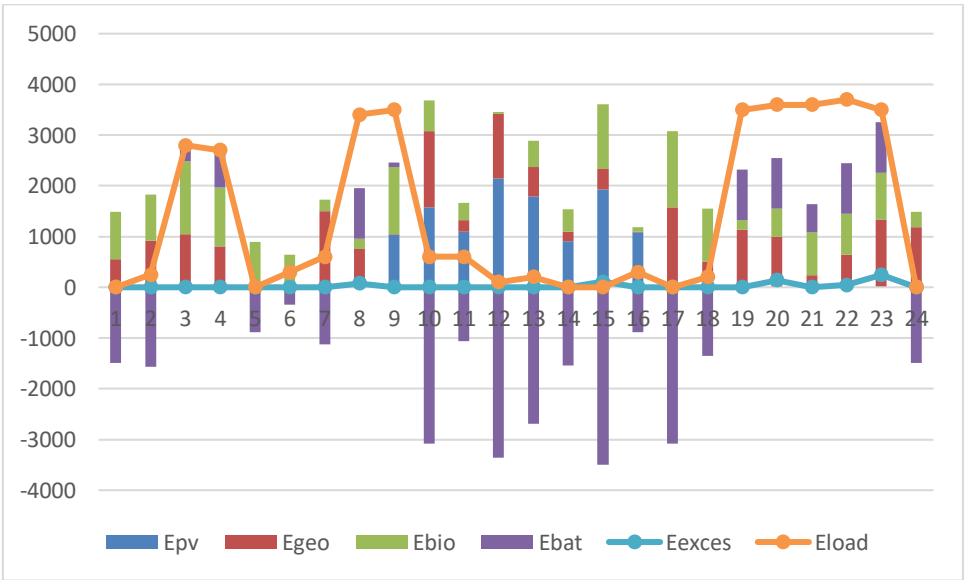


Figura 37. Rezultatele managementului energiei pentru luna Martie

5.5 Luna Aprilie

Cost: -689.65 €.

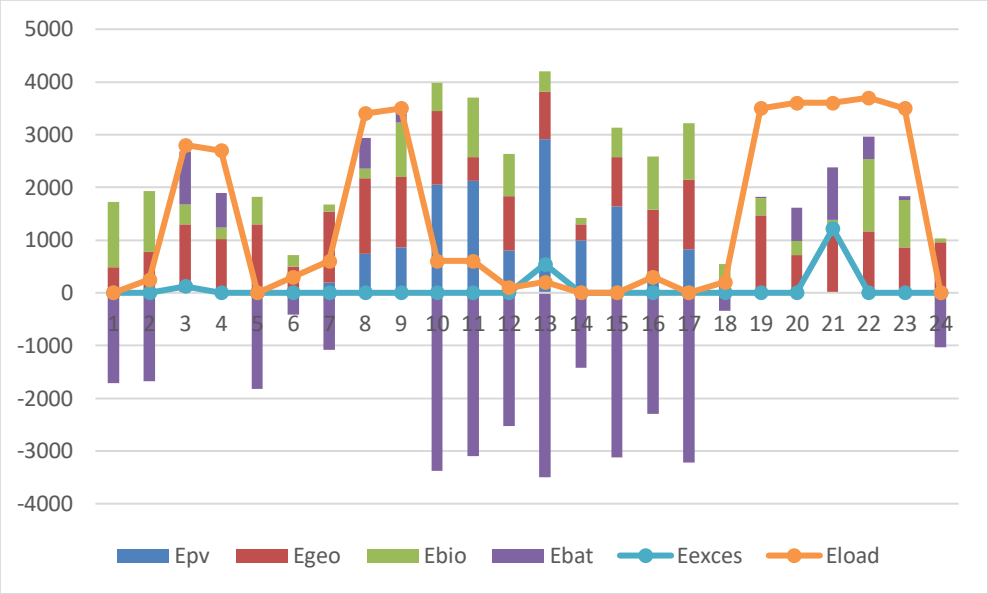


Figura 38. Rezultatele managementului energiei pentru luna Aprilie

5.6 Luna Mai

Cost: -1585.35 €.

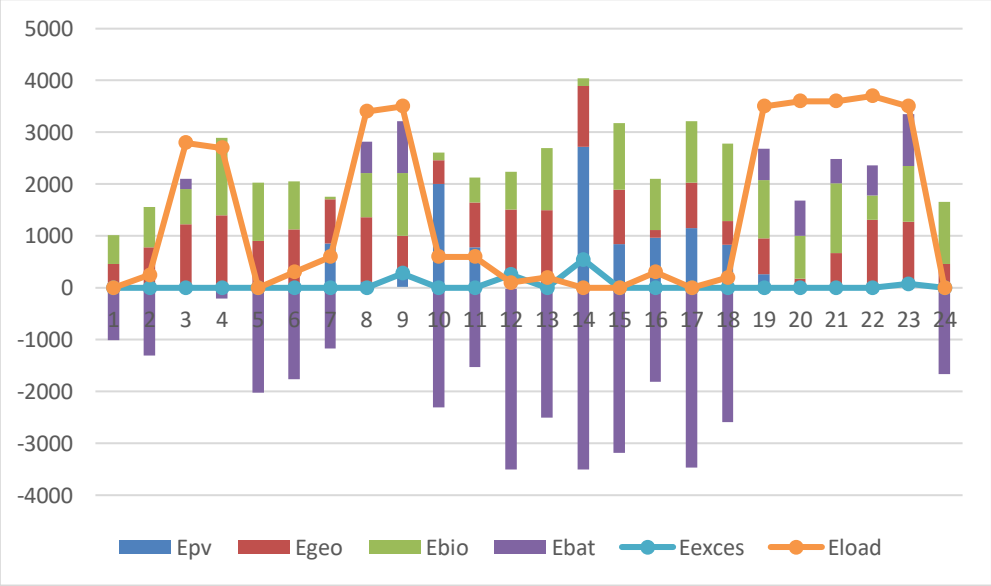


Figura 39. Rezultatele managementului energiei pentru luna Mai

5.7 Luna Iunie

Cost: -1437.25 €.

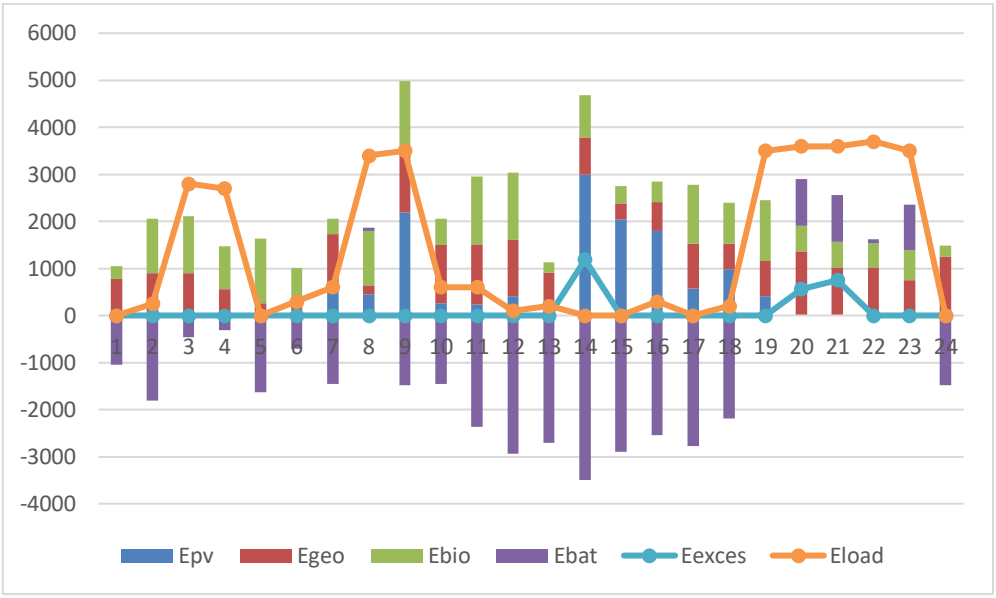


Figura 40. Rezultatele managementului energiei pentru luna Iunie

5.8 Luna Iulie

Cost: -2452.75 €.

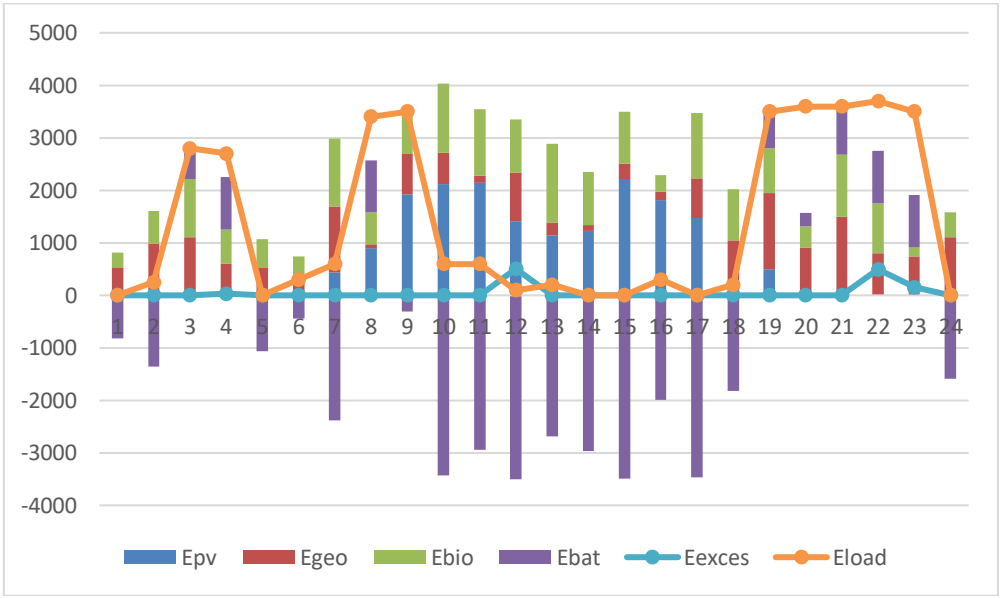


Figura 41. Rezultatele managementului energiei pentru luna Iulie

5.9 Luna August

Cost: -1884.15 €.

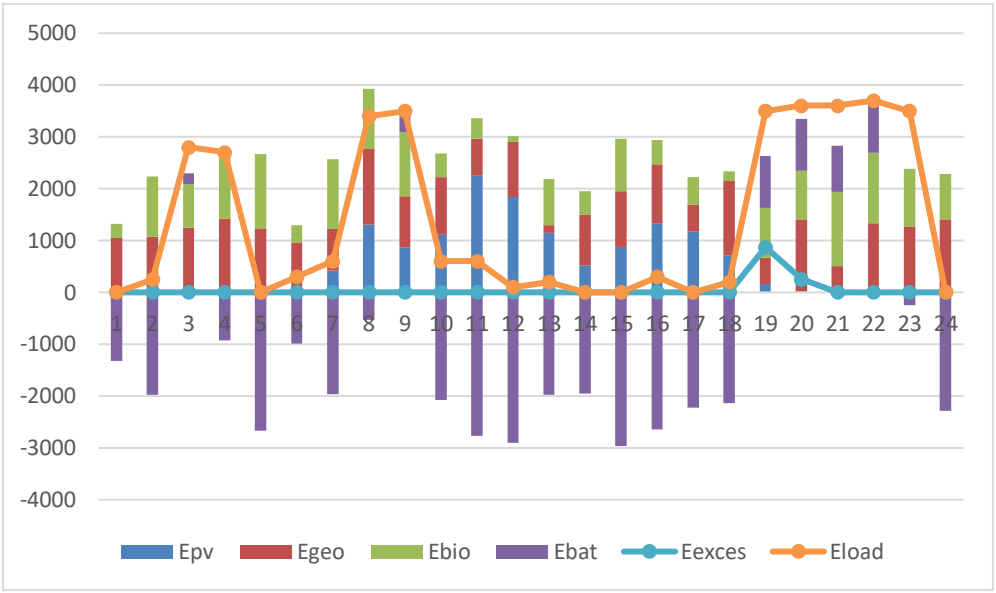


Figura 42. Rezultatele managementului energiei pentru luna August

5.10 Luna Septembrie

Cost: -1209.8 €.

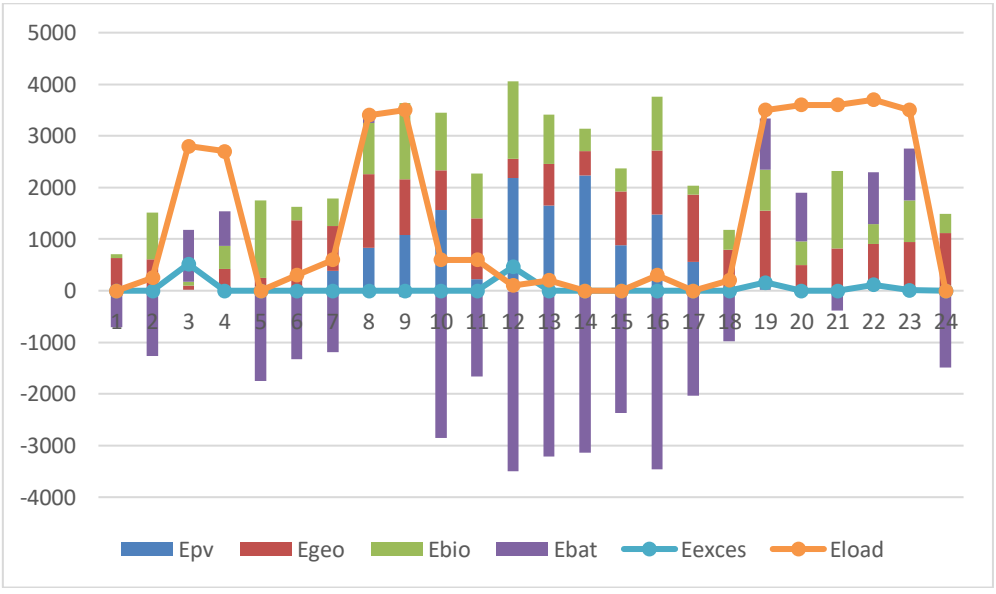


Figura 43. Rezultatele managementului energiei pentru luna Septembrie

5.11 Luna Octombrie

Cost: -84.8 €.

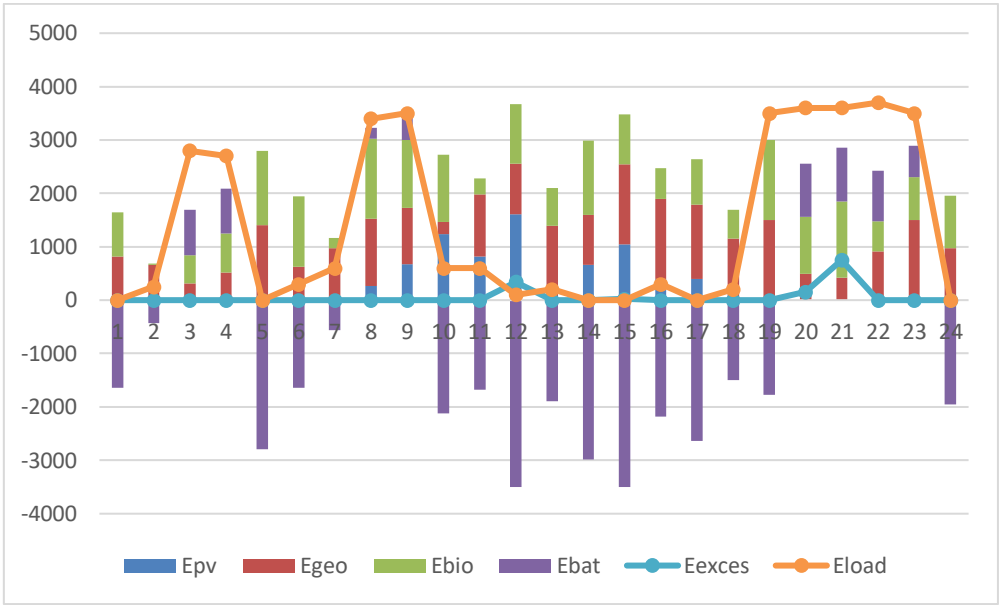


Figura 44. Rezultatele managementului energiei pentru luna Octombrie

5.12 Luna Noiembrie

Cost: 56.40 €.

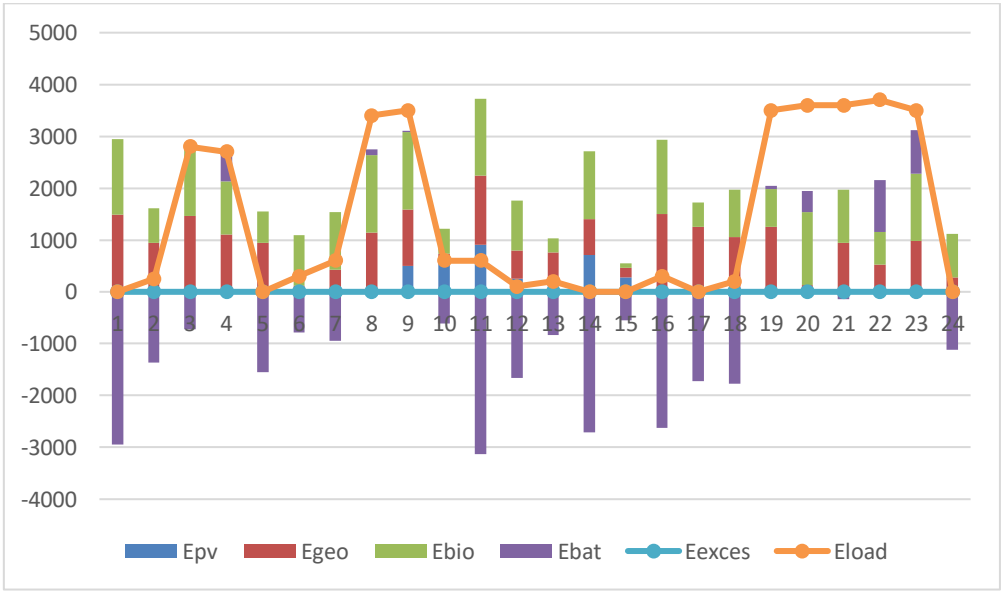


Figura 45. Rezultatele managementului energiei pentru luna Noiembrie

5.13 Luna Decembrie

Cost: 220.40 €.

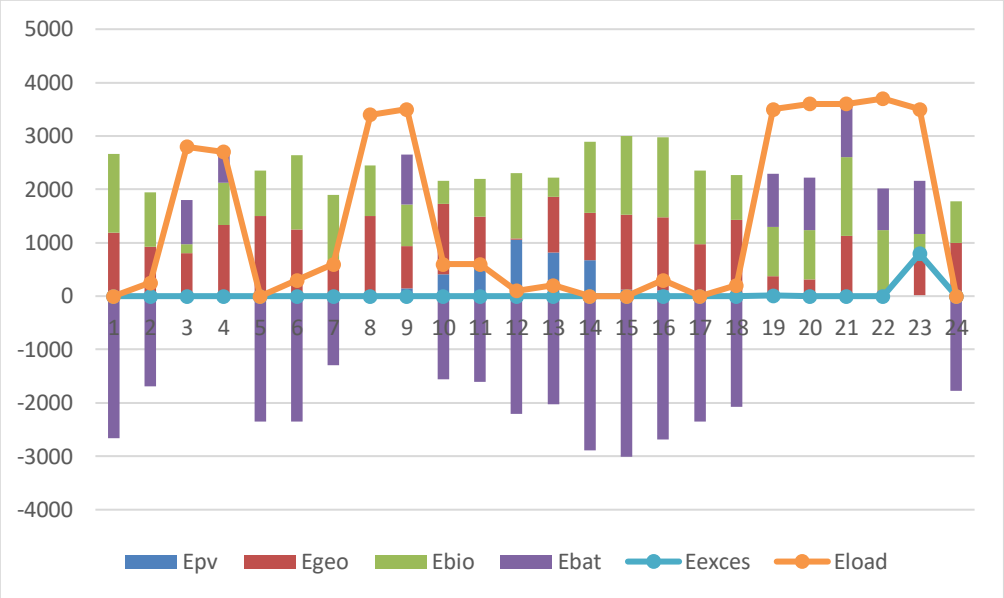


Figura 46. Rezultatele managementului energiei pentru luna Decembrie

6 Concluzii

Tabel 5. Costurile pentru cele 3 procedee folosite pentru rezolvarea problemei planificării zilnice a unei microrețele izolate

Luna	Algoritmul Harmony Search Optimization (€)	Algoritmul Particle Swarm Optimization (€)	Algoritmul Simulated Annealing (€)
Ianuarie	6.3847	6.391	247.05
Februaie	6.38715	6.3882	45.25
Martie	6.14945	6.13845	30.5643
Aprilie	5.98865	6.12471	-689.65
Mai	5.63315	5.614	-1585.35
Iunie	5.4915	5.4811	-1437.25
Iulie	5.48825	5.47914	-2452.75
August	5.6911	5.84621	-1884.15
Septembrie	6.04065	6.032	-1209.8
Octombrie	6.3674	6.489	-84.8
Noiembrie	6.3632	6.365	56.40
Decembrie	6.694	6.681	220.40

În Tabelul 5 sunt afișate costurile preluate în urma execuției fiecărui algoritm. Valorile pentru algoritmiul Harmony Search Optimization și procedeul Particle Swarm Optimization sunt preluate din [11]. Valorile introduse în coloana a 4-a din Tabel sunt preluate din Capitolul 5.

Se poate observa din Tabelul 5 că versiunea algoritmului Simulated Annealing dezoltată, implementată și testată în această lucrare de diplomă are un cost mult mai mare în unele lunii ale anului față de celelalte procedee folosite. În schimb, în lunile Aprilie, Mai, Iunie, Iulie, August, Septembrie și Octombrie, costul realizat în urma execuției procedeului Simulated Annealing are semnul negativ.

Rezultatele obținute folosind algoritmul Simulated Annealing ne arată că este o soluție viabilă pentru rezolvarea problemei planificării zilnice a unei microrețele izolate. Cererea de sarcină este satisfăcută de sursele de energie regenerabile utilizate.

Avantajul acestui procedeu este timpul său de execuție. Procedeul Simulated Annealing realizează toate calculele necesare în mai puțin de o oră. În schimb, dezavantajul acestei versiuni implementate este că pe lângă încărcarea bateriilor disponibile, produce și un exces de energie.

7 Bibliografie

[1] Geem ZW, Kim JH, Loganathan GV "A new heuristic optimization algorithm: Harmony Search", Simulation, vol. 76, pp. 60-68, February 2001

[2] Lee KS, Geem ZW "A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice", Computer Methods in Applied Mechanics and Engineering, vol. 194, pp. 3902-3933, September 2005

[3] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942-1948

[4] B. Borowska, "Nonlinear inertia weight in particle swarm optimization," 2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, 2017, pp. 296-299.

[5] www.smartgrids.eu

[6] www.galvinelectricity.org

[7] Comisia Europeană – "Vision and Strategy for Europe's Electricity Networks of the Future", Platforma Tehnologică Europeană SmartGrids, 2008

[8] ERPI – "The integrated Energy and Communication Systems Arhitecture", Vol. IV, Technical Analysis, Electric Power Research Institute, 2004

[9] Aftab Ahmad Khan, Muhammad Naeem, Muhammad Iqbal, Saad Qaisar, Alagan Anpalagan, „Renewable and Sustainable Energy Reviews", Volume 58, May 2016, pp. 1664-1683

[10] Eniko Lazar, Andreea Ignat, Dorin Petreus, Radu Etz "Energy Management for an Islanded Microgrid Based on Harmony Search Algorithm", 41st International Spring Seminar on Electronics Technology (ISSE), 2018

[11] Andreea Ignat, Eniko Lazar, Dorin Petreus "Energy Management for an Islanded Microgrid Based on Particle Swarm Optimization", IEEE 24th International Symposium for Design and Technology in Electronic Packaging (SIITME), 2018

[12] https://en.wikipedia.org/wiki/GitHub

[13] https://en.wikipedia.org/wiki/Git

[14] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[15] https://en.wikipedia.org/wiki/Simulated_annealing

[16] Lucian Toma, "Rețele electrice inteligente", Suport de curs, Universitatea Politehnică din București

8 Anexe

Din următoarele capturii de ecran, efectuate asupra output-ului aplicației pentru fiecare lună a anului, s-au folosit informațiile necesare pentru a se realiza graficele de la Capitoul 5.

- Luna Ianuarie

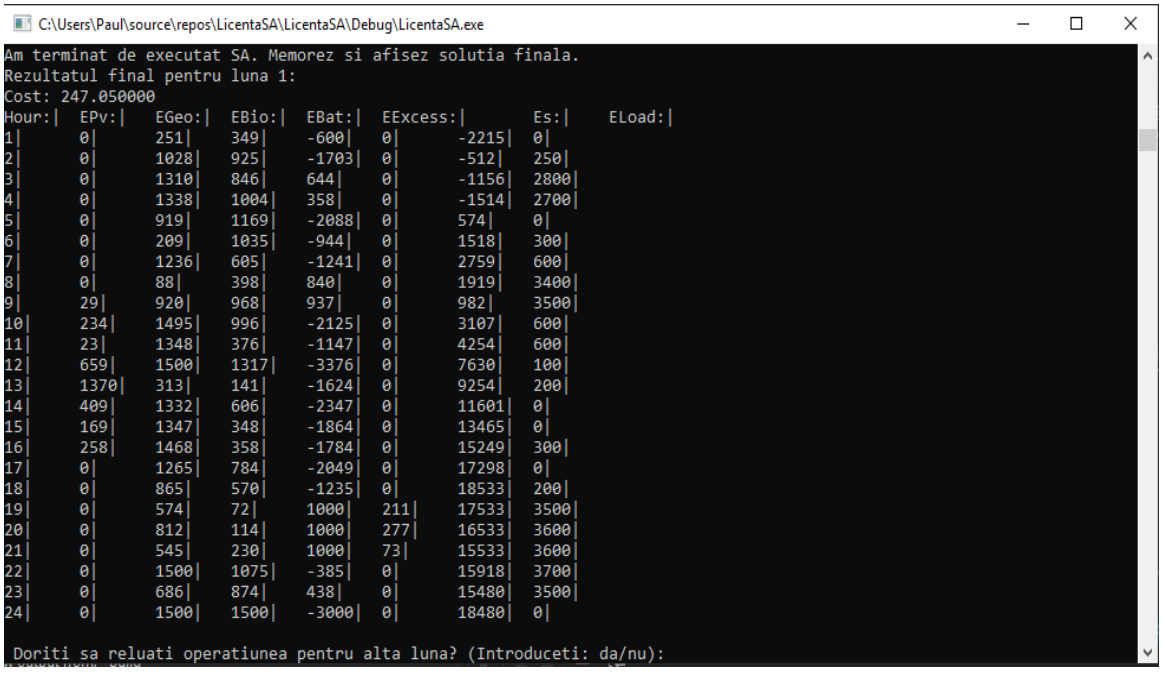


Figura 47. Costul pentru luna Ianuarie

- Luna Februarie

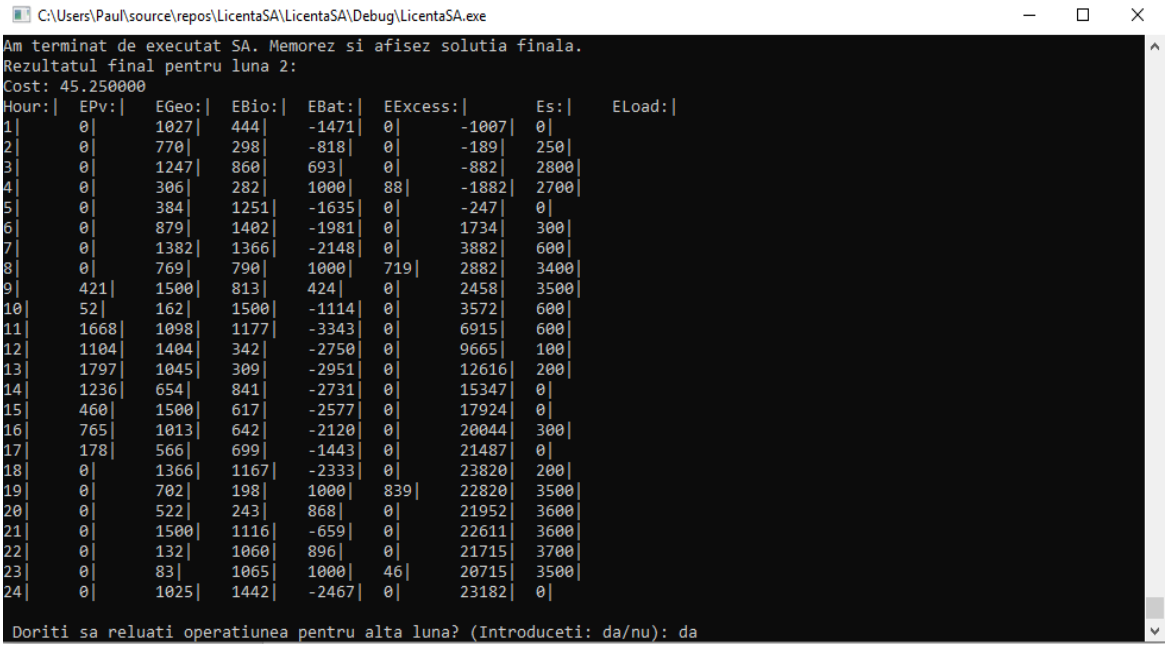


Figura 48. Costul pentru luna Februaie

- Luna Martie

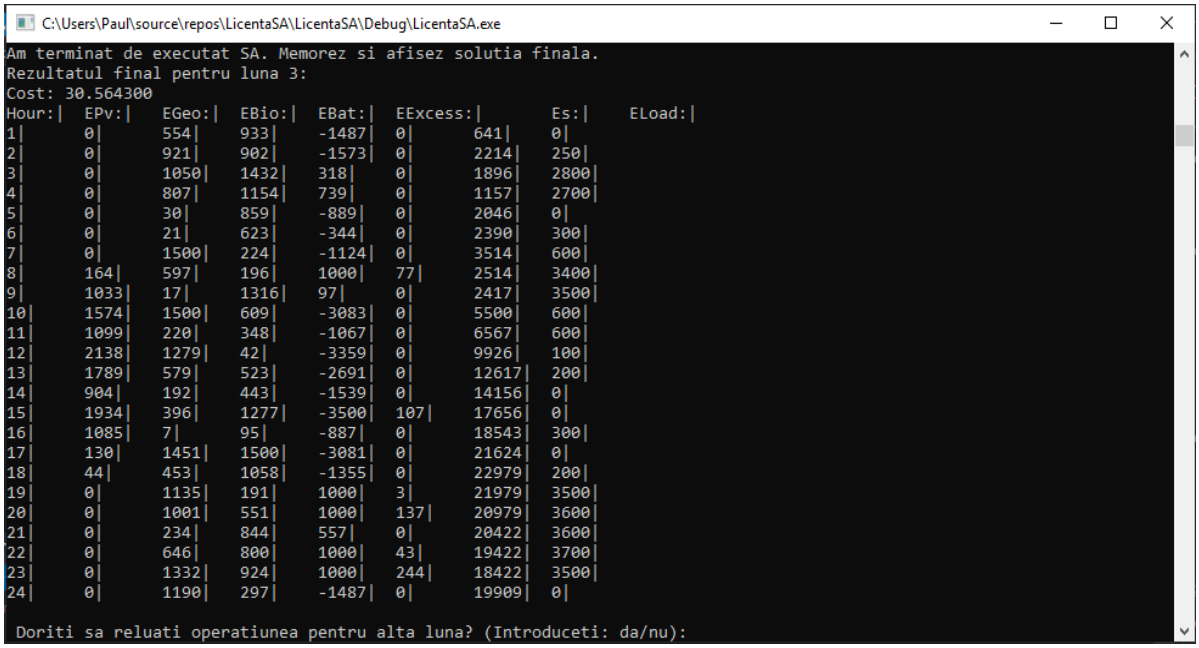


Figura 49. Costul pentru luna Martie

- Luna Aprilie

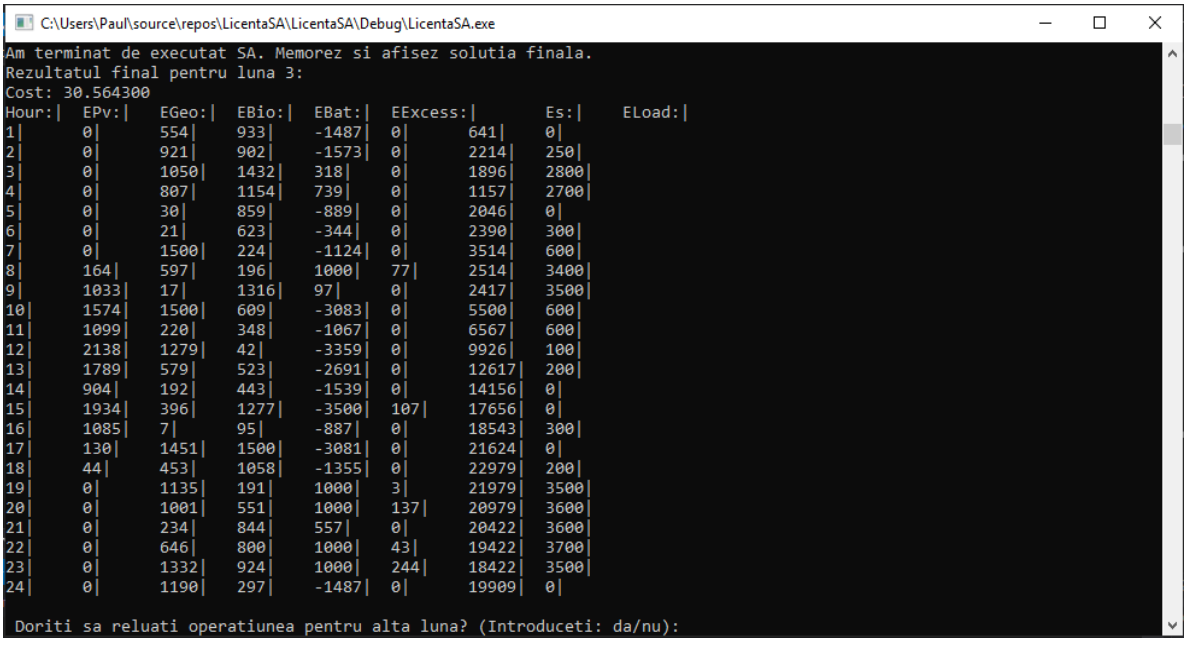


Figura 50. Costul pentru luna Aprilie

- Luna Mai

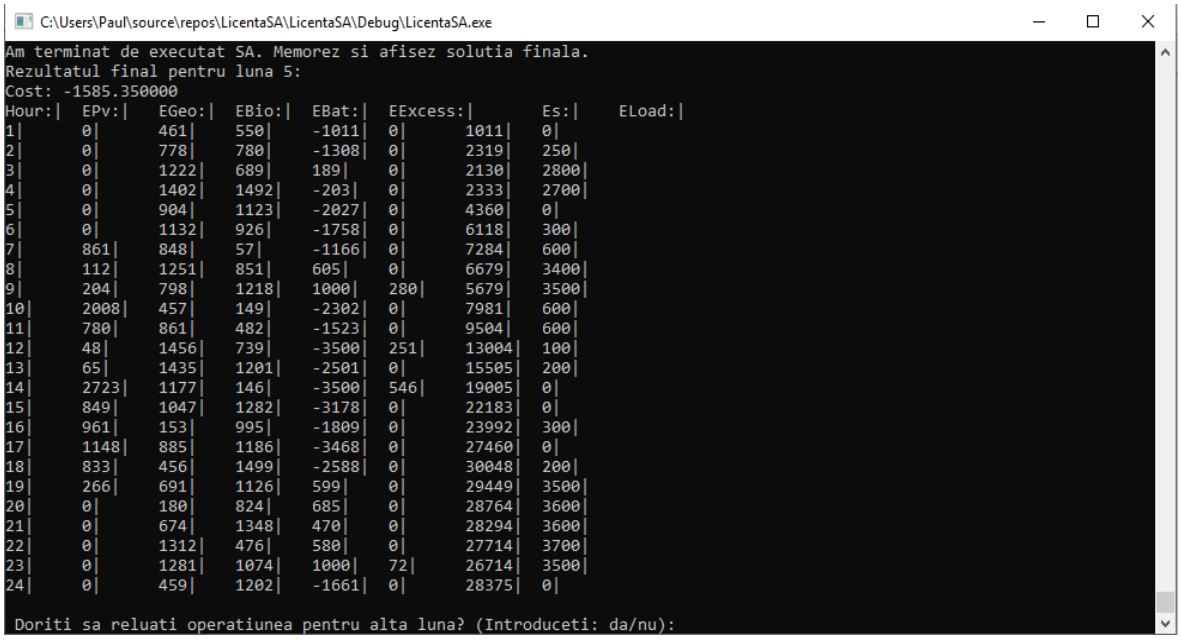


Figura 51. Costul pentru luna Mai

- Luna Iunie

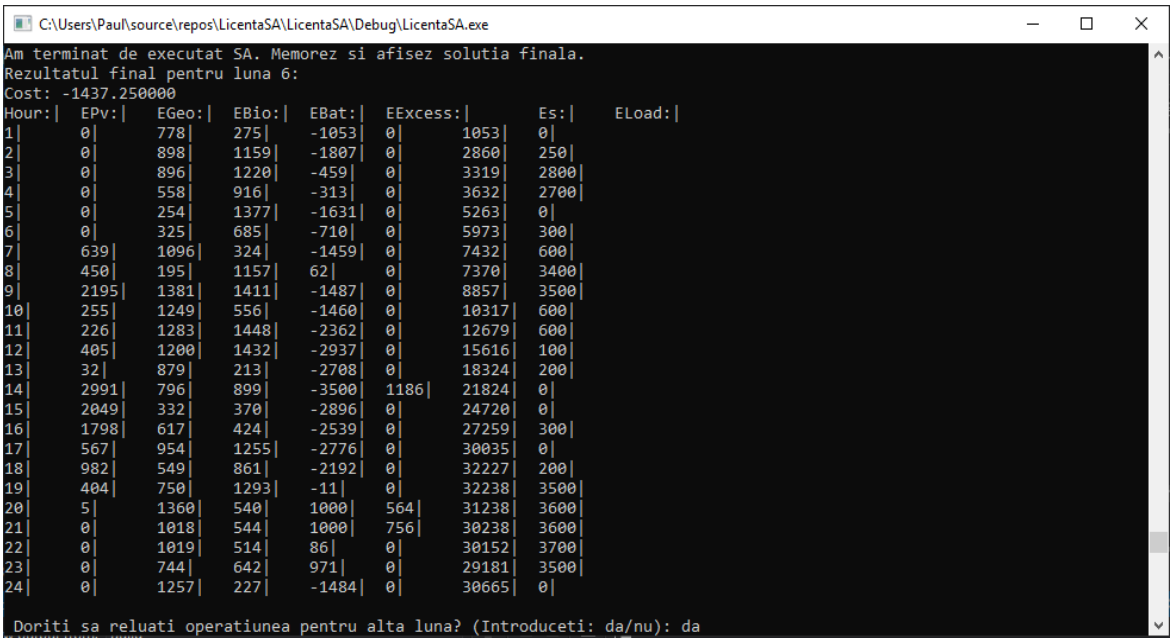


Figura 52. Costul pentru luna Iunie

- Luna Iulie

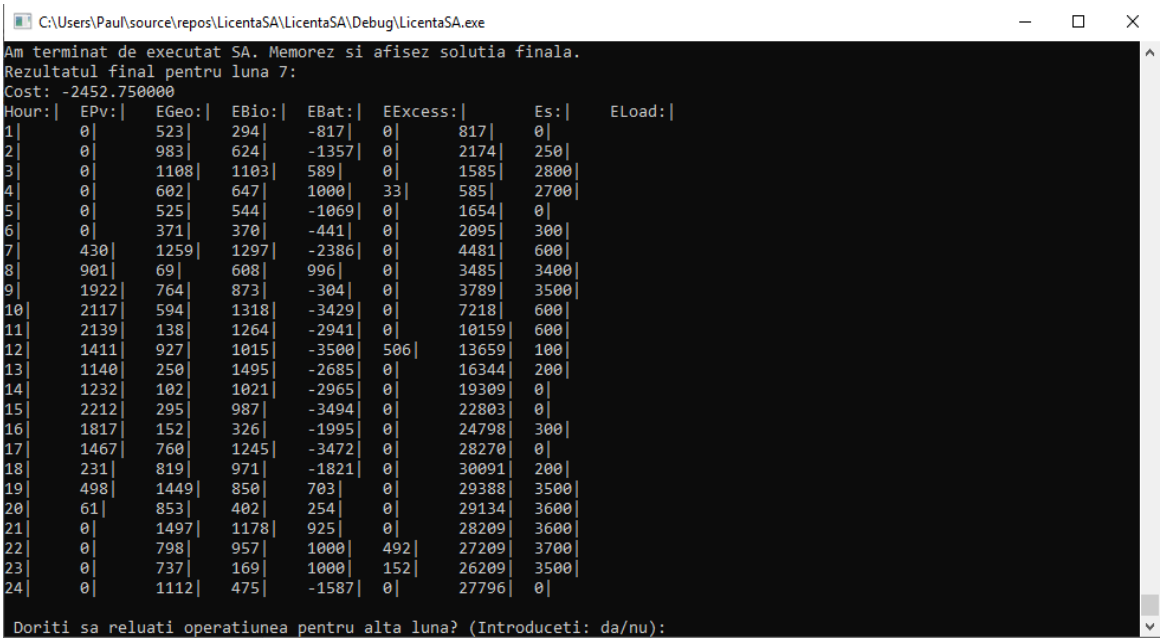


Figura 53. Costul pentru luna Iulie

- Luna August

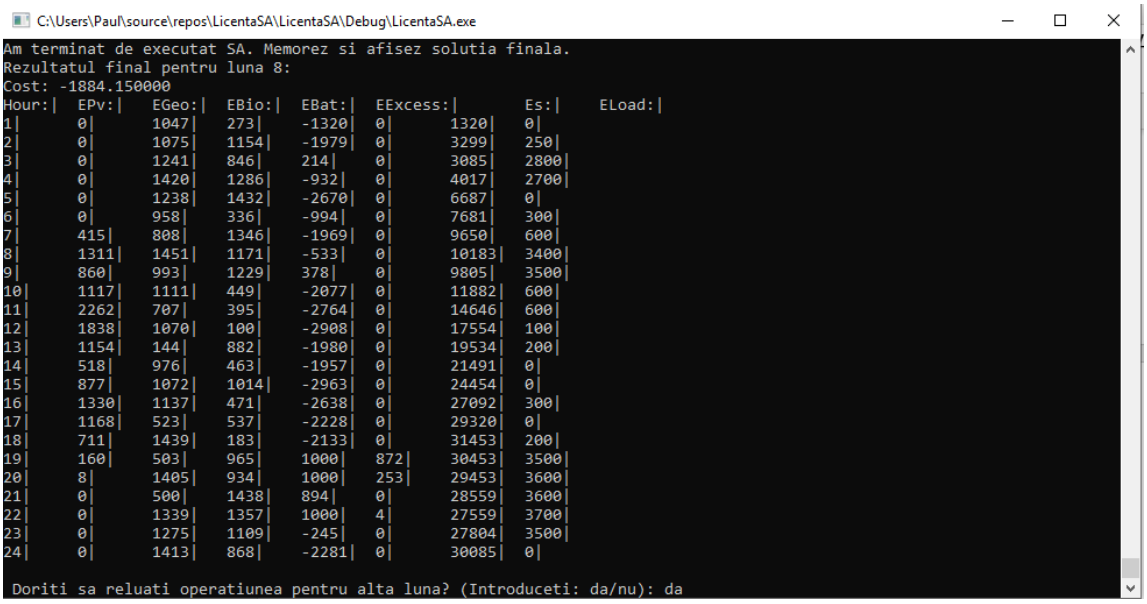


Figura 54. Costul pentru luna August

- Luna Septembrie

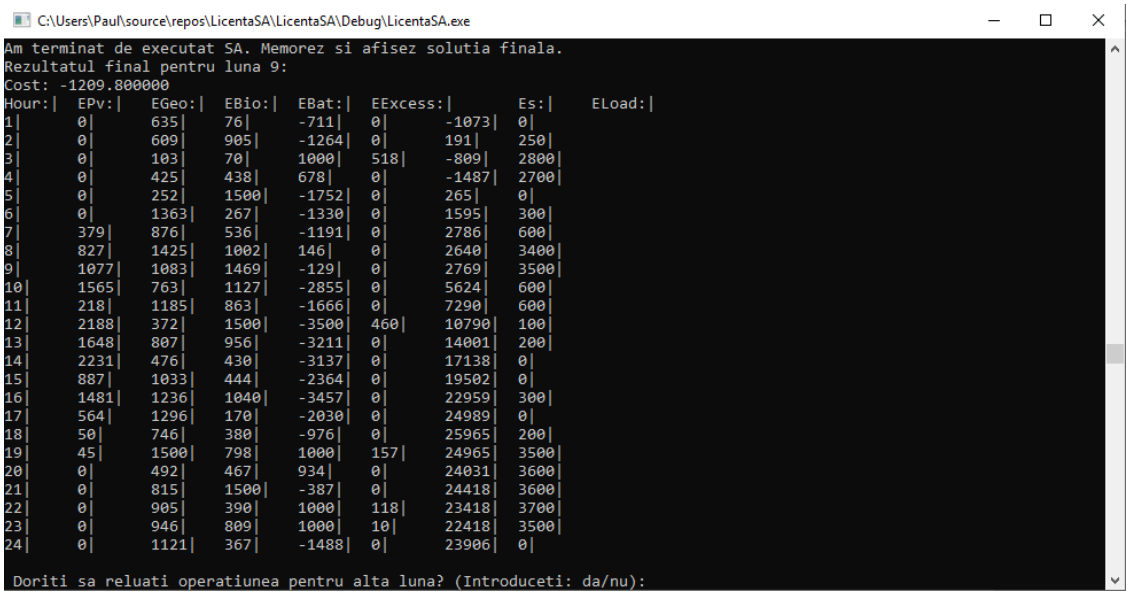


Figura 55. Costul pentru luna Septembrie

- Luna Octombrie

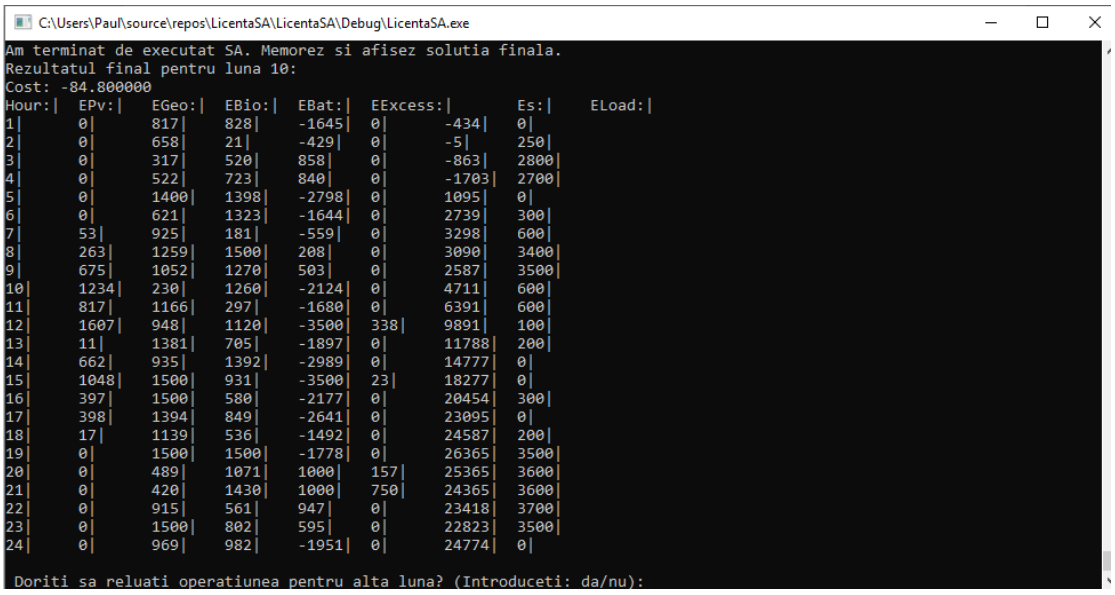


Figura 56. Costul pentru luna Octombrie

- Luna Noiembrie

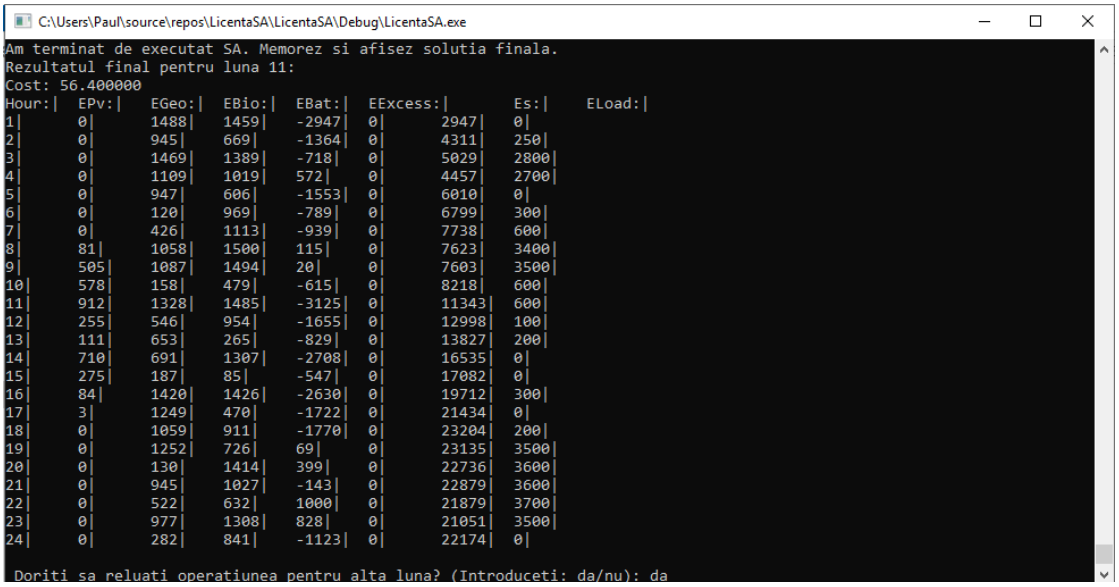


Figura 57. Costul pentru luna Noiembrie

- Luna Decembrie

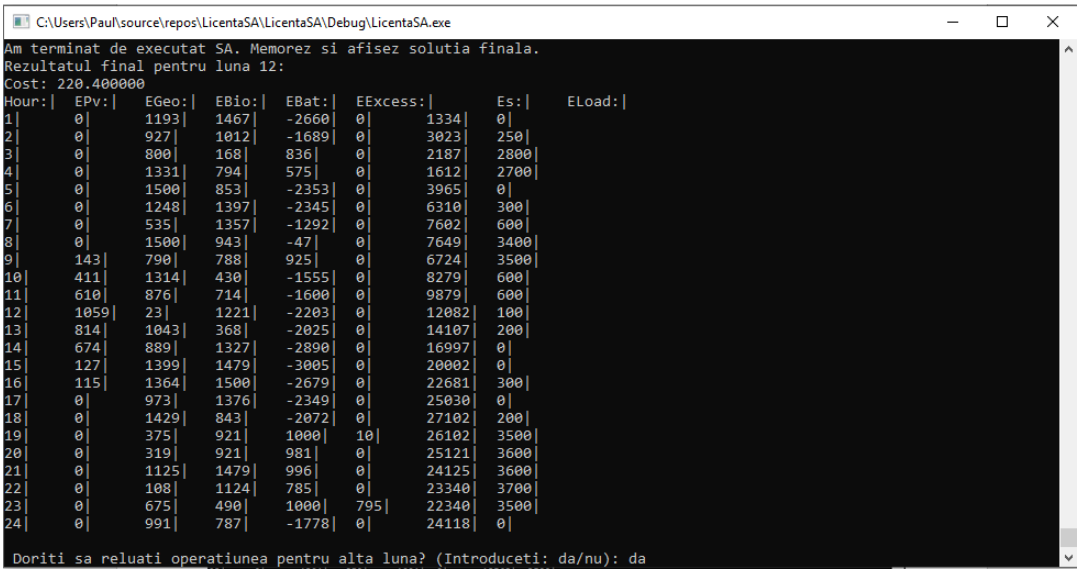


Figura 58. Costul pentru luna Decembrie

9 CV-ul autorului

INFORMAŢII PERSONALE Adrian-Paul Gheorghian

Strada Putnei, Nr. 1, 725400 Rădăuți (România)

(+40) 742 38 2755

gheorghian.paul@gmail.com

https://www.linkedin.com/in/gheorghianpaul/

Sexul Masculin | Data naşterii 12/10/1995 | Naţionalitatea română

STUDIILE PENTRU CARE SE CANDIDEAZĂ Diplomă de inginer

EXPERIENŢA PROFESIONALĂ

03/06/2019–Prezent — Application Software Engineer

Magneti Marelli Automotive Cluj S.R.L., Cluj-Napoca (România)

Dezvoltarea unui proiect pe partea de Telematics utilizând tehnologiile: C/C++, diagrame UML, GIT, Python, BASH, UNIX etc.

02/07/2018–10/08/2018 — Software Engineer Intern

Magneti Marelli Automotive Cluj S.R.L., Cluj-Napoca (România)

- Instruire asupra tehnologiilor QT, QML, JSON, C++ și GIT
- Dezvoltarea unui State Machine
- Dezvoltarea unui proiect HMI (Human Machine Interface)
- Depanarea unui proiect
- Dezvoltarea unui proiect în industria automotive
- Proiectul implementează o modalitate de comunicare V2X și V2V în era autonomă de conducere

EDUCAŢIE ŞI FORMARE

02/10/2015–16/07/2019 — Diplomă de Inginer

Universitatea Tehnică din Cluj-Napoca, Cluj-Napoca (România)

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Specializarea Electronică Aplicată

01/09/2016–02/07/2017 — Atestat Analist Programator

Academy+Plus Cluj-Napoca, Cluj-Napoca (România)

- Unix, HTML+CSS,GitHub,PHP,SQL
- C

10/09/2011–02/06/2015 — Diplomă de Bacalaureat

Colegiul Național "Eudoxiu Hurmuzachi", Rădăuți (România)

Matematică-Informatică Intensiv Informatică

COMPETENŢE PERSONALE

Limba(i) maternă(e) română

Limbile străine

	ÎNȚELEGERE		VORBIRE		SCRIERE
	Ascultare	Citire	Participare la conversație	Discurs oral	
engleză	B1	B1	B2	B1	B2
	Diploma de Bacalaureat				
germană	A2	A2	A1	A1	A1

Niveluri: A1 și A2: Utilizator elementar - B1 și B2: Utilizator independent - C1 și C2: Utilizator experimentat
Cadrul european comun de referință pentru limbi străine

Competenţe de comunicare

- bune abilități de comunicare dobândite în urma experienței mele ca voluntar în Organizația Studenților din Cluj-Napoca

Competenţe organizaţionale/manageriale

- bune abilități de organizare dobândite în urma organizării campionatului de PaintBall din cadrul Organizației Studenților din Universitatea Tehnică
- bune abilități de conducere a unei echipe dobândite în urma campionatelor de luptă pe echipe din cadrul clubului de arte marțiale HONG-HA Rădăuți

Competenţe dobândite la locul de muncă

- o bună cunoaștere a proceselor de dezvoltare software

Competențele digitale

AUTOEVALUARE				
Procesarea informației	Comunicare	Creare de conținut	Securitate	Rezolvarea de probleme
Utilizator experimentat	Utilizator experimentat	Utilizator experimentat	Utilizator experimentat	Utilizator experimentat

Competentele digitale - Grilă de auto-evaluare

ECDL – European Computer Driving Licence

Permis de conducere

B

INFORMAŢII SUPLIMENTARE

Distincţii

Locul 3 pe județ la Olimpiada de Informatică Aplicată (OTI) în anul 2012

Proiecte

Robot cu roți omniwheel controlat prin portul serial
-feature: implementarea modului autonom în proporție de 25%
Proiectul a fost abandonat.