Алгоритм Знаходження Максимального Потоку для Управління Комунікаційними Режимами Електричних Мереж в Системі «Smart City»

Ілона Ревенчук
кафедра програмної інженерії
Харківський національний університет
радіоелектроніки
Харків, Україна
ilona.revenchuk@nure.ua

Анастасія Чуприна
кафедра програмної інженерії
Харківський національний університет
радіоелектроніки
Харків, Україна
anastasiya.chupryna@nure.ua

Algorithm for Finding Maximum Flow for Managing Communication Modes of Electric Networks in the Smart City System

Ilona Revenchuk
Software Engineering Department
Kharkiv National University
of Radioelectronics
Kharkiv, UKRAINE,
ilona.revenchuk@nure.ua

Anastasia Chupryna
Software Engineering Department
Kharkiv National University
of Radioelectronics
Kharkiv, UKRAINE,
e-mail: anastasiya.chupryna@nure.ua

***Анотація*—В статті представлено підхід застосування Big Data для розумного міста. Цей підхід включає кілька простих систем автоматизації. На самому базовому рівні ці системи можуть бути не більше ніж рефлекторними агентами: агентами, які спостерігають навколишнє середовище та реагують відповідно до отриманої інформації. Smart city, що складається з таких простих рефлекторних агентів, може бути використане для спрощення життя своїх мешканців, для покращення комфорту та використання в системі управління електричною мережею завдяки алгоритму знаходження максимального потоку для управління комунікаційними режимами.**

***Abstract*—The article presents way to apply big data for smart city. This approach includes few simple automation systems. At the most basic level, these systems could be no more than reflex agents: agents which observe the environment and react accordingly to the acquired information. A smart city constituted of such simple reflex agents can be exploited to simplify the life of its residents, to improve the comfort and used for electrical network management system via algorithm is to find through paths with positive flows from source to drain.**

***Ключові слова*—Smart city, big data, система управління електричною мережею, алгоритм максимального потоку, графік мережі, електрична мережа.**

***Keywords*—Smart city, big data approach, electrical network management system, maximum flow algorithm, graph of network, electrical network.**

I. Вступ

Регулювання енергетичних потоків в умовах динамічної зміни мережі енергопостачання, за умови високої надійності та в умовах впровадження системи “Smart City”, є задача високої складності [1].

Для вирішення цієї задачі можна використовувати алгоритми знаходження максимального потоку в мережі. Використання нових швидких алгоритмів покращить аналіз управління потоками в енергосистемах. І допоможе впровадженню системи «Smart City» із запобіганням аварійним ситуаціям що виникають за умови помилкових дій диспетчерських служб заснованих на застарілих даних.

В роботі представлений алгоритм знаходження максимального потоку для управління комунікаційними режимами, загальна модель, розрахунок максимального потоку в мережі електропостачання в реальному часі, при любих змінах конфігурації електричної мережі і практична реалізація аналізу впливу перемикань на стан мережі.

II. Реалізація алгоритму

Диспетчерські центри по управлінню електричними мережами здійснюють безперервний контроль за станом об'єктів – лініями електропередач, трансформаторними підстанціями, розподільними пунктами. Інформація про це у графічній формі відображається на мнемосхемі електричних мереж [2].

Для ефективного управління електричними мережами використовують оперативне перемикання в мережі електропостачання. Інформація про пропускну здатність електромережі необхідна для прийняття рішення про допустимість оперативного підключення, - передачі частки навантаження з одної дільниці мережі на іншу.

Ділянка електричної мережі може бути представлена як орієнтований зважений граф див. рис.1.

Вузлам мережі в даному випадку будуть асоційовані трансформатори, перемикачі, роз'єднувачі або трансформаторні підстанції чи комутаційні групи, в залежності від деталізації схеми живлення. А дугам мережі асоційовані лінії електропередач, шини розподільних пристроїв тощо. Мінімальною пропускною здатністю дуг буде максимально допустимий струм для ліній або шин. Дуги які направлені в стік мають пропускну здатність порівняну з потужністю трансформаторних підстанцій. Вузли які асоційовані з розподільчими пристроями не зв'язані зі стоком, бо ці вузли мають змінну степінь і приймають участь тільки в перерозподілі потоку [2].

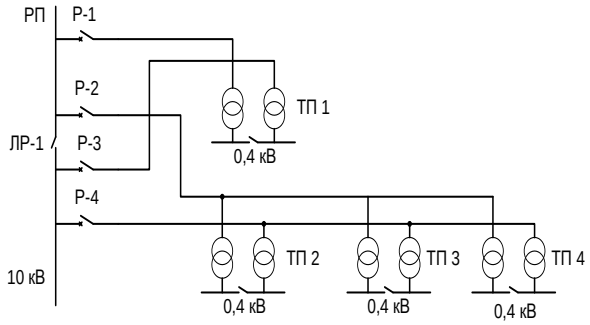

Рис. 1. Приклад схеми електропостачання для 4 ТП

Для представлення електричної мережі яка зображена на рис/ 1 у вигляді транспортної мережі треба додати джерело живлення, якщо воно відсутнє на схемі мережі, в даному випадку вузол S, та стік мережі T. Однак схема має точку розділу мережі – лінійний роз'єднувач ЛР-1, тому перед вузлом ЛР-1 додані два допоміжних вузли S1, S2. Вся комутаційна апаратура конвертується в вузли мережі, а лінії в дуги мережі. Виходи трансформаторів з'єднуються зі стоком через проміжні вузли Т1, Т2 (рис. 2). При перемиканнях в мережі вона динамічно змінюється тому постійно треба слідкувати за перерозподілом потоків в мережі.

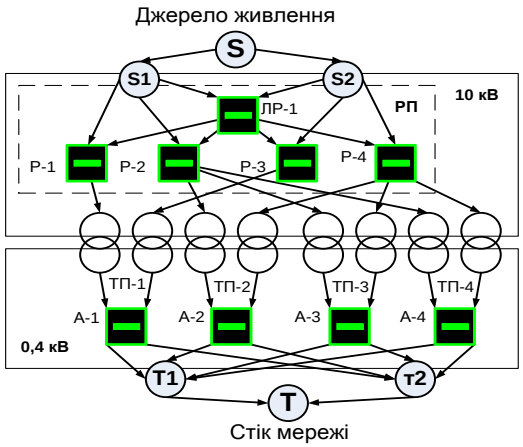

Рис.2 – Транспортна мережа схеми з 4 ТП

На рис. 3 мережа показана в одному з станів після перемикань, на даний момент ввімкнені ЛР-1, Р-1, Р-2 а також під напругою знаходяться по одному трансформатору кожної ТП, та секціонуючи вимикачі на ТП.

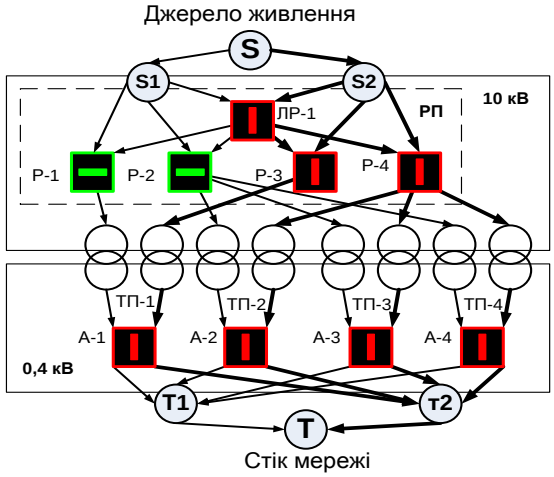

Рис.3. Транспортна мережа при перемиканнях

Актива частина мережі виділена жирними лініями, ця частина мережі трансформується в орграф з одним джерелом та одним стоком (рис. 4).

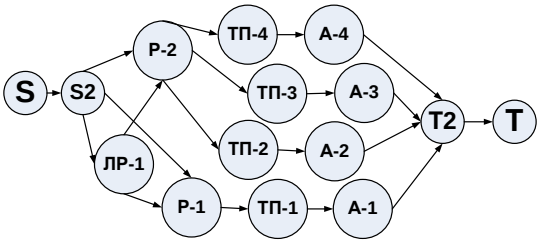

Рис.4. Граф мережі

Після цього кожній дузі задається пропускна здатність, яка дорівнює максимально допустимому рівню струму котрий може проходити через лінії та шини асоційовані з дугою [2].

Якщо з дугою асоційована не одна лінія або секція шин, а група ліній, або ділянка мережі чи фідер, то максимальна пропускна здатність дуги буде дорівнювати значенню яке може пропустити через себе самий слабий елемент в групі яка асоційована з дугою.

III. Алгоритм знаходження максимального потоку

Для пошуку максимального потоку в мережі реалізовано алгоритм на базі PushRelabel методу, як простий в реалізації та з задовільною швидкодією. Ідея алгоритму полягає в знаходженні наскрізних шляхів з позитивними потоками від джерела до стоку [3].

Розглянемо ребро *(i, j)* з начальною пропускною здатністю (C_{ij}, C_{ji}). В процесі виконання алгоритму частки цих пропускних здатностей «забираються» потоками, які проходять через дане ребро. Як результат кожне ребро буде мати остаточну пропускну здатність (c_{ij}, c_{ji}). Мережа де всі ребра мають остаточну пропускну здатність називається остаточна. Для вузла *j*, який отримує потік від вузла *i*, визначимо мітку $[a_j, i]$, де a_j – величина потоку, від вузла *j* до вузла *i*.

Крок 1. Для всіх ребер *(i, j)* встановимо пропускну здатність яка дорівнює начальній пропускній здатності $(c_{ij}, c_{ji}) = (C_{ij}, C_{ji})$. Призначимо $a_1=\infty$ та помітимо вузол 1 міткою [∞,-]. Призначаємо $i = 1$ і йдемо до 2-ого кроку.

Крок 2. Визначаємо множину S_i як множину вузлів *j*, в які можна перейти з вузла і по ребру з позитивною пропускною здатністю ($c_{ij}>0$ для всіх $j \in S_i$). Якщо S_i множина непуста, виконуємо 3-ій крок, в противному випадку переходимо на крок 4.

Крок 3. В множині S_i знаходимо вузол k, такий, що $c_{ik} = max\{c_{ij}\}$ для всіх *j*, які належать до S_i. Дорівнюємо $a_k = c_{ik}$ та помічаємо вузол *k* міткою $[a_k, i]$. Якщо міткою помічений вузол стоку ($k = n$), наскрізний путь знайдено, переходимо до кроку 5.

Крок 4. (Відкат назад): Якщо $i=1$, наскрізний путь неможливо знайти, переходимо до кроку 6. Якщо і не равно 1, знаходимо помічений вузол r безпосередньо попередній вузлу *i*, та видаляємо вузол *i* з множини вузлів суміжних з вузлом *r*. Дорівнюємо $i = r$ і повертаємось к 2-ому кроку.

Крок 5. (Визначення остаточної мережі): Позначимо через $N_p = \{1, k_1, k_2, ..., n\}$ множину вузлів, через які проходить *p*-й знайдений наскрізний путь від джерела до стоку. В цьому випадку максимальний потік по цьому шляху обчислюється як $f_p = min\{a_1,a_2,..., a_n\}$. Інші пропускні здатності ребер, які знаходяться в даному наскрізному шляху, зменшаться на величину f_p в напряму руху потоку, і збільшаться на ту ж величину в зворотному напрямку. Таким чином, для ребра *(i, j)*, яке входить до наскрізного потоку, текучі остаточні пропускні здатності (c_{ij}, c_{ji}) зміняться наступним чином:

а) $(c_{ij} - f_p, c_{ji} + f_p)$, якщо потік йде від вузла *i* до вузла *j*;

б) $(c_{ij} + f_p, c_{ji} - f_p)$, якщо потік йде від вузла *j* до вузла *i*.

Далі відновлюємо всі вузли видалені на 4 кроці. Встановлюємо $i = 1$ та повертаємося к 2 кроку для пошуку нового наскрізного шляху.

Крок 6. При *m* знайдених наскрізних путях максимальний потік обчислюється по формулі $F = f_1 + f_2 + ... + f_m$.

Маючи значення начальних пропускних здатностей (C_{ij}, C_{ji}) і остаточних пропускних здатностей (c_{ij}, c_{ji}) ребра *(i, j)*, можна обчислити оптимальний потік через нього наступним чином: $(a, b) = (C_{ij}-c_{ij}, C_{ji}-c_{ji})$. Якщо $a>0$, поток який проходить через ребро $(i, j) = a$. У випадку коли $b>0$ поток дорівнює *b*. Випадок коли одночасно $a>0$ і $b>0$ ніколи не станеться.

Розглянемо роботу алгоритму на тестовій мережі, яка зображена на рис. 5, матриця пропускних здатностей мережі зображена на рис. 6, з виду матриці і графічного зображення мережі видно що є ребра які з'єднують вершини в обох напрямках, наприклад ребро *(1,2)* з пропускною здатністю *300*, та ребро *(2,1)* з такою ж пропускною здатністю.

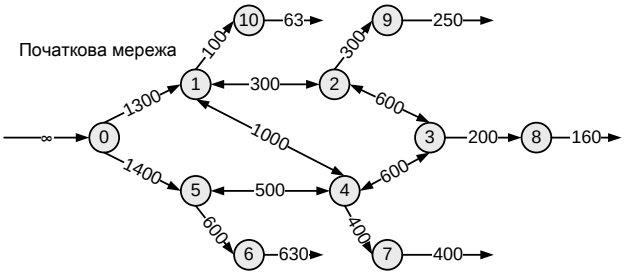

Рис.5. Тестова мережа

0	[0.0]	[1300.0]	[0.0]	[0.0]	[0.0]	[1400.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
1	[0.0]	[0.0]	[300.0]	[0.0]	[1000.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[100.0]	[0.0]
2	[0.0]	[300.0]	[0.0]	[600.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[300.0]	[0.0]	[0.0]
3	[0.0]	[0.0]	[600.0]	[0.0]	[600.0]	[0.0]	[0.0]	[0.0]	[200.0]	[0.0]	[0.0]	[0.0]
4	[0.0]	[1000.0]	[0.0]	[600.0]	[0.0]	[500.0]	[0.0]	[400.0]	[0.0]	[0.0]	[0.0]	[0.0]
5	[0.0]	[0.0]	[0.0]	[0.0]	[500.0]	[0.0]	[600.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
6	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[600.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[630.0]
7	[0.0]	[0.0]	[0.0]	[0.0]	[400.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[400.0]
8	[0.0]	[0.0]	[0.0]	[200.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[160.0]
9	[0.0]	[0.0]	[300.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[250.0]
10	[0.0]	[100.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	

Рис. 6. Матриця пропускних здатностей тестової мережі

Джерелом мережі є вузол з номером *(0)*, стоком мережі є вузол *(11)*. Сумарна пропускна здатність перерізу біля джерела, ребра *(0,1), (0,5) = 2700*, а сумарна пропускна здатність перерізу біля стоку, ребра *(6, 11), (7, 11), (8, 11), (9, 11), (10, 11)* = 1503, тестова мережа зображена на рисунку 5, стік мережі не показано, ним з'єднані вузли *6, 7, 8, 9, 10.*

Перша ітерація алгоритму.

Покладемо остаточні пропускні здатності (c_{ij}, c_{ji}) начальним пропускним здатностям (C_{ij}, C_{ji}).

Крок 1. Назначаємо $a_0 = \infty$ та помічаємо вузол *(0)* міткою $[\infty,-]$, дорівнюємо $i = 0$.

Крок 2. $S_1=[1,5]$ (Множина не пуста).

Крок 3. *k=5*, оскільки $C_{05}=max\{c_{01}, c_{05}\}=max\{1300, 1400\} = 1400$. Задаємо $a_5=c_{05}=1400$ і маркуємо вузол *(5)* міткою *[1400,0]* і повертаємось до кроку 2.

Крок 2. $S_2=[6,4]$ (Множина не пуста).

Крок 3. *k=6*, оскільки $C_{56}=max\{c_{54}, c_{56}\}=max\{500, 600\} = 600$. Задаємо $a_6 = c_{56} = 600$ і маркуємо вузол *(6)* міткою *[600,5]* і повертаємось до кроку 2.

Крок 2. $S_3=[11]$ (Множина не пуста).

Крок 3. *k=11*, оскільки $C_{6,11}=max\{c_6, c_{11}\}=max\{630\} = 630$. Задаємо $a_{11}=c_{6,11}=630$ і маркуємо вузол *(11)* міткою *[630,6]*. Отримали наскрізний путь. Далі до кроку 5.

Крок 5. Наскрізний путь визначаємо за мітками, починаючи з вузла 11 і кінець буде вузол *(0)*: *(11)→[630,6] →(6)→[600,5] →(5)→[1400,0] →(0).* Таким чином $N_1=\{0,5,6,11\}$ і $f_1=min\{a_0, a_5, a_6, a_{11}\} = min\{\infty, 1400, 600, 630\}=600$. Обчислюємо остаточні пропускні здатності вздовж шляху N_1:

$(C_{05},C_{50}) = (1400-600,0+600)=(800,600)$

$(C_{56},C_{65}) = (630-600,0+600)=(0,600)$

$(C_{6,11},C_{11,6}) = (600-600,0+600)=(30,600)$

Вид мережі після першої ітерації показано на рис. 7.

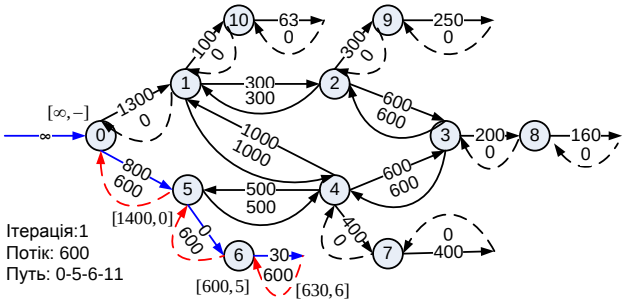

Рис.7. Мережа після першої ітерації алгоритму

Потоки які йдуть у напрямку протилежному орієнтації дуги на рисунках зображені пунктирними лініями.

2-га ітерація алгоритму.

Крок 1. Назначаємо $a_0 = \infty$ та помічаємо вузол *(0)* міткою $[\infty, -]$, дорівнюємо *i=0.*

Крок 2. $S_1=[1,5]$ (Множина не пуста).

Крок 3. *k=1*, оскільки $C_{01}=max\{c_{01}, c_{05}\}=max\{1300, 1800\} = 1300.$ Задаємо $a_1=c_{01}=1300$ і маркуємо вузол *(1)* міткою *[1300, 0]* і повертаємось до кроку 2.

Крок 2. $S_2=[2,4,10]$ (Множина не пуста).

Крок 3. *k=4*, оскільки $C_{1,4}=max\{c_{1,2}, c_{1,4}, c_{1,10}\}=max\{300, 1000, 1000\} = 1000$.. Задаємо $a_4 = c_{1,4} = 1000$ і маркуємо вузол *(4)* міткою *[1000, 1]* і повертаємось до кроку 2.

Крок 2. $S_3=[3,5,7]$ (Множина не пуста).

Крок 3. *k=3*, оскільки $C_{4,3}=max\{c_{4,3}, c_{4,5}, c_{4,7}\}=max\{600, 500, 400\} = 600.$

Задаємо $a_3=c_{4,3}=600$ і маркуємо вузол *(3)* міткою *[600,4]* і повертаємось до кроку 2.

Крок 2. $S_4=[2, 8]$ (Множина не пуста).

Крок 3. k=2, оскільки $C_{3,2}=max\{c_{3,2}, c_{3,8}\}=max\{600, 200\} = 600$. Задаємо $a_2=c_{3,2}=600$ і маркуємо вузол *(2)* міткою *[600,3]* і повертаємось до кроку 2.

Крок 2. $S_5=[9]$ (Множина не пуста).

Крок 3. *k=9*, оскільки $C_{2,9}=max\{c_{2,9}\}=max\{300\} = 300.$ Задаємо $a_9=c_{2,9}=300$ і маркуємо вузол *(9)* міткою *[300,2]* і повертаємось до кроку 2.

Крок 2. $S_6=[11]$ (Множина не пуста).

Крок 3. *k=11*, оскільки $C_{9,11}=max\{c_{9,11}\}=max\{250\} = 250$.. Задаємо $a_{11}=c_{9,11}=250$ і маркуємо вузол *(11)* міткою *[250, 9]*. Отримали наскрізний путь. Переходимо до кроку 5.

Крок 5. Наскрізний путь визначаємо за мітками, починаючи з вузла *(11)* і кінець буде вузол *(0)*: *0*: *(11)→[250,9] →(9)→[300,2] →(2) →[600,3] →(3)→[600,4] → (4) → [1000,1] →(1)→[1300,0]→(0).* Таким чином $N_2=\{0,1,4,3,2,9,11\}$ і $f_2=min\{a_0, a_1, a_4, a_3, a_2, a_9, a_{11}\} = min\{\infty, 1300, 1000, 600, 600, 300\ 250\}=600$. Обчислюємо остаточні пропускні здатності вздовж путі N_2:

$(C_{0,1},C_{1,0}) = (1300-250,0+250)=(1050,250)$

$(C_{1,4},C_{4,1}) = (1000-250,1000+2500)=(750,1250)$

$(C_{4,3}C_{3,4}) = (600-250,600+250)=(350,850)$

$(C_{3,2}C_{2,3}) = (600-250,600+250)=(350,850)$

$(C_{2,9}C_{9,2}) = (300-250,0+2500)=(50,250)$

$(C_{9,11}C_{11,9}) = (250-250,0+250)=(0,250)$

Вид мережі після другої ітерації показано на рис.8.

3 ітерація алгоритму: Отримано шлях $N_3=\{0,1,4,7,11\}$з $f_3=400$.

4 ітерація алгоритму: Отримано шлях $N_4=\{0,5,4,1,2,3,8,11\}$з $f_4=160$.

5 ітерація алгоритму: Отримано шлях $N_5=\{0,1,10,11\}$ з $f_5=63$.

6 ітерація алгоритму: Нові наскрізні путі неможливі, оскільки всі ребра які входять в вузол *(11)*, мають нульові остаточні пропускні здатності, окрім ребра $C_{6,11} = 30$, але в вузол *(6)* неможливо перейти, тому що єдине ребро по котрому можна це здійснити має нульову остаточну пропускну здатність $C_{5,6}=0$. Переходимо до кроку 6.

Крок 6. Максимальний обсяг потоку в мережі дорівнює: $F=f_1+f_2+...+f_5=600+250+400+160+63=1473$.

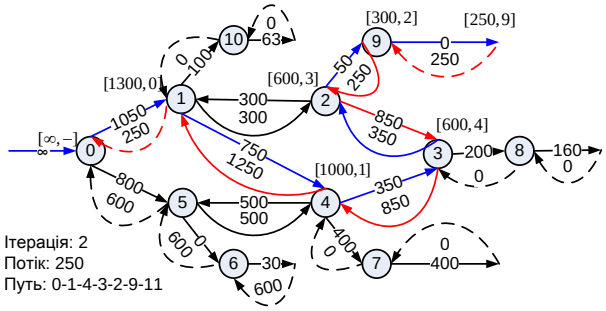

Рис.8. Мережа після другої ітерації алгоритму

Напрями та обсяги потоків обчислюються шляхом віднімання остаточних пропускних здатностей з начальних пропускних здатностей. Вид мережі наприкінці роботи алгоритму показані на рис.9.

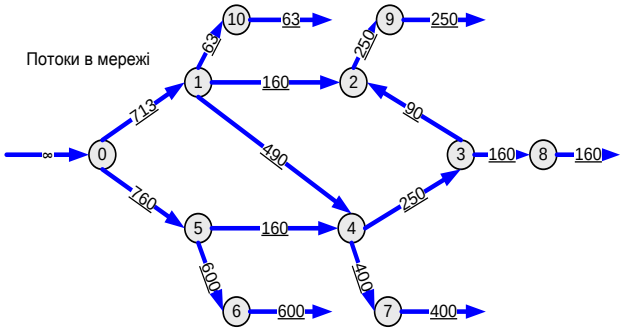

Рис.9. Обсяги та напрями потоків в мережі

Отже потік по ребру *(0,1) = 713, (0,5) = 760, (1,2) = 160, (1,4)= 490, (4,5) =160, (5, 6) =600, (6,7) =600, (3,4)=250, (2,3)=90, (1,10)=63, (10,9)=63, (2,9)=250, (9,11)=250, (3,8)=160, (8,11)=160, (4,7)=400, (7,11)=400.*

IV. Програмна реалізація алгоритму

Структура проекту - пакет org містить усі пакети програми: org.algo – містить у собі алгоритм знаходження максимального потоку в мережі, org.elements – містить у собі класи які відповідають за графічне відображення елементів схеми, а також містять дані про те приймають учать елементи в розподілі потоку або в його споживанні, org.graph – містить класи які представляють собою граф мережі і надають методи для роботи з нею, - додавання, видалення – ребер чи вузлів, org.gui – містить клас головного вікна програми, org.utils - містить клас, що відстежує динамічну конфігурацію мережі та надає дані: для роботи алгоритму, для виводу результатів роботи для головного класу (рис.10).

Алгоритм знаходження максимального потоку в мережі виділено в окремий пакет org.algo, назва класу PushRelabel (рис.11).

При створенні об’єкту класу він потребує для ініціалізації матрицю пропускних здатностей мережі, цю матрицю повертає метод класу VoltageTracker.getNetwork():

- PushRelabel pr = new PushRelable(voltageTracker.getNetwork());

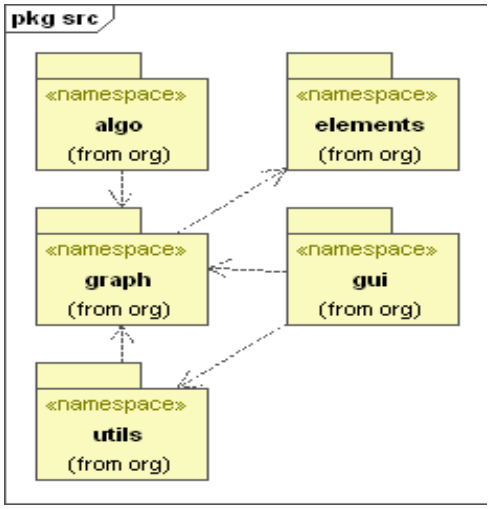

Рис.10. UML діаграма проекту

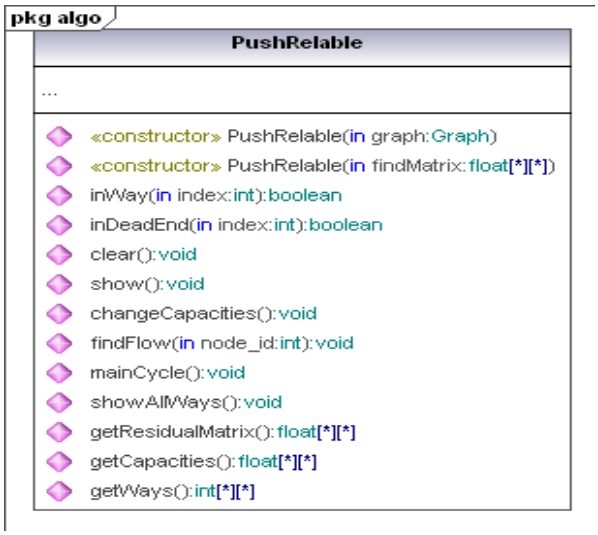

Рис.11. Клас PushRelabel

Методи класу PushRelable:

- public boolean inWay(int index) – повертає булеве значення, true якщо вузол вже знаходиться в списку вузлів задіяних в наскрізному шляху, якщо вузол не знайдено в цьому списку – метод повертає значення false;
- public boolean inDeadEnd(int index) – повертає булеве значення, true якщо вузол з індексом index, знаходиться в списку тупикових вузлів, якщо вузол не значиться в цьому списку метод повертає false;
- public void clear() – метод очищує списки: тупикових вузлів, вузлів наскрізного потоку. Та записує знайдений маршрут в матрицю знайдених шляхів. Викликається після кожної ітерації алгоритму;

- public void findFlow(int node_id) – метод шукає ребро по котрому можна перейти далі по мережі. Викликається рекурсивно до тих пір доки не досягне стоку мережі, або не буде ребер по яким можна перейти далі;
- public void changeCapacities() – метод викликається після завершення роботи методу findFlow(int node_id), змінює остаточні пропускні здатності мережі;
- public float[][] getResidualMatrix() – метод повертає матрицю остаточних пропускних здатностей мережі, для подальшої роботи програми;
- public float[][] getCapacities() - метод повертає матрицю начальних пропускних здатностей мережі, для подальшої роботи програми;
- public int[][] getWays() - метод повертає матрицю індексів вузлів котрі задіяні в знайдених наскрізних путях, для подальшої роботи програми;
- public void mainCycle() – головний цикл роботи алгоритму.

Пакет org.utils. При розробці програмного забезпечення було враховано, те що підстанція може виступати не тільки як споживач електроенергії а й як транспортуюча ланка мережі. Саме для цього передбачена можливість трансформування одного виду вузла (Трансформатор - Transformer) в інший (Перемикач - Switch). Коли вузол виглядає як перемикач він не вносить в обсяги циркулюючих потоків ніяких змін. Він тільки може або дозволити проходження потоку, або заборонити його проходження. Якщо вузол виглядає як трансформатор він автоматично з'єднується з вузлом стоку, тому що трансформатор є споживачем електричної енергії. За цими змінами слідкує клас під назвою VoltageTracker діаграма якого зображена на рис. 12.

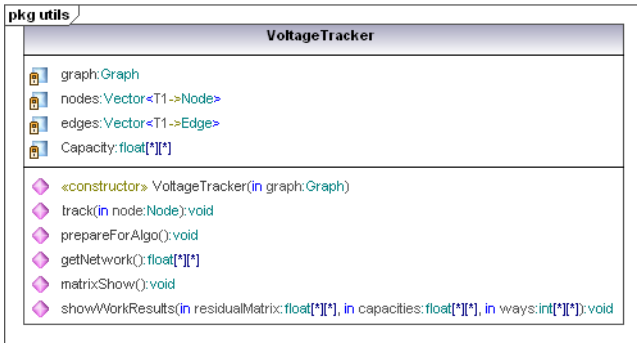

Рис.12. Клас VoltageTracker

Його метод track() починає діяти в тому випадку, якщо сталися зміни в конфігурації мережі. Також цей метод стежить за тим, які перемикачі на даний момент ввімкнені, а які вимкнуті. При цьому ввімкнені перемикачі, трансформатори та ділянки мережі які знаходяться під напругою позначаються червоним кольором, а ділянки без напруги позначуються зеленим кольором. Трансформатори та вимикачі при знятої напрузі відображаються сірим кольором.

Під час створення об'єкту класу потребуються для ініціалізації дані, які містить в собі клас Graph, це інформація про кількість вузлів мережі та яким чином вони з'єднані.

Метод prepareForAlgo() класу VoltageTracker слідкує за станом елементів мережі який змінюється методом track().

public float[][]getNetwork() – повертає матрицю початкових пропускних здатностей мережі, для роботи алгоритму PushRelabel, яка сформована в даний момент часу.

public void showWorkResults(float[][]residualMatrix, float[][]capacities) – Відображає результати роботи алгоритму PushRelabel.

Пакет org.graph – містить в собі три класи: Node, Edge, Graph, які забезпечують роботу з графом, - додавання, видалення вершин та ребер, повертання суміжних вершин для обраної, повертання кількості вершин та ребер в графі, зміна напряму дуги, зміни асоційованих вершин для дуг, повертання степеню і напівстепенів входу та виходу для вершин, та зберігання графу в пам'яті комп'ютера (див рис. 13).

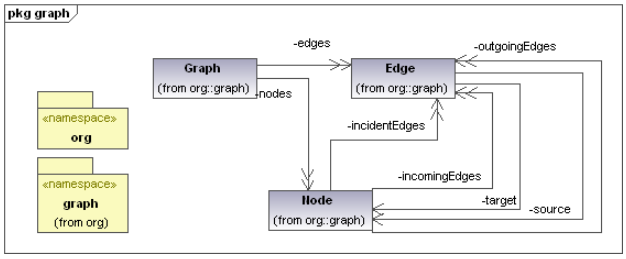

Рис.13. UML діаграма пакету org.graph

Клас Node, UML діаграма якого зображена на рис.14, представляє описання одиниці нижнього рівня графа – вершини. Для кожного вузлу графа створюється об'єкт класу. В ньому зберігаються посилання на всі інцидентні вузли, у вигляді зв'язних списків, що дозволяє швидко знаходити потрібні вузли.

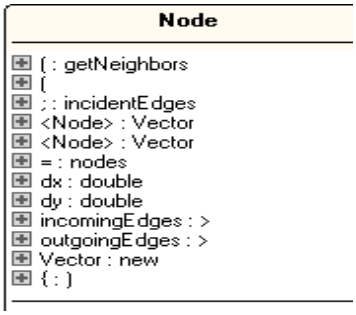

Рис.14. Клас Node

private Vector<Edge> incidentEdges – зберігає посилання на усі ребра які з'єднані з вершиною.

private Map<Node, Edge> incomingEdges – зберігає посилання на пари Вузол-Ребро для всіх вузлів та ребер які входять у дану вершину.

private Map<Node, Edge> outgoingEdges - зберігає посилання на пари Вузол-Ребро для всіх вузлів та ребер які виходять з даної вершини.

Клас має такі методи:

- public int getId() – повертає ціле число яке є індексом вершини, для джерела мережі індекс завжди дорівнює нулю id = 0;
- public String getLabel() – повертає строку з назвою яку ми асоціювали з вершиною, за замовчуванням назва є індексом вершини;
- public float getValue() – повертає значення вершини, застосовується для потужності трансформаторів;
- public void setVaule(float value) – встановлює нове значення потужності, за замовчуванням дорівнює 1.0f;
- public Vector<Node>getNeighbors() – повертає список вершин які з'єднані з даною;
- public int degree() – повертає ціле число яке вказує на степінь вершини;
- public int outDegree() – повертає ціле число яке вказує на полустепінь виходу;
- public int inDegree() – повертає ціле число яке вказує на полустепінь заходу;
- public boolean areAdjacent(Node node) – повертає булеве значення, true якщо вершина поєднана з вершиною яка передається в параметрах методу, в протилежному випадку повертає false;
- public Vector<Node> getOutgoingNodes() – повертає список всіх вершин в які можна перейти з даної вершини;
- public Vector<Node> getIncomingNodes() – повертає список вершин з яким можна перейти в дану вершину;
- public Vector<Edge> getIncidentEdges() – повертає список ребер які поєднані з даною вершиною;
- public Map<Node, Edge> getOutgoingEdges() – повертає всі пари Вершина-Ребро для ребер які виходять з даної вершини;
- public Map<Node, Edge> getIncomingEdges()– повертає всі пари Вершина-Ребро для ребер які входять в дану вершину;
- public void addIncidentEdge(Edge edge) – метод дозволяє додати інцидентне ребро;
- public void removeIncidentEdge(Edge edge) - метод дозволяє видалити інцидентне ребро;
- public void removeAdjacentNode(Node node) – метод видаляє суміжний вузол зі списків в яких він є.

Клас Edge, UML діаграма (рис.15), представляє абстракцію ребра графа, зберігає в собі посилання на вершини які поєднані даним ребром Node target, Node source. Має поле для зберігання пропускної здатності ребра. Для кожного ребра створюється об'єкт класу.

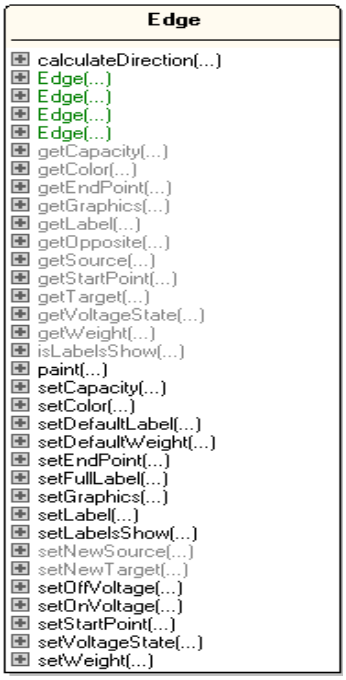

Рис.15. Клас Edge

Методи класу:

- public Node getSource() – повертає вузол з якого ребро виходить;
- public Node getTarget() – повертає вузол в який ребро входить;
- public Node getOpposite(Node node) – повертає протилежний, вузлу який подається в параметрах методу;
- public String getLabel() – повертає назву ребра, за замовчуванням назва виглядає як індекс начального вузла, індекс кінцевого вузла та пропускна здатність ребра;
- public void setLabel(String label) – встановлює назву для ребра;
- public void setDefaultLabel() – встановлює назву для ребра за замовчуванням;
- public float getCapacity() – повертає пропускну здатність ребра;
- public void setCapacity(float capacity) – встановлює пропускну здатність ребра;
- public boolean setNewTarget(Node target) – асоціює ребро з новим вузлом;
- public boolean setNewSource(Node source) – асоціює ребро з новим вузлом.

Клас Graph – представляє абстракцію моделі графа (рис.16).

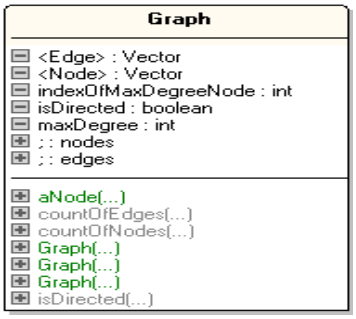

Рис.16. Клас Graph

Зберігає в собі посилання на всі вершини та ребра які входять до графу. Дозволяє роботу з графом: додавання, видалення вершин та ребер.

Розглянемо методи класу. public int countOfNodes() – повертає ціле число яке вказує на кількість вузлів. public int countOfEdges() – повертає ціле число яке вказує на кількість ребер. public Node insertNode(String label, float value) – добавляє новий вузол в граф. public void removeNode(Node node) – видаляє вузол з графу. public Node getNode(int index) – повертає вузол за індексом. public Vector<Node> getAllNodes() – повертає список всіх вузлів. public Edge addEdge(Edge edge) – додає нове ребро до графу. public void removeEdge(Edge edge) – видаляє ребро з графу. public Edge getEdge(int index) – повертає ребро за індексом під яким воно зберігається в списку усіх ребер графа. public Vector<Edge> getAllEdges() – повертає список всіх ребер графа.

Пакет org.elements. Текучий стан мережі відображається на мнемосхемі, на якій спеціальними символами зображено комутаційну апаратуру та стан в якому вона знаходиться. Чи перебуває дана ланка мережі під напругою чи вона відключена або навпаки ввімкнена але на неї відсутня напруга, це має відображатись на мнемосхемі.

Лінії електропередач які знаходяться під напругою повинні відображатися червоним кольором який вказує «НЕБЕЗПЕЧНО», лінії на яких немає напруги відображаються зеленим кольором. Всі трансформатори які перебувають під напругою відображаються чорним кольором що говорить про те що вони активні. Трансформатори які відключені від мережі зображені блідо сірим кольором – неактивні. Перемикачі ввімкнені і якщо на вході чи навпаки в зворотному напрямку до нього підходить напруга відображені червоним кольором. Якщо перемикач вимкнено але на вході до нього є напруга він зображується зеленим кольором. Якщо лінія знаходиться без напруги то всі перемикачі на неї незалежно від того ввімкнені чи вимкнені зображуються сірим кольором.

Пакет org.elements, в якій включені всі класи, що відповідають за відображення елементів мовою паттернів.

Клас SwitchElement – є абстракцією всіх видів комутаційної апаратури має декілька станів відображення елемента:

- ввімкнений – зображується як квадрат чорного кольору з червоною окантовкою, всередині містить вертикально орієнтований червоний прямокутник;
- вимкнений – зображується як квадрат чорного кольору з зеленою окантовкою, всередині містить горизонтально розташований зелений прямокутник;
- під напругою – графічно зображується залежності від стану (ввімкнений, вимкнений) колір зберігається;
- без напруги – графічне зображення стану (ввімкнено, вимкнено) зберігається, але колір окантовки і прямокутника всередині замінюється на сірий.

Клас TransformerElement – є абстракцією всіх видів трансформаторів та підстанцій має два стани відображення активний (чорний колір), не активний (сірий колір). Графічне позначення - це два кільця які пересікаються в вигляді вісімки. Первинна обмотка трансформатора це верхнє кільце, а нижнє кільце представляє вторинну обмотку трансформатора.

Програма дозволяє:

- створювати новий граф мережі;
- завантажувати існуючі графи мережі;
- задавати пропускну здатність ребра;
- змінювати тип вузла, якщо на підстанції вимкнено трансформатор, але вона приймає участь в перерозподілі потоків потужностей, тип вузла необхідно змінити на перемикач. У випадку коли трансформатор знов ввімкнули і він є споживачем потоку потужності, потрібно знов змінювати тип вузла з перемикача на трансформатор.

На рис. 17 зображена тестова мережа

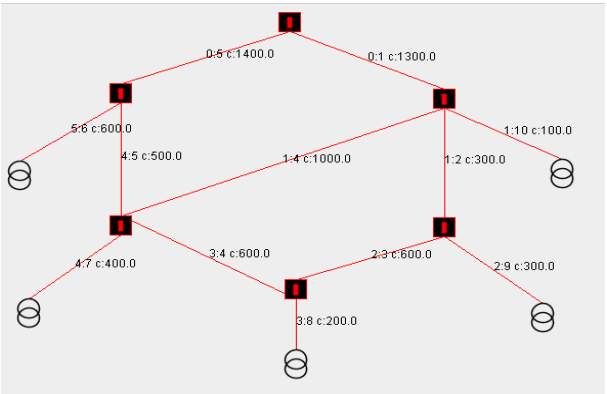

Рис.17. Тестова мережа

Біля кожного ребра вказана його пропускна здатність та номери вузлів які воно поєднує. Після виконання алгоритму ці назви зміняться на дрібні значення в чисельнику буде пропускна здатність ребра а в знаменнику потік по цьому ребру.

Для запуску алгоритму розрахунку пропускної здатності та закінчення його роботи, вид назв ребер зміниться на розрахований потік (рис. 18).

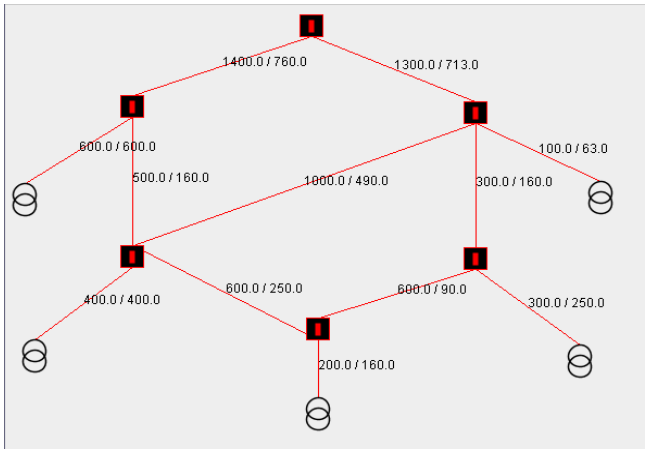

Рис.18. Мережа з розрахованими потоками

Мережа прийме начальний вигляд, якщо є необхідність повернути назви ребер до начального виду.

При роботі реальної електричної мережі її конфігурація змінюється в часі, здійснюються перемикання для ремонтних потреб, або для передачі частини потужності з одного на інший фідер. Для того щоб змінювати конфігурацію мережі передбачена можливість перемикання комутаційних апаратів на схемі мережі (рис. 19).

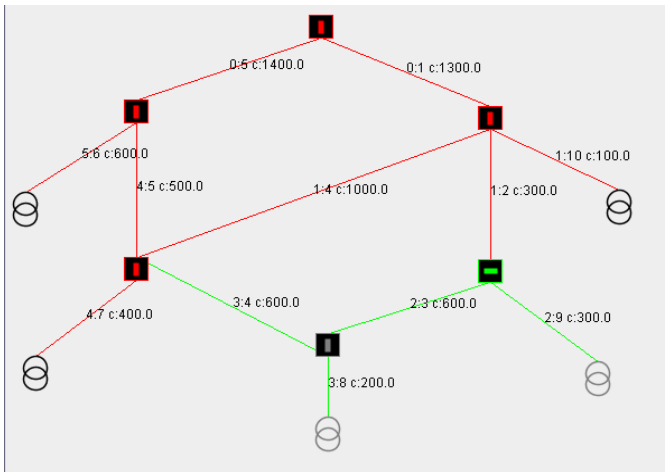

Рис.19. Вигляд мережі після перемикання

При здійсненні перемикань мережа змінює свій вигляд, що дозволяє швидко орієнтуватися в тому – який апарат відключено, які трансформатори і комутаційні апарати знаходяться під напругою, а які знеструмлені. На схемі це відображається за допомогою кольору – всі елементи схеми і лінії які знаходяться під напругою відображаються червоним кольором, всі елементи схеми які знаходяться без напруги відображаються сірим кольором, для перемикачів не важливо в якому він стані (ввімкнений чи вимкнений). Якщо перемикач вимкнений і до нього з одного боку подається напруга він графічно зображується як вимкнений, але його колір зелений а не сірий, це говорить про те що він під напругою.

Після проведення кожного перемикання програма автоматично розраховує конфігурацію мережі і вносить необхідні зміни в граф мережі. Але після цього необхідно знов розраховувати потоки в мережі.

V. Висновки

В роботі був реалізований алгоритм знаходження максимального потоку на зваженому графі для управління комунікаційними режимами електричних мереж в оперативно-диспетчерському керуванні для системи «Smart City».

Головною метою реалізації даної системи є попередження аварійних ситуацій, що підвищує безпеку і забезпечує підвищення надійності електропостачання споживачів електроенергією.

Головними особливостями розробленої системи є :

- можливість внесення динамічних змін в електричну мережу і швидкий розрахунок наслідків таких змін;
- можливість внесення статичних змін в мережу;
- можливість трансформування типів елементів мережі;
- можливість швидкого розрахунку потоків в умовах динамічних змін конфігурації мережі.

На основі проведеного аналізу та розробленої системи була обґрунтована доцільність застосування моделі теорії графів для подання електричної мережі в умовах постійної динамічності останньої, а також застосування алгоритмів знаходження максимального потоку в мережі для аналізу управління комутаційними режимами в електричних мережах.

Література References

[1] A Smart Approach to Planning Smart Cities. 2018 [Online]. https://sites.atis.org/insights/a-smart-approach-to-smart-cities/Nation.

[2] ГОСТ 24291-90. Электрическая часть электростанции и электрической сети. Термины и определения [Online]. Available: http://docs.cntd.ru/document/gost-24291-90

[3] Алгоритмы нахождения максимального потока 2018 [Online]. Available: http://algolist.ru/maths/graphs/maxflows/