Dokumentacja projektu

‘ Znajdowanie optymalnego miejsca parkingowego w oparciu w ważony MaxSat Solver’

Studio Projektowe I

Twórcy : Bartosz Biegun , Paweł Hanzlik

Opiekun: prof. Radosław Klimek

12.06.2021

Spis treści

1. Cel projektu 3
2. Zasada działania 3
   a. Strefy 3
   b. Solver 5
3. Architektura systemu 6
4. Schemat bazy danych 7
5. Przykłady rekordów w tabelach 8
6. Znaczenie poszczególnych pól w tabelach 9
   a. Tabela Strefy: 9
   b. Tabela Kierowcy: 9
   c. Tabela Parkingi: 9
7. Diagram klas 10
8. Wykorzystane technologie 11
9. Generowanie klauzul z dostępnych danych 11
   a. Określenie zmiennych zdaniowych 11
   b. Określenie klauzul 12
   c. Interpretacja wyników 15
10. Biblioteka SAT4J 15
    a. Format danych wejściowych 15
11. Jak działa nasz solver 16

1. Cel projektu

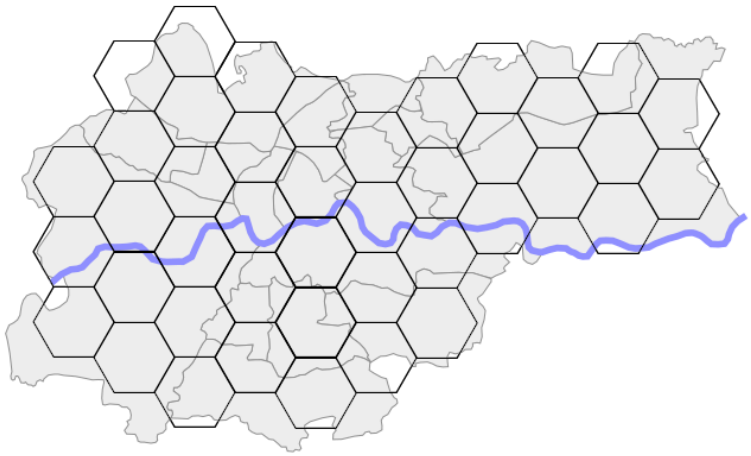
Celem projektu jest zaimplementowanie aplikacji wykorzystującej solver typu MaxSat do znajdowania miejsc parkingowych dla klientów wypożyczalni samochodów z uwzględnieniem popytu na następne wypożyczenia w danym sektorze miasta.

Aplikacja będzie symulowała obszar danego miasta podzielonego na wyznaczone strefy, wewnątrz których jest pewna liczba samochodów dostępnych do wypożyczenia. Każda strefa będzie mieć przewidywany popyt na samochody, na który aplikacja będzie odpowiadać przekierowując samochody do danej strefy gdy są tam potrzebne. Użytkownik będzie posiadał możliwość utworzenia zapytania odpytującego serwer o miejsce parkingowe w pobliżu pewnej lokacji lub uruchomić prostą symulację generującą wiele podobnych zapytań oraz modyfikującą stan bazy w zależności od pory dnia.

2. Zasada działania

a. Strefy

Miasto zostanie podzielone na heksagonalne strefy w celu zarządzania popytem i podażą samochodów na zróżnicowanym obszarze.



W celu optymalnego rozlokowania klientów dla każdej strefy jest regularnie obliczany współczynnik zajętości:

$$a_n = \frac{S_n}{D_n}$$

gdzie:

a_n- Współczynnik zajętości

S_n- Suma wag klientów chcących wypożyczyć samochód w odległości do 1 km od środka strefy

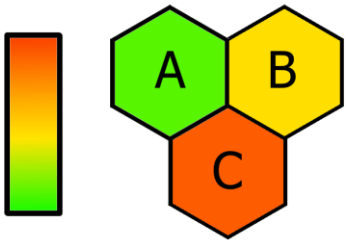
D_n- Suma wag wolnych samochodów w odległości do 1 km od środka strefy

Wagi klientów i miejsc parkingowych maleją wraz z odległością od środka strefy co odwzorowuje możliwość że klienci na skraju strefy zostaną przypisani do miejsc parkingowych w sąsiadującej strefie

Aplikacja będzie manipulować klientami chcącymi zaparkować w pobliżu strefy a przez to sumą wag wolnych samochodów aby utrzymać współczynnik zajętości zbliżony do 1.

Ważnym czynnikiem będzie również wskaźnik atrakcyjności strefy, który także wpłynie na rezultat pracy solvera.

Na przykład:



Samochody ze strefy A (O niskim współczynniku zajętości – samochodów jest za dużo) będą przekierowywane do strefy C (O wysokim współczynniku zajętości – samochodów brakuje)

b. Solver

Problem spełnialności to koncept związany z logiką matematyczną, zostanie on wykorzystany do rozwiązania problemu zarządzania wypożyczanymi samochodami w Krakowie.

Z problemem SAT mamy do czynienia gdy mając formułę zdaniową chce się określić, czy istnieje podstawienie wartościami ‘0’ i ‘1’ pod zmienne zdaniowe, by formuła była spełniona.

Problemy Max-Sat składają się z ważonych klauzul połączonych koniunkcjami :

$(\neg p1 \vee \neg p2) \wedge (\neg p1 \vee p3) \wedge (\neg p1 \vee \neg p3)$

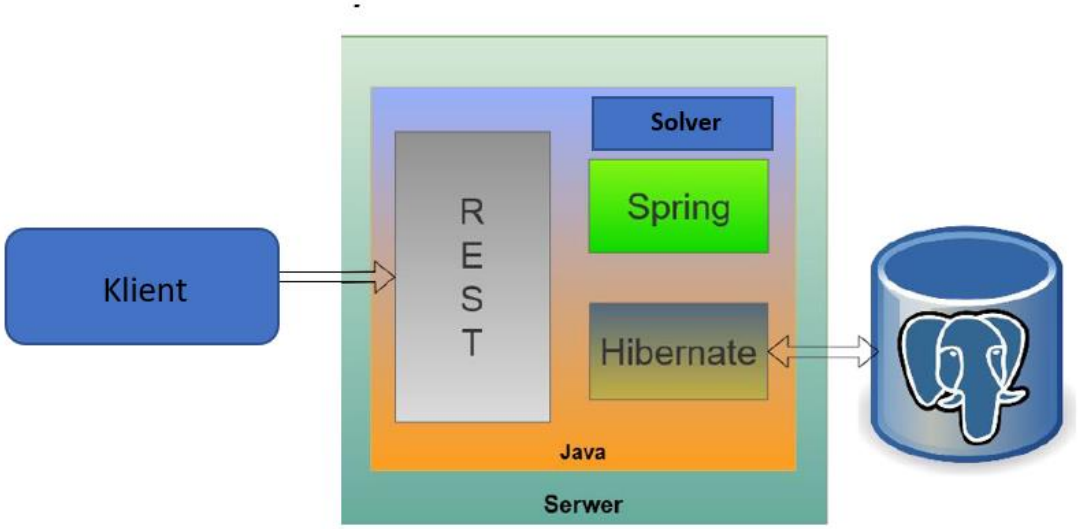
Ponieważ nie zawsze da się spełnić wszystkie klauzule, solver Max Sat znajduje rozwiązanie w którym największa liczba klauzul jest spełniona

Z problemem Max-SAT to rozszerzenie problemu SAT w taki sposób, aby w przypadku gdy nie da się dobrać wartości tak, aby spełniona była cała formuła dobiera się je tak, by zmaksymalizować ilość spełnionych formuł.

Ważony Max-SAT to kolejne rozszerzenie, dodające tym razem odpowiadające wagi każdej z klauzul i uwzględnienie ich a procesie rozwiązywania w taki sposób aby suma wag niespełnionych klauzul była jak najmniejsza.

Aby dobrać najlepsze dla systemu rozwiązanie będzie on generować formułę w której zdania będą odpowiadać optymalnemu rozlokowaniu samochodów.

3. Architektura systemu



Główny moduł aplikacji zostanie napisany w języku Java z użyciem Spring Framework. Baza danych działać będzie na serwerze PostreSQL, natomiast łączenie jej z projektem realizowane będzie przy użyciu Hibernate. Z zewnątrz klient wysyłać będzie zapytanie obsługiwane przez REST API, które na podstawie danych z bazy oraz obliczeń Solvera zwróci wynik.

4. Schemat bazy danych

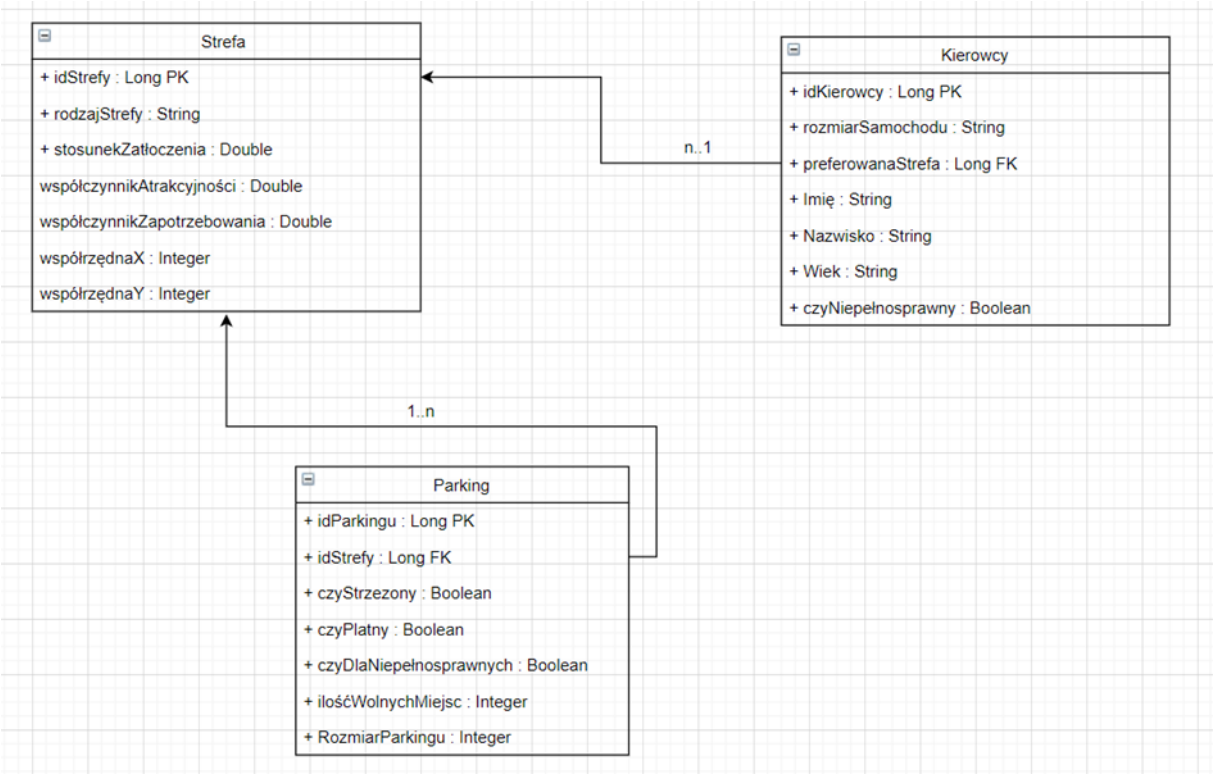


Tabela Strefa oznacza podział miasta na obszary posiadające wartość zatłoczenia w danym momencie, współczynnik atrakcyjności, współczynnik zapotrzebowania oraz rodzaj Strefy tj. obrzeża, centrum, poza miastem, przemysłowa, a także jej współrzędne. Zawiera wiele rekordów typu Parking.

Tabela Kierowcy zawiera dane klienta, informacje o rozmiarach ich samochodów oraz preferencjach dotyczących miejsca parkingowego.

Tabela Parking zawiera informacje dotyczące parkingu w danej strefie miasta.

5. Przykłady rekordów w tabelach

Tabela: strefy

idStrefy	rodzajStrefy	WspółczynnikZatłoczenia	WspółczynnikAtrakcyjności	WspółczynnikZapotrzebowania	WspółrzędnaX	WspółrzędnaY
1	centrum	0.7	0.5	0.3	-4	-3
2	centrum	0.8	0.5	0.5	2	1
3	obrzeża	0.5	0.3	0.4	-1	2
4	obrzeża	0.6	0.7	0.7	1	0
5	poza miastem	0.3	0.2	0.4	0	-2
6	poza miastem	0.2	0.4	0.5	3	1

Tabela: kierowcy

idKierowcy	rozmiarSamochodu	preferowanaStrefa	Imię	Nazwisko	Wiek	czyNiepełnosprawny
1	mały	2	Anna	Nowak	35	nie
2	średni	4	Krzysztof	Kowalski	45	tak
3	mały	1	Michał	Kowalczyk	65	tak
4	duży	6	Jan	Nowakowski	24	nie
5	duży	3	Wojciech	Kwiatkowski	38	nie

Tabela: parkingi

idParkingu	idStrefy	czyStrzeżony	czyPłatny	czyDlaNiepełnosprawnych	ilośćWolnychMiejsc	RozmiarParkingu
1	4	tak	tak	tak	50	3
2	3	nie	tak	nie	45	4
3	1	nie	nie	tak	16	1
4	3	tak	nie	nie	47	2
5	6	tak	nie	tak	38	3

6. Znaczenie poszczególnych pól w tabelach

a. Tabela Strefy:

idStrefy – klucz podstawowy tabeli

rodzaj strefy – określenie położenia strefy na ‘mapie’. Będzie on mógł być wykorzystany do bardziej inteligentnego generowania wartości zapotrzebowania oraz zatłoczenia

współczynnik zatłoczenia – określa nam jak bardzo zatłoczona jest strefa, tj. ile wolnych, czekających na klienta samochodów jest w danej strefie. Będziemy starali się jak najbardziej dopasowywać te wskaźniki pomiędzy strefami, tak aby nie było w nich za dużo, ani za mało samochodów. Dlatego więc, gdy współczynnik zapotrzebowania będzie dużo większy niż współczynnik zatłoczenia będziemy starali się kierować tam klientów w celu parkowania nieużytkowanego już przez niego auta.

współczynnik atrakcyjności – określa jak bardzo przyjazna dla użytkownika jest to strefa. Wpływać na to będzie kilka czynników takich jak: ilość i rodzaj parkingów w obrębie tej strefy oraz poszczególne ich parametry.

współczynnik zapotrzebowania – określa jak dużo klientów w danej strefie szuka samochodu do wynajęcia. Będziemy starali się właśnie w te strefy z najwyższym współczynnikiem kierować osoby parkujące swoje auta.

współrzędnaX – pierwsza współrzędna położenia strefy na mapie miasta

współrzędnaY – druga współrzędna położenia strefy na mapie miasta

b. Tabela Kierowcy:

idKierowcy – klucz podstawowy tabeli

rozmiar samochodu – wielkość auta, którym porusza się użytkownik. Może on posłużyć do urozmaicenia formuły naszego solvera, tak aby kierował go na taki parking w obrębie strefy, który jest dostosowany do rozmiaru jego auta.

preferowana strefa – strefa, którą użytkownik uważa za najodpowiedniejszą dla niego, solver będzie starał się dopasować parkingi, tak aby były jak najbliżej jego strefy.

imię, nazwisko, wiek – dane użytkownika

niepełnosprawność – określa czy kierowca ma orzeczenie o stopniu niepełnosprawności, solver przydzieli mu najlepszy parking posiadający

c. Tabela Parkingi:

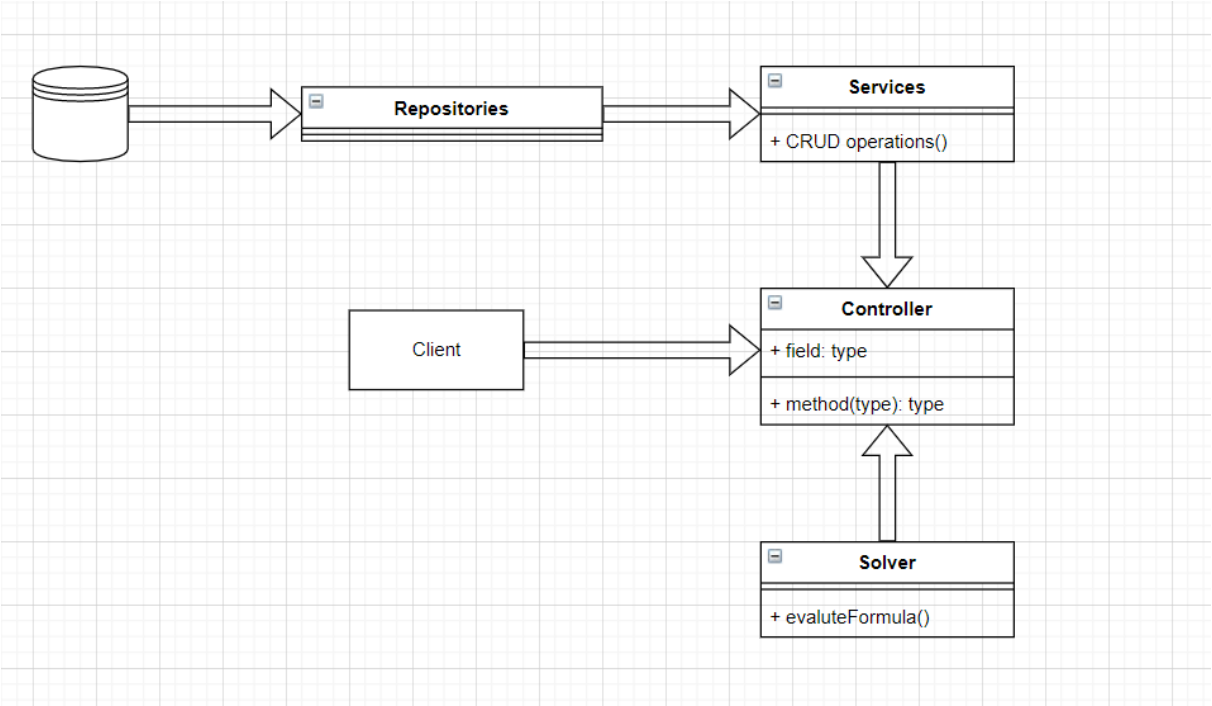
idParkingu – klucz podstawowy tabeli

idStrefy – strefa, w której znajduje się parking

strzeżony, płatny, dla niepełnosprawnych, rozmiar parkingu – parametry parkingu, które użytkownik będzie mógł wybrać w swoim zapytaniu

ilość wolnych miejsc – brane pod uwagę będą tylko te parkingi, które mają jeszcze jakieś wolne miejsca postojowe

7. Diagram klas



Sekcja Repositories to interfejsy dziedziczące po JPARepository, realizujące połączenie z bazą. Serwisy będą przechowywały metody umożliwiające pobieranie, dodawanie czy też modyfikowanie oraz usuwanie danych z bazy. Solver będzie to zbiór klas, które implementują algorytm Max-Sat solver i na podstawie zapytania klienta zwróci najlepsze miejsce parkingowe. Kontroler będzie przetwarzał zapytanie, przekazywał do solvera i zwracał odpowiedź klientowi.

8. Wykorzystane technologie

- **Java**

  Będziemy używać języka Java w wersji 15

- **Spring**

  Użyjemy SpringBoot'a w wersji 2.4.

- **PostgreSQL**

  Do zarządzania bazą danych. Wersja 13.

- **Hibernate**

  Czyli framework realizujący dostęp do danych w bazie. Podstawową rzeczą jaką będziemy wykorzystywać to mapowanie O-R.

- **Lombok**

  Biblioteka Javy udostępniająca adnotacje do tworzenia metod, głównie getterów i setterów.

9. Generowanie klauzul z dostępnych danych

Do przetwarzania danych za pomocą algorytmu Weighted Max-Sat wymagane jest przekształcenie je na zmienne zdaniowe a następnie przyporządkowanie do klauzul.
Każdej klauzuli przypisywana jest waga określająca jak ważne jest spełnienie klauzuli.
Każde dostępne miejsce parkingowe zostaje porównane z klauzulą i przedstawione użytkownikowi. Lista dostępnych miejsc zostaje posortowana po ilości spełnionych klauzul tak, aby najbardziej dopasowane miejsca znalazły się na górze listy.

a. Określenie zmiennych zdaniowych

Na podstawie bazy danych wyróżnimy 6 zmiennych zdaniowych:

- S1 – Parking znajduje się w strefie 1
- S2 – Parking znajduje się w strefie 2
- S3 – Parking znajduje się w strefie 3
- S4 – Parking znajduje się w strefie 4
- S5 – Parking znajduje się w strefie 5
- S6 – Parking znajduje się w strefie 6
- S7 – Parking znajduje się w strefie 7
- S8 – Parking znajduje się w preferowanej strefie klienta
- S9 – Parking ma więcej niż 10 wolnych miejsc
- S10 – Parking jest strzeżony
- S11 – Parking jest płatny
- S12 – Parking jest przystosowany dla osób niepełnosprawnych
- S13 – Rozmiar parkingu > 5

Rozpatrujemy tylko 7 stref znajdujących się dookoła pozycji użytkownika. Strefa nr 1 to ta, w której znajduje się użytkownik.

Przykład:

```
//  S1 = 0
//  S2 = 0
//  S3 = 0
//  S4 = 1
//  S5 = 0
//  S6 = 0
//  S7 = 0
//  S8 = 0
//  S9 = 1
//  S10 = 1
//  S11 = 0
//  S12 = 1
//  S13 = 0

//  [S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13] = [0,0,0,1,0,0,0,0,1,1,0,1,0]
```

Takie ułożenie zmiennych zdaniowych zawiera informację o tym, że parking znajduje się w strefie 4, która nie jest jednak preferowaną strefą użytkownika oraz ma więcej niż 10 wolnych miejsc, dodatkowo jest strzeżony, a także przystosowany dla osób niepełnosprawnych lecz nie jest płatny, ani nie jest to bardzo duży parking.

b. Określenie klauzul

Analizując dane o kliencie możemy oszacować jego preferencje i dopasować do nich zmienne zdaniowe. Wagi klauzul związanych z atrakcyjnością zostały odpowiednio pomniejszone, aby bardziej liczyło się zapotrzebowanie na samochody niż wygoda klienta.

- U0 – Przynajmniej jedna strefa, w której klient może zaparkować musi zostać wybrana
- U1 – Strefa nr 1 jest najbardziej pożądana, aby klient tam zaparkował
- U2 – Strefa nr 2 jest najbardziej pożądana, aby klient tam zaparkował
- U3 – Strefa nr 3 jest najbardziej pożądana, aby klient tam zaparkował
- U4 – Strefa nr 4 jest najbardziej pożądana, aby klient tam zaparkował
- U5 – Strefa nr 5 jest najbardziej pożądana, aby klient tam zaparkował
- U6 – Strefa nr 6 jest najbardziej pożądana, aby klient tam zaparkował
- U7 – Strefa nr 7 jest najbardziej pożądana, aby klient tam zaparkował
- U8 – Preferowana strefa klienta to ta sama strefa, którą wskazał solver
- U9 – Klient jest niepełnosprawny
- U10 – Rozmiar samochodu klienta > 5
- U11 – Klient jest skąpy
- U12 – Klient dba o wygodę parkowania

U0 -> U1 v U2 v U3 v U4 v U5 v U6 v U7 (Infinity)

U1 -> S1 (Waga obliczana dynamicznie)

U2 -> S2 (Waga obliczana dynamicznie)

U3 -> S3 (Waga obliczana dynamicznie)

U4 -> S4 (Waga obliczana dynamicznie)

U5 -> S5 (Waga obliczana dynamicznie)

U6 -> S6 (Waga obliczana dynamicznie)

U7 -> S7 (Waga obliczana dynamicznie)

U8 -> S8 (15)

Klauzula U0 jest tworem sztucznym, stworzonym tylko po to, żeby solver wybrał przynajmniej jedną strefę, dlatego jej waga to nieskończoność.

Waga obliczana dynamicznie oznacza:

Math.*round*((occupiedRatio-requestRatio)*100),

gdzie occupiedRatio to współczynnik zajętości, a requestRatio to współczynnik zapotrzebowania. Oznacza to, że im większe zapotrzebowanie na samochody w danej strefie, tym większą wagę otrzyma klauzula chcąca wybrać tą strefę jako tą, w której klient chce zaparkować.

W przyszłości uwzględnimy jeszcze współczynnik atrakcyjności.

Powyższe klauzule są niezależne od użytkownika, ponieważ to my narzucamy, by kierujący w pierwszej kolejności zostawił samochód w strefie, w której my chcemy, gdyż brakuje tam aut dla innych wypożyczających. Mimo wszystko delikatnie idziemy użytkownikowi na rękę i lekko faworyzujemy jego strefę.

U9  S12 ∧ S9 v S13 (25)

U10  S10 ∧ S13 (20)

U11  ¬S11 ∧ S9 (15)

U12  S10 v S9 ∧ S8 v ¬S11 (20)

¬U9  ¬S12 ∧ ¬S13 (15)

¬U10  ¬S10 ∧ ¬S13 v ¬S9 (10)

¬U11  S11 ∧ ¬S9 v S10 (25)

¬U12  ¬S9 ∧ ¬S8 (10)

U9 -> S12 ∧ (S9 v S13) – Użytkownik posiada orzeczenie o niepełnosprawności, będzie więc chciał parkować na parkingach dostosowanych do niego oraz takich, które są duże i mniej oblegane.

U10 -> S10 ∧ S13 – Użytkownik prowadzi duży samochód, będzie więc chciał parkować na dużym, strzeżonym parkingu.

U11 -> ¬S11 ∧ S9 – Użytkownik jest skąpy, wybierze on więc niepłatny, mało oblegany parking.

U12 -> (S10 v S9) ∧ (S8 v ¬S11) – Użytkownik chce parkować bezpiecznie i wygodnie, wybierze on więc strzeżony, obszerny parking, w dogodnej dla niego strefie, chyba, że parking nie będzie płatny.

¬U9 -> ¬S12 ∧ ¬S13 – Użytkownik nie posiada orzeczenia o niepełnosprawności, będzie więc chciał parkować na parkingach bez stref dla niepełnosprawnych oraz nie będzie musiał mieć zapewnionego dużego parkingu.

¬U10 -> ¬S10 ∧ (¬S13 v ¬S9) – Użytkownik nie prowadzi dużego samochód, nie będzie więc musiał parkować na dużym, strzeżonym parkingu, lub na mniej uczęszczanym.

¬U11 -> S11 ∧ (¬S9 v S10) – Użytkownik nie jest skąpy, nie musi on więc wybierać niepłatnego, mało oblegany parking, pod warunkiem, że jest on strzeżony.

¬U12 -> ¬S9 ∧ ¬S8 – Użytkownik nie musi parkować bezpiecznie i wygodnie, nie musi więc być to parking w strefie klienta lub być mało uczęszczany.

Przykład:

```
//  U9 = 1
//  U10 = 0
//  U11 = 0
//  U12 = 1
//  [U9,U10,U11,U12] = [1,0,0,1]
```

Podane parametry przez użytkownika. Oznaczają one, że kierujący jest niepełnosprawny oraz ceni sobie wygodę parkowanie, nie kieruje on natomiast dużym samochodem, ani nie jest skąpy.

Podane klauzule dołączą w solverze do już obecnych tworząc formułę:

```
/*  U0 => (S1 v S2 v S3 v S4 v S5 v S6 v S7)
    U9 => (S12 ∧ (S9 v S13))
    ¬U10 => (¬S10 ∧ (¬S13 v ¬S9))
    ¬U11 -> (S11 ∧ (¬S9 v S10))
    U12 -> ((S10 v S9) ∧ (S8  v ¬S11))
*/

/*  (S1 v S2 v S3 v S4 v S5 v S6 v S7) ∧ (S12 ∧ (S9 v S13)) ∧ (¬S10 ∧ (¬S13 v ¬S9))
    (S11 ∧ (¬S9 v S10)) ∧ ((S10 v S9) ∧ (S8  v ¬S11))
*/
```

c. Interpretacja wyników

Algorytm iteruje po wszystkich miejscach parkingowych i wyodrębnia te z największą ilością spełnionych klauzul, a następnie przedstawia je użytkownikowi jako listę. W tym momencie powinny to być głównie parkingi znajdujące się w strefie z najwyższą wagą, dopiero wtedy staramy się jak najlepiej dopasować parametry parkingu, tak aby spełniały one jak najwięcej klauzul U9-12

10. Biblioteka SAT4J

SAT4J to biblioteka open source stworzona dla języka Java w celu rozwiązywania problemów logicznych i optymalizacyjnych. Idealnie nadaję się do obliczania zadań związanych z SAT oraz jego wariancjami. Użyte w niej wzorce projektowe takie jak strategia oraz dekorator sprawiają, iż można dopasować solver do swoich potrzeb. Mimo wszystko jest ona jednak stosunkowo wolna w działaniu co jest jej znacznym minusem.

a. Format danych wejściowych

Biblioteka wymaga od użytkownika specjalnego wejścia.

```
c
c WCNF w formacie DIMACS
c
p wcnf 2 3
3 1 -2 0
7 1 2 0
4 -1 2 0
```

Gdzie w linii p wcnf 2 3 trzeci argument to nvars czyli liczba zmiennych zdaniowych, a czwarty to nclauses czyli liczba klauzul. Następnie następuje przekazanie właściwych danych tj. nclauses linii zawierających wagę danej klauzuli, po której występuje nvars liczb z zakresu [-nvars; nvars] przy czym dodatnie liczby odpowiadają danej formule zdaniowej, a ujemna jej zaprzeczeniu. Każda linia kończy się liczbą 0.

11. Jak działa nasz solver

W pierwszym będziemy wybierali użytkownika z bazy, dla którego obliczony zostanie najbardziej optymalny parking, na którym będzie on mógł zaparkować. Wywołamy więc dla niego też Solver podając listę parametrów tj. listę wartości klauzul. Będzie ona wyglądała np. w taki sposób:

[U9, U10, U11, U12] = [0,0,1,1]

Następnie zostaną wygenerowane dla niego koordynaty, na których się on znajduje. Wtedy także wyszukane zostaną strefy otaczające użytkownika, wraz z tą, w której się znajduje. Dla nich zostanie wygenerowanych i dodanych 7 klauzul do Solvera, po jednej dla każdej strefy. Ich wagi będą uzależnione od parametrów danej strefy, tj. większą wagę dostaną strefy z większym współczynnikiem zapotrzebowania oraz współczynnikiem atrakcyjności, a mniejszą te z mniejszym współczynnikiem zapotrzebowania i większym współczynnikiem zajętości, jako, że nie potrzeba tam w tej chwili więcej samochodów.

Do stworzonych już klauzul dodane zostaną te, które podał użytkownik.

To co będzie się działo, to tak naprawdę w pierwszej kolejności wybór strefy z najwyższą wagą, a później dopasowanie parkingu o parametrach spełniających klauzule o najwyższych wagach.