Dokumentacja projektu zespołowego

Szymon Kozakiewicz Joanna Świętosławska

1 Opis problemu

Sieć sensoryczna jest wykorzystywana do zbierania informacji z danego obszaru. Sensory są często rozstawiane losowo (na przykład poprzez zrzucanie ich z samolotu). Obszary z których sensory mogą zbierać informacje mogą się więc pokrywać. Z tego powodu nie wszystkie sensory muszą być jednocześnie włączone. Wykorzystując ten fakt można znacząco wydłużyć czas życia danej sieci poprzez odpowiednie włączanie i wyłączanie sensorów. Powinno się to robić tak by jak najbardziej ograniczyć zużycie energii przy jednoczesnym zachowaniu akceptowalnego poziomu pokrycia monitorowanego obszaru.

2 Cele aplikacji

- Symulacja sieci sensorycznej.
- Graficzne przedstawienie rozstawienia sensorów oraz ich zasięgów wykrywania.
- Optymalizacja działania sensorów tak by czas życia symulowanej sieci był jak najdłuższy przy jednoczesnym zachowaniu akceptowalnego poziomu pokrycia monitorowanego obszaru.
- Wyświetlanie statystyk po zakończeniu symulacji

3 Opis aplikacji

Menu dla użytkownika zawiera zawierać możliwość wprowadzenia parametrów wejściowych - liczba sensorów, liczba celów. Obie liczby muszą być całkowite i większe od zera, ograniczone z góry możliwościami typów Pythona. Użytkownik może również wprowadzić wymiary prostokątnej mapy (w metrach), te liczby również muszą być większe od zera, będą również ograniczone z góry. Kolejnymi parametrami wejścia, narzuconymi z góry będą pojemność baterii sensora (w mAh), a także zasięg sensora (w metrach).

Po wprowadzeniu parametrów (o ile zostały wprowadzone poprawne wartości) można uruchomić symulację sieci sensorowej. Sensory i cele zostaną losowo rozrzucone na prostokątnej mapie. Symulacja będzie trwała dopóki sensory będą monitorować więcej niż 90% celów (w stosunku do liczby celów, które w ogóle mogły być obserwowane). Aby wydłużyć czas życia sieci sensorowej sensory podczas symulacji będą znajdowały się w jednym z dwóch stanów – aktywności lub uśpienia. Sensory aktywne w danej chwili będą stosownie oznaczone

na mapie. Cele obserwowane w danym momencie również będą odpowiednio zaznaczone.

Po zakończonej symulacji zostaną wyświetlone statystyki działania sieci – parametry wejściowe, czas trwania symulacji, liczba naładowanych i rozładowanych sensorów.

4 Algorytm

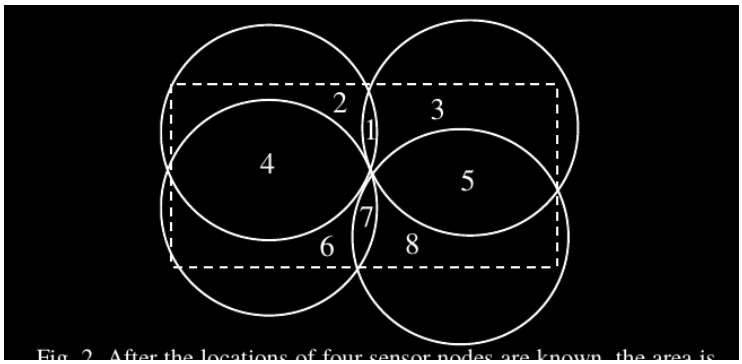
Nazwa Minimally Constraining Paradigm(minimalnie ograniczony paradygmat)[1]

4.1 ogólny opis działania

Zbiór S to zbiór wszystkich sensorów.
Zbiór $C = C_1, C_2, \ldots, C_p$. $\forall C_i \in C \quad C_i$ jest takim podzbiorem zbioru sensorów S, że należące do niego sensory pokrywają co najmniej q procent celów ze zbioru celów T. Celem jest maksymalizacja mocy zbioru C.

- Przypisujemy targety i sensory do pól. Polem nazywamy zbiór targetów pokrywanych przez ten sam zbiór sensorów(patrz rysunek 1)
- Dla każdego sensora tworzymy listę pól jakie pokrywa
- tworzymy listę pól A
- wykonujemy sekwencje dopóki zbiór S nie będzie pusty lub pozostałe w nim sensory będą miały poziom pokrycia celów poniżej q procent.
  - Przypisujemy targety i sensory do pól. Polem nazywamy zbiór targetów pokrywanych przez ten sam zbiór sensorów(patrz rysunek 1)
  - Dla każdego sensora tworzymy listę pól jakie pokrywa
  - tworzymy tymczasową listy pól A i listę sensorów S
  - Szukamy pola pokrytego przez najmniejszą liczbę sensorów (nazywamy takie pole elementem krytycznym).
  - Dla każdego sensora pokrywającego element krytyczny obliczamy wartość funkcji f (patrz wzór 1). Sensor dla którego funkcja f osiąga największa wartość jest wybierany do pokrycia C_i.
  - Usuwamy wybrany sensor ze listy S oraz pole z listy A
  - sprawdzamy czy lista A jest pusta
  - **Jeśli tak**
    - Po kolei wyłączamy wszystkie sensory w pokryciu C_i(pozostałe sensory z C_i są włączone). Dla każdego sensora wyliczamy jaki procent celów był pokryty gdy sensor był wyłączony.
    - Wyłaniamy sensor którego usunięcie z pokrycia C_i najmniej zmniejszy procent pokrytych celów.

[1] Sasa Slijepcevic i Miodrag Potkonjak, *Power Efficient Organization of Wireless Sensor Networks*, 2001.



Rysunek 1: pola

- Jeżeli w wypadku jego usunięcia najlepszego sensora z C_i procent pokrytych celów jest większy lub równy q to sensor jest usuwany z C_i i ponownie dodawany do S.
- Sekwencja jest powtarzana do momentu gdy warunek z poprzedniego punktu jest spełniony.
- Dodajemy powstały zbiór C_i do zbioru pokryć C.
- Ponownie tworzymy zbiór pól A dla sensorów które pozostały w S

Jeśli nie to sekwencje powtarzamy aż do opróżnienia zbioru A

4.2 Funkcja F

4.2.1 Wzór funkcji f

dla sensora V_i.
L* oraz T* są oznaczeniami na zbiory powstałe w sposób opisany w nawiasach.
M Liczba pól które pokrywa dany sensor
N Liczba wszystkich używanych sensorów # oznacza moc zbioru

$$f(V_i) = \sum\{M - T*(\#\quad V_j \in V | e \in V_j)\} - \sum\{N - L*(\#\quad V_j \in C - C_i | e \in V_j)\} \tag{1}$$

5 Opis najważniejszych klas i metod

Program został napisany języku Python

5.1 klasa Parameter

Realizuje wszystko co związane z frontendem aplikacji oraz odpowiada za walidacje wprowadzanych przez użytkownika danych. Wykorzystuje biblioteki pygame oraz PyQt5.

5.2 klasa Statistic

Jej zadaniem jest zwracanie procentu obserwowanych celów oraz sumarycznego czasu symulacji.

5.2.1 metody

stop_ time zapisuje czas końca symulacji

get_ percent_ observed_ targets() zwraca procent obserwowanych celów

get_ simulation_ time() zwraca czas symulacji

5.3 klasa Scheduler

Klasa włącza i wyłącza sensory oraz implementuje algorytm opisany w sekcji 4.

5.3.1 metody

get_ covers_ list() zwraca listę pokryć(C z algorytmu opisanego w sekcji 4)

get_ best_ cover(sensors) Zwraca najbardziej optymalny podzbiór sensorów z listy sensors. Podzbiór jest tym bardziej optymalny im bliżej mu do pokrywania q procent targetów

get_ best_ sensor(critical_ field,cover,uncovered_ fields) zwraca najlepszy sensor pkrywający element krytyczny. sensor jest najlepszy gdy ma najwyższą wartość funkcji f (patrz wzór 1). Pod uwagę brane są tylko sensory pokrywające element krytyczny(critical_ field)

get_ sensor_ value(self,sensor,uncovered_ fields,critical_ field,cover) zwraca wartość funkcji f (wzór 1) dla sensora sensor i elementu krytycznego critical_ field

optimization(cover) usuwa z pokrycia jak najwięcej sensorów, tak by pozostałe sensory nadal pokrywały co najmniej q procent celów

run() odpowiada za egzekucje całej symulacji. uruchamia metody obliczające pokrycia a następnie przeprowadza symulacje, wywołuje funkcje odpowiedzialne za wyświetlanie wszystkiego na ekranie.

build_ fields_ list() tworzy listę pól na podstawie list sensorów i targetów

activate_ covers_ sensors(cover) Zmienia wartość atrybutu active na True we wszystkich sensorach należących do listy cover

disable_ cover(cover) Zmienia wartość atrybutu active na False we wszystkich sensorach należących do listy cover

compute_ sensors_ targets(sensor_ list) przypisuje sensory do targetów i targety do sensorów

get_ critical_ field(fields_ list) zwraca najmniej pokryte pole

sensory	cele	szerokość	wysokość	bateria	czas obliczeń	czas życia	zasięg
400	100	100	100	2	28	50	20
100	400	100	100	2	18	14	20
100	40	100	100	2	0.15	18	20
2000	1000	1000	1000	2	70	7	30
400	100	100	100	2	8.3	78	30

Tablica 1: testy

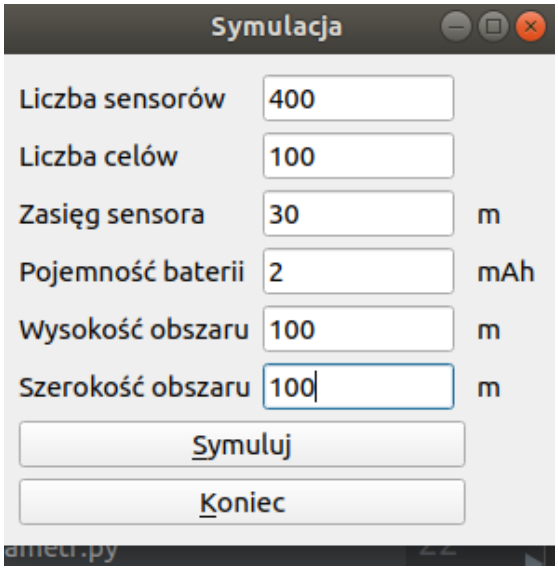
6 Testy

Wyniki testów zostały zaprezentowane w tabeli 1. Z testów wynika kilka wniosków.

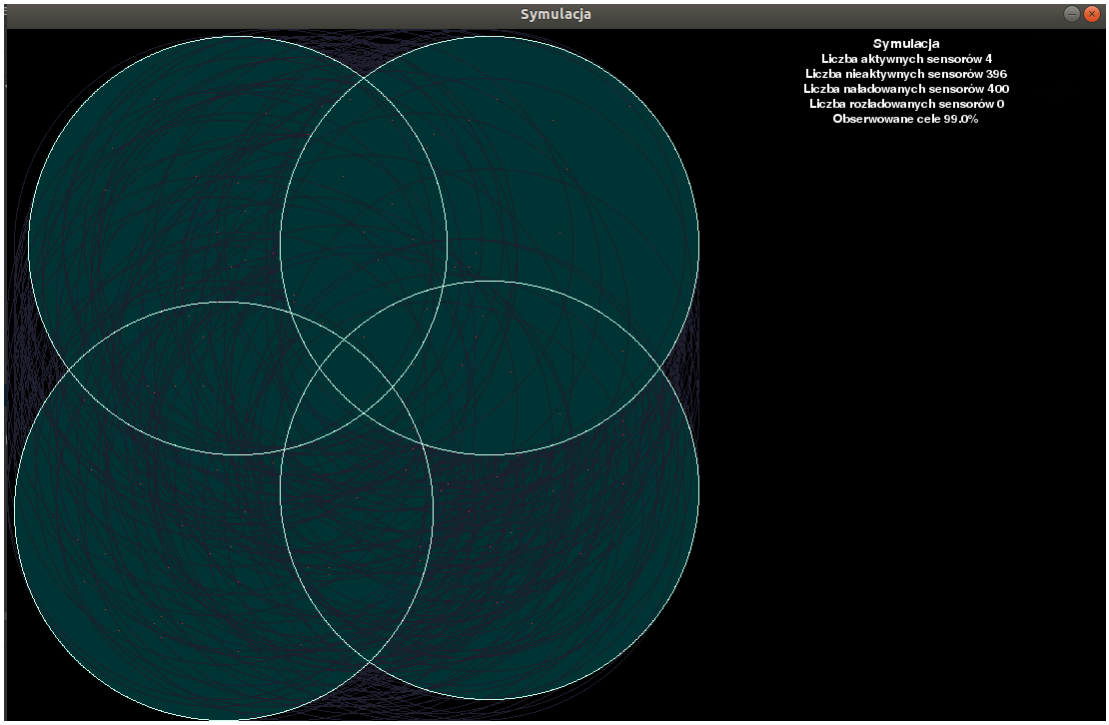
- Duży wpływ na wydajność obliczeń ma zarówno liczba celów jak i sensorów. Liczba sensorów wpływa na szybkość obliczeń bardziej niż liczba celów.
- Przy dużej liczbie sensorów czas życia sieci sensorycznej był nawet o 1 sekundę dłuższy niż przewidywaliśmy. Powodem był czas poświęcany na włączanie, wyłączanie i zerowanie baterii sensorów.
- Zwiększenie zasięgu sensora znacząco obniżało czas obliczeń. Wynika to z faktu że zmniejszała się liczba pól (zawierające targety pokryte przez ten sam zestaw sensorów)

7 Przykład użycia

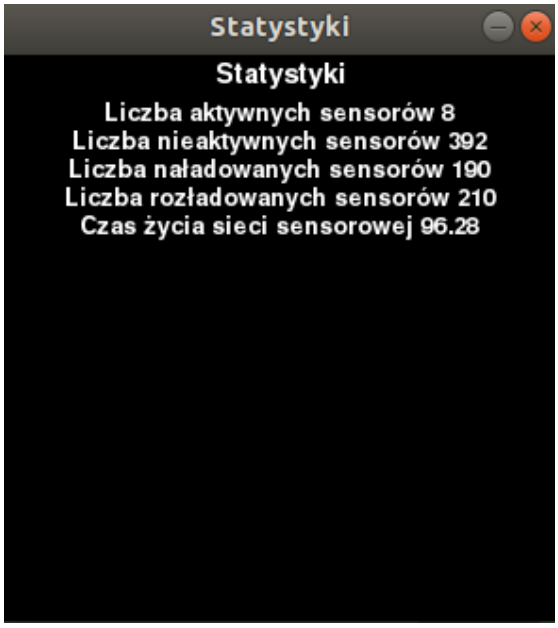
Dla danych 400 sensorów 100 celów 30 zasięg 100 szerokość 100 wysokość 2 bateria. Na grafikach 2, 3 i 4 przedstawiono kolejne etapy działania programu



Rysunek 2: Wprowadzanie danych



Rysunek 3: Symulacja



Rysunek 4: Statystyki