# Rozmieszczanie kamer bezpieczeństwa

Wiktor Franus
Grzegorz Staniszewski

20 stycznia 2018

## Spis treści

# 1 Treść zadania

Jak optymalnie rozmieścić kamery monitoringu w ustalonym pomieszczeniu (rzut z góry), aby minimalną liczbą kamer móc obserwować dowolne miejsce (z uwzględnieniem maksymalnej dopuszczalnej odległości od kamery). W rozwiązaniu należy uwzględnić możliwość zapewnienia parametryzowanej redundancji - tzn. wymagania, aby każde miejsce było obserwowane przez co najmniej $n$ kamer.

# 2 Założenia

1. Pomieszczenie jest wielokątem zawierającym tylko kąty o mierze 90 lub 270 stopni. Pomieszczenie reprezentowane jest przez zbiór punktów (z I ćwiartki układu współrzędnych) podanych w formie listy. Połączenie tych punktów linią, zgodnie z ich kolejnością na liście, skutkuje otrzymaniem linii łamanej ograniczającej pomieszczenie. Punkty podawane są w kolejności zgodnej z ruchem wskazówek zegara. Pierwszy i ostatni punkt jest taki sami (należy domknąć pomieszczenie).

2. Kamery mają jednakowy zasięg reprezentowany przez kwadrat o parametryzowanej długości boku. Współrzędne kamery są jednocześnie współrzędnymi środka tego kwadratu. Kamera musi znajdować się wewnątrz pomieszczenia i nie przenika przez ściany.

3. Wnętrze pomieszczenia zdyskretyzowane jest do zbioru punktów o współrzędnych całkowitych poprzez nałożenie siatki o parametryzowanej gęstości.

4. Punkty leżące na krawędziach wielokąta opisującego pomieszczenie nie należą do jego wnętrza.

# 3 Przestrzeń przeszukiwań

- Elementem przestrzeni przeszukiwań jest wektor par liczb całkowitych oznaczających współrzędne kamer:

$$[(x_1, y_1), ..., (x_i, y_i), ..., (x_k, y_k)]$$

  gdzie:
  $x_i$ - współrzędna x i-tej kamery,
  $y_i$ - współrzędna y i-tej kamery,
  $k$ - liczba kamer.

- Przejście do sąsiedniego elementu możliwe jest poprzez:
  - zmianę położenia jednej z kamer na 2 sposoby (sposób ustalany jest na początku zadania):
    - zmiana współrzędnych x lub y jednej z kamer o 1 jednostkę,
    - przeniesienie jednej z kamer do innego punktu z wnętrza pomieszczenia wylosowanego zgodnie z rozkładem jednostajnym,
  - dodanie nowej kamery w losowym miejscu (rozkład jednostajny),
  - usunięcie jednej kamery.

- Przestrzeń ma strukturę grafową, w której każda krawędź odpowiada jednemu z wymienionych wyżej przejść między elementami przestrzeni.

# 4 Funkcja celu

Informacje znane dla danej instancji problemu:
$X$ - zbiór punktów reprezentujących wnętrze pomieszczenia.
$n_{kmin}$ - minimalna teoretyczna liczba kamer wymagana do pokrycia danego pomieszczenia (obliczana jako stosunek liczności zbioru $X$ do liczby punktów pokrywanych przez 1 kamerę, zaokrąglany do jedności w górę),

Parametry funkcji celu:
$\alpha$ - zysk z pokrywania powierzchni pomieszczenia,
$\beta$ - koszt użycia nadmiarowej kamery,
$r_{min}$ - minimalna liczba kamer pokrywająca każde miejsce w pomieszczeniu.

Zadanie polega na maksymalizacji funkcji:

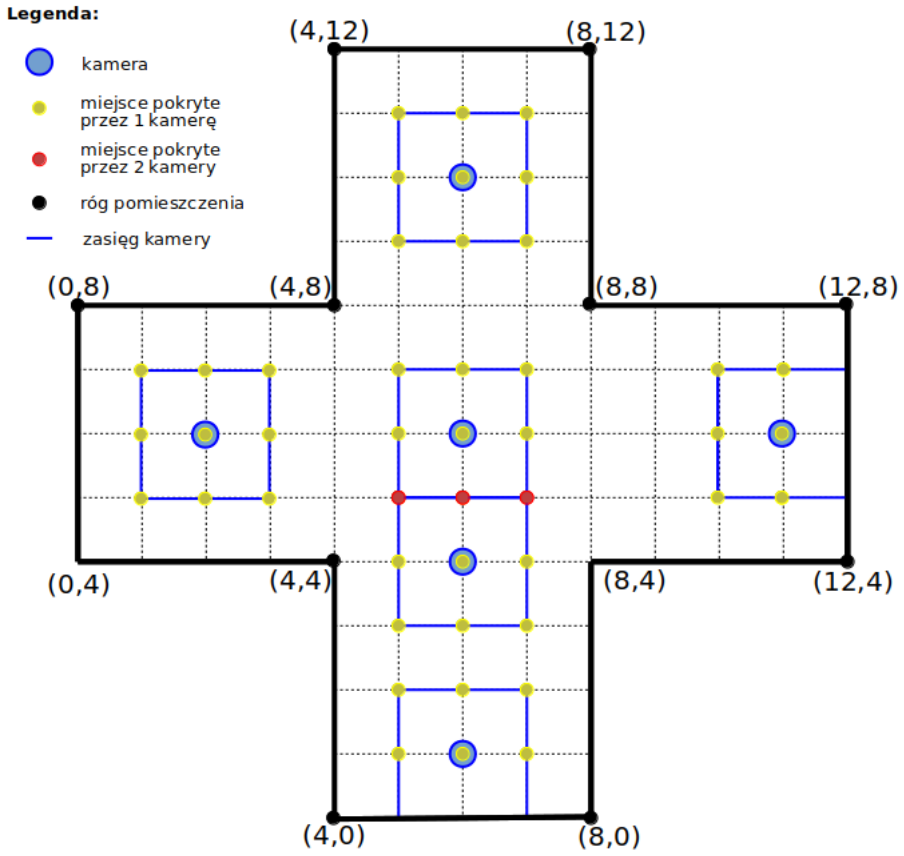
$$f(p,k,r) = \alpha * p - \beta * k - \frac{1}{r_{min}} * r$$

gdzie:
$p$ - stosunek liczby punktów pokrytych przez kamery do liczności zbioru $X$
$k$ - stosunek nadwyżki liczby kamer do $n_{kmin}$, obliczany wg. wzoru:
$k = \frac{max(0, n_k - n_{kmin})}{n_{kmin}}$, gdzie $n_k$ - liczba kamer w aktualnym stanie

Parametr $r$ może być obliczany na dwa sposoby (sposób ustalany jest na początku zadania):

- jako średni stopień niespełnienia warunku redundancji dla punktu z wnętrza pomieszczenia, obliczany wg. wzoru:
  $r = \frac{\sum_{x \in X} max(0, r_{min} - r_x)}{|X|}$, gdzie $r_x$ - liczba kamer pokrywających punkt $x$
- jako maksymalne niespełnienie warunku redundancji spośród wszystkich punktów z wnętrza pomieszczenia, wg. wzoru:
  $r = max(0, r_{min} - r_x)$, gdzie $r_x$ - liczba kamer pokrywających punkt $x$, który jest najsłabiej pokrytym punktem.

# 5 Przykład




- Wartości parametrów:
  $\alpha = 1$
  $\beta = 1$
  $r_{min} = 1$

- Informacje obliczone dla powyższego pomieszczenia:
  Rozmiar boku kwadratu reprezentującego zasięg kamery: 2
  Liczba punktów pokrywanych przez jedną kamerę: 9
  $|X| = 57$
  $n_{kmin} = \lceil \frac{57}{9} \rceil = 7$

- Obliczenie wartości funkcji celu dla stanu z rysunku:
  Liczba kamer użytych: 6
  Liczba punktów pokrytych przez kamery: 45
  $p = \frac{45}{57} = 0.7895$
  $k = \frac{max(0,6-7)}{7} = \frac{0}{20} = 0$
  Parametr $r$ obliczany pierwszym sposobem:
  $r = \frac{45*0+12*1}{57} = \frac{12}{57} = 0.2105$
  $f(p,k,r) = 1 * 0.7895 - 1 * 0 - 1 * 0.2105 = 0.5790$

  Parametr $r$ obliczany drugim sposobem:

$r = max(0, 1 - 0) = 1$
$f(p, k, r) = 1 * 0.7895 - 1 * 0 - 1 * 1 = -0.2105$

# 6 Metaheurystyka

Element początkowy przestrzeni przeszukiwań jest zbiorem zawierającym $n_{kmin}$ kamer rozmieszczonych losowo wewnątrz pomieszczenia.
Do rozwiązania problemu użyjemy algorytmu symulowanego wyżarzania. Przy odpowiednio dobranych parametrach metoda ta, w porównaniu do algorytmów wspinaczkowych, daje większą szansę na znalezienie optymalnego rozwiązania, ponieważ zmniejsza ryzyko zatrzymania się w optimach lokalnych. W początkowej fazie przeszukiwania przestrzeni dopuszczalne jest przechodzenie do stanów gorszych (o mniejszej wartości funkcji celu). Wraz z rosnącą liczbą iteracji algorytm bardziej skupia się na poprawie bieżącego rozwiązania.

# 7 Przewidywane wyniki pracy

Przeprowadzona zostanie seria eksperymentów z różnymi wartościami parametrów $\alpha$, $\beta$, $r_{min}$ na kilku instancjach problemu (różne pomieszczenia). Dla ustalonych parametrów funkcji celu, sterować będziemy parametrami metaheurystyki, tj. funkcją wygaszania temperatury i jej wartością początkową. Ponadto sprawdzimy dwa podejścia do zmiany położenia kamery oraz dwa sposoby obliczania parametru $r$ funkcji celu. Dla wybranej instancji zadania sprawdzimy też wpływ gęstości siatki punktów z wnętrza pomieszczenia na zachowanie metaheurystyki. Sporządzone zostaną wykresy przedstawiające wartość funkcji celu oraz liczbę użytych kamer w zależności od liczby wykonanych iteracji.

# 8 Implementacja

Do realizacji zadania wykorzystaliśmy gotową implementację metaheurystyki zawartą w pakiecie simanneal w wersji 0.4.1, której dokumentacja jest dostępna pod adresem `https://github.com/perrygeo/simanneal`. Biblioteka implementuje algorytm symulowanego wyżarzania z wykładniczą funkcją wygaszania temperatury, minimalizujący zadaną funkcję celu. Z racji, że naszym zadaniem miało być maksymalizowanie opisanej w specyfikacji funkcji celu, musieliśmy zmodyfikować tę funkcję, zmieniając jej znak na przeciwny. Poprawiony wzór funkcji celu ma postać:

$$f(p, k, r) = -(\alpha * p - \beta * k - \frac{1}{r_{min}} * r)$$

Biblioteka simanneal udostępnia użytkownikowi 3 parametry, którymi można sterować zachowaniem metaheurystyki:

- *steps* - liczba iteracji algorytmu, domyślnie 50000,
- *Tmax* - początkowa wartość temperatury, domyślnie 25000,
- *Tmin* - końcowa wartość temperatury, domyślnie 2.5.

Początkowo sprawdziliśmy zachowanie metaheurystyki dla domyślnych wartości udostepnionych parametrów, jednak okazały się one nietrafione. Postanowiliśmy zmniejszyć temperaturę początkową 100-krotnie, czyli do wartości 250. W rezultacie na wykresie wartości funkcji celu względem

numeru iteracji zaczęły pojawiać się charakterystyczne dla symulowanego wyżarzania „skoki", których nasilenie zmniejszało się z kolejnymi iteracjami. Zaobserwowaliśmy także, że zmniejszenie liczby iteracji 2-krotnie, czyli do wartości 25000, nie wpłynęło znacząco na uzyskiwane rezultaty, ale za to, zgodnie z intuicją, skrócił się czas obliczeń. Badania postanowiliśmy zatem przeprowadzać na 25000 iteracjach.

## 8.1 Plik konfiguracyjny

Przykładowy plik konfiguracyjny wraz z komentarzami znajduje się poniżej.

```
{
  "room": [                       // Pomieszczenie opisane jako lista wierzchołków,
    {                             // zgodnie z założeniami:
      "x": 0.0,                   // * punkty podawane są w kolejności
      "y": 0.0                    //   zgodnej z ruchem wskazówek zegara,
    },                            // * Pomieszczenie jest wielokątem zawierającym
    {                             //   tylko kąty o mierze 90 lub 270 stopni,
      "x": 0.0,
      "y": 4.0
    },
    {
      "x": 4.0,
      "y": 4.0
    },
    {
      "x": 4.0,
      "y": 0.0
    },
    {
      "x": 0.0,                   // * ostatni punkt musi domykać pomieszczenie,
      "y": 0.0                    //   musi pokrywać się z pierwszym na liście.
    }
  ],
  "t_max": 2500.0,                // Temperatura maksymalna
  "t_min": 2.5,                   // Temperatura minimalna
  "alpha": 10,                    // Zysk z pokrywania powierzchni pomieszczenia
  "beta": 1,                      // Koszt użycia nadmiarowej kamery
  "r_min": 1,                     // Minimalna liczba kamer
                                  // pokrywająca każde miejsce w pomieszczeniu
  "num_iterations": 10,           // Liczba iteracji
  "num_updates" : 1,              // Liczba wyników pośrednich (nie ma wpływu na wyniki)
  "camera_move_method": "local",  // Sposób przesuwania kamery:
                                  // * local - o 1 jednostkę w dowolnym kierunku
                                  // * random - w dowolne miejsce w pomieszczeniu
  "camera_side" : 4,              // Długość boku kwadratu reprezentującego zasięg kamery
  "r_count_method": "average",    // Sposób wyznaczania niespełnienia redundancji:
                                  // * average - średni stopień
                                  // * max - maksymalne
  "density" : 1                   // Odległość pomiędzy punktami (gęstość siatki)
}
```

## 8.2 Sposób uruchomienia

Program wymaga do działania Pythona w wersji 3.x. Przed uruchomieniem programu należy zainstalować wymagane pakiety języka Python znajdujące się w pliku `pip.requirements`. Można to zrobić poleceniem:

```
pip3 install -r pip.requirements
```

Opis parametrów uruchomienia dostepny jest po wpisaniu polecenia:
`python3 main.py --help`

```
Usage: main.py [options]

Options:
  -h, --help            show this help message and exit
  -f CONFIGFILE, --config=CONFIGFILE
                        Json file with configuration
  -e EXPERIMENTS, --experiments=EXPERIMENTS
                        Number of experiments
```

Domyślnie program wczytuje konfigurację z pliku o nazwie `config.json`, a liczba eksperymentów (liczba uruchomień) wynosi 1. Wyniki badań zapisywane są w katalogu `./out/<data>`. W wyjściowym katalogu tworzone są podkatalogi dla każdej iteracji programu (eksperymentu). Każdy podkatalog zawiera wykres przedstawiający wartości funkcji celu względem liczby iteracji. Wykres zapisany jest w pliku o nazwie `costs.png`. W tym samym podkatalogu znajdują się także pliki graficzne przedstawiające stan pomieszczenia w wybranych, równomiernie rozłożonych punktach czasowych. Liczbę tych punktów można określić zmieniając wartość parametru *num_ updates* w pliku konfiguracyjnym. Dodatkowo w pliku `best_state.png` zapisany jest wygląd stanu (pomieszczenia), w którym wartość funkcji celu była minimalna. W pliku `./out/<data>/average_costs.png` znajduje się wykres przedstawiający średnią wartość funkcji celu dla wszystkich iteracji (eksperymentów). Ponadto do katalogu wyjściowego kopiowany jest plik konfiguracyjny użyty w danym uruchomieniu programu.

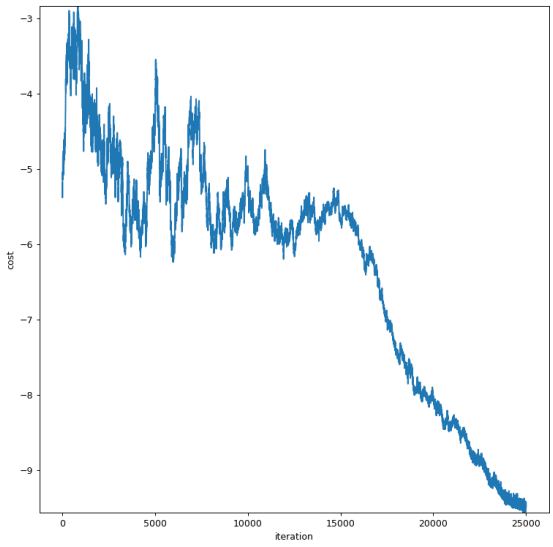
# 9 Badania

## 9.1 Wpływ liczby iteracji na wartość funkcji celu

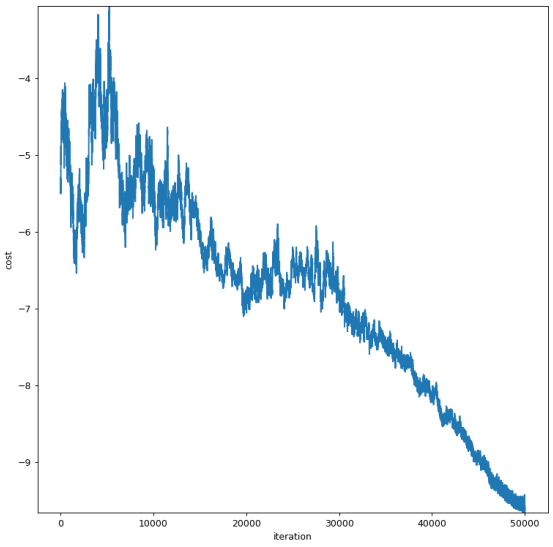
W badaniu sprawdzaliśmy wpływ liczby iteracji algorytmu symulowanego wyżarzania na wygląd wykresu przedstawiającego wartość funkcji celu. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

```
"t_max": 250.0,
"t_min": 0.01,
"alpha": 10,
"beta": 1,
"r_min": 1,
"num_iterations": <zmienna>,
"num_updates" : 10,
"camera_move_method": "local",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```

Wyniki badania dla 25 tyś i 50 tyś iteracji zaprezentowane zostały na rys. 1. Wykresy są do siebie podobne, w szczególności granica między fazą eksploracji i eksploatacji występuje mniej więcej w tym samym miejscu, czyli po ok. 60% iteracji. Osiągane wartości minimalne funkcji celu są w obu przypadkach takie same i wynoszą ok. -9.5. Większa liczba iteracji wpływa jednak negatywnie na ilość czasu potrzebną na wykonanie badania. Z tego powodu wszystkie pozostałe badania postanowiliśmy przeprowadzić na 25 tysiącach iteracji.




(a) 25000 iteracji



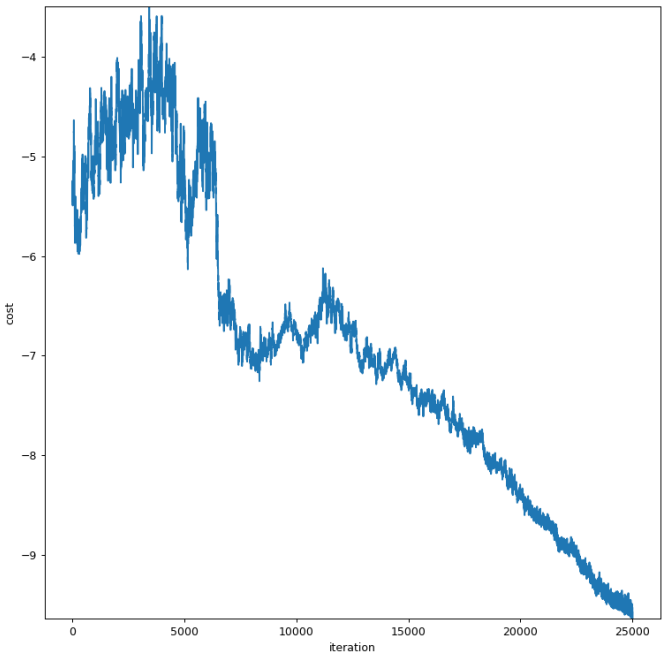

(b) 50000 iteracji

Rysunek 1: Wpływ liczby iteracji na wartość funkcji celu.

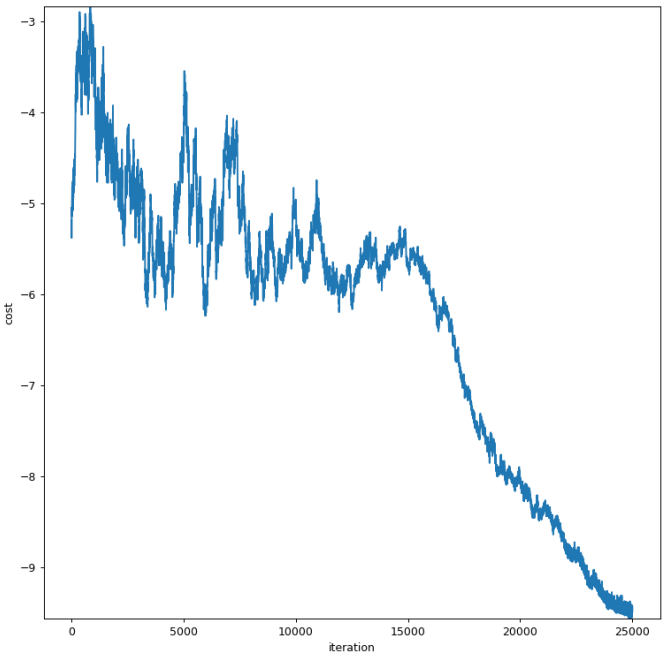
## 9.2 Wpływ wartości temperatury początkowej funkcji wygaszania na wartość funkcji celu

W badaniu sprawdzaliśmy wpływ początkowej wartości temperatury na wygląd wykresu przedstawiającego wartość funkcji celu dla 25000 iteracji. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

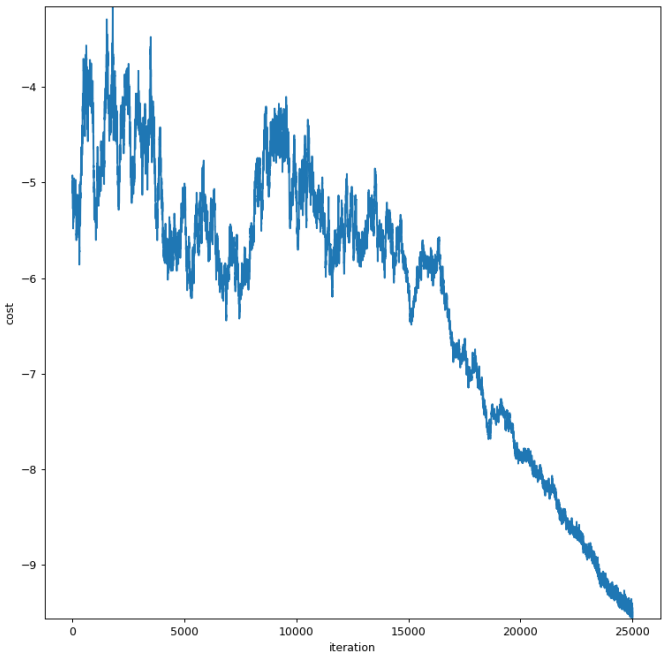
```
"t_max": <zmienna>,
"t_min": 0.01,
"alpha": 10,
"beta": 1,
"r_min": 1,
"num_iterations": 25000,
"num_updates" : 10,
"camera_move_method": "local",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```



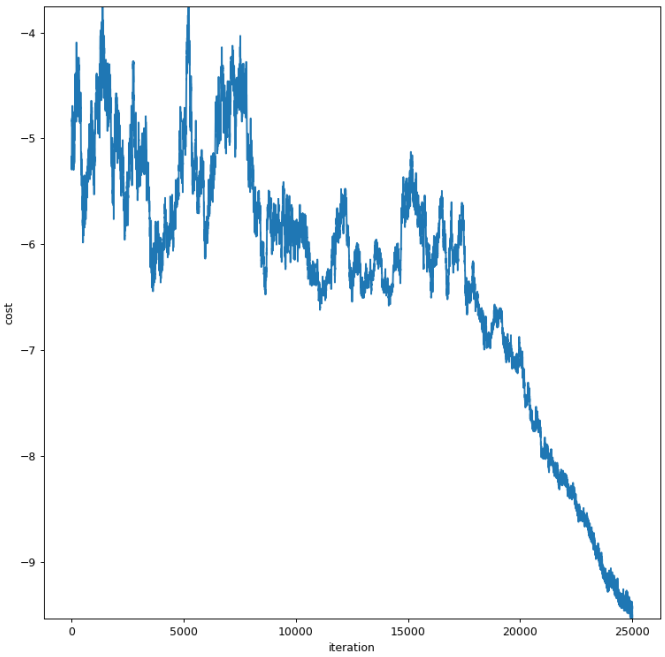

(a) $T_{max} = 50$




(b) $T_{max} = 250$




(c) $T_{max} = 2500$




(d) $T_{max} = 25000$

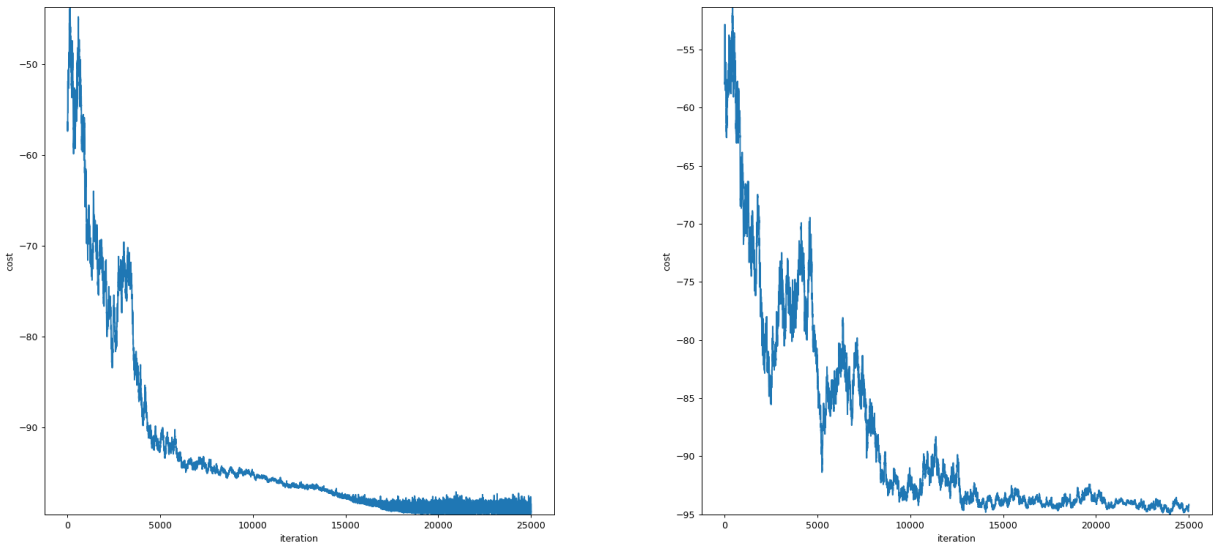
Rysunek 2: Wpływ parametru $T_{max}$ na wartość funkcji celu.

Badanie zostało przeprowadzone dla czterech różnych wartości parametru $T_{max}$: 50, 250, 2500 i 25000. Wykresy zostały z przedstawione na rys. 2. Widoczny jest wpływ parametru $T_{max}$ na długość fazy eksploracji, czyli okresu, w którym algorytm akceptuje rozwiązania gorsze od aktualnego. Po fazie eksploracji następuje faza eksploatacji, kiedy to algorytm stara się polepszyć aktualne rozwiązanie, coraz rzadziej pozwalając na jego pogarszanie. Dla $T_{max} = 50$ granica pomiędzy wspomnianymi fazami przeszukiwania umiejscowiona jest w pobliżu 12000 iteracji, dla $T_{max} = 250$ w pobliżu 15000 iteracji, dla $T_{max} = 2500$ w pobliżu 16000 iteracji, a dla $T_{max} = 25000$ około 19000 iteracji.

## 9.3 Wpływ wartości temperatury końcowej funkcji wygaszania na wartość funkcji celu

W badaniu sprawdzaliśmy wpływ końcowej wartości temperatury na wygląd wykresu przedstawiającego wartość funkcji celu dla 25000 iteracji. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

```
"t_max": 50.0,
"t_min": <zmienna>,
"alpha": 100,
"beta": 1,
"r_min": 1,
"num_iterations": 25000,
"num_updates" : 10,
"camera_move_method": "local",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```




(a) $T_{min} = 0.001$ (b) $T_{min} = 1$

Rysunek 3: Wpływ parametru $T_{min}$ na wartość funkcji celu.

Badanie zostało przeprowadzone dla dwóch wartości parametru $T_{min}$ (rys. 3). Dla wartości 0.001 wykres wartości funkcji celu względem numeru iteracji jest bardziej gładki niż dla wartości 1. Temperatura ma bezpośredni wpływ na wartość prawdopodobieństwa akceptacji rozwiązań gorszych niż obecne. Większa wartość $T_{min}$ implikuje większą wartość tego prawdopodobieństwa, a co za tym idzie większe oscylacje na wykresie w końcowym iteracjach.

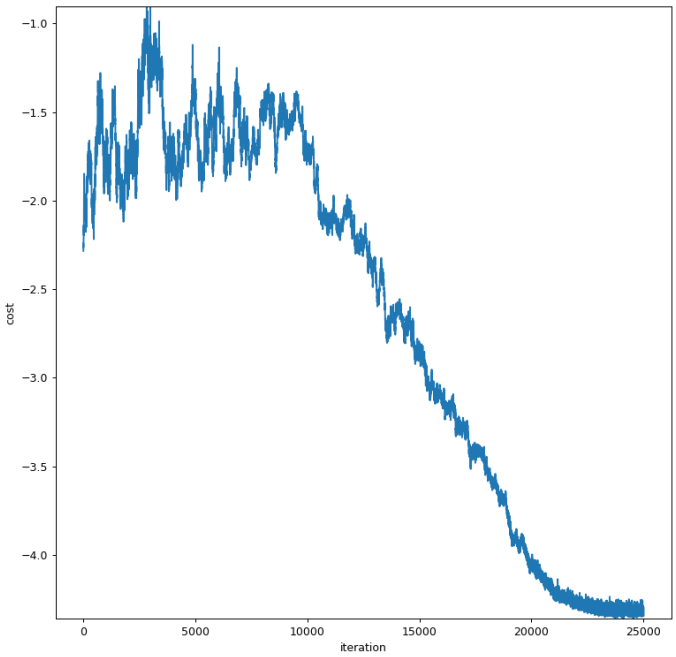
## 9.4 Wpływ wartości parametru *alpha* na wartość funkcji celu i końcowe rozmieszczenie kamer

W badaniu sprawdzaliśmy wpływ wartości parametru *alpha* na wygląd wykresu przedstawiającego wartość funkcji celu dla 25000 iteracji oraz wpływ tego parametru na końcowe rozmieszczenie kamer w pomieszczeniu. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

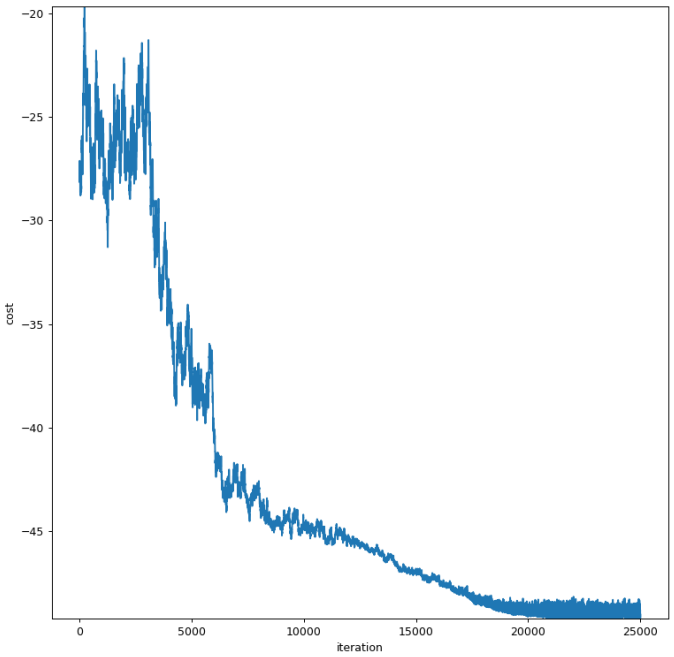
```
"t_max": 250.0,
"t_min": 0.001,
"alpha": <zmienna>,
"beta": 1,
"r_min": 2,
"num_iterations": 25000,
"num_updates" : 10,
"camera_move_method": "local",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```

Badanie zostało przeprowadzone dla trzech różnych wartości parametru *alpha*: 5, 50, 100, przy wymaganej redundancji równej 2 (każdy punkt ma być pokryty przez dwie kamery). Rys. 4 prezentuje wyniki badania. Wzrost wartości parametru *alpha* skutkuje liniowym spadkiem wartości funkcji celu np. przy *alpha* równym 5, wartości funkcji celu należą do przedziału od -1 do -5, przy *alpha* równym 50 do przedziału od -20 do -50, a w ostatnim przypadku do przedziału od -35 do -100.

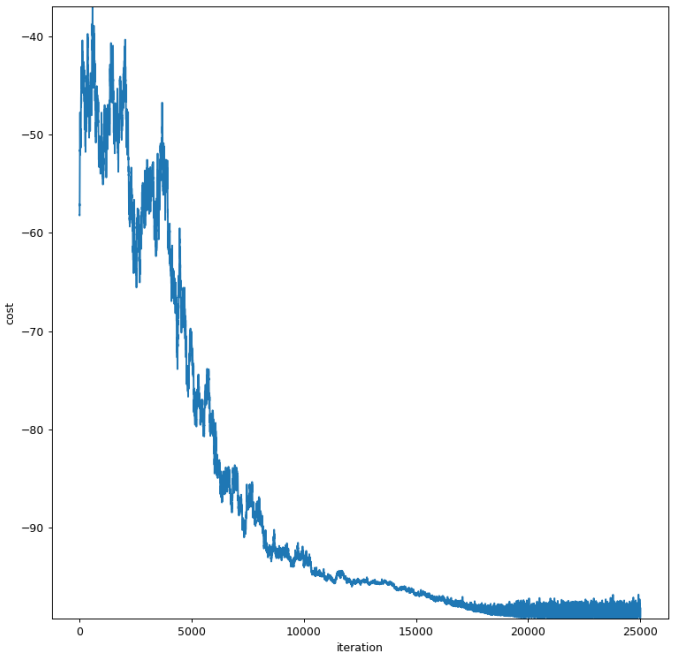
Rozmieszczenie kamer w pomieszczeniu zostało zaprezentowane na rys. 5. Liczba używanych kamer zależna jest od stosunku wartości parametrów *beta* i *alpha*. W przypadku, gdy stosunek ten jest mały, czyli dla *beta* równego 1 i *alpha* równego 5, pomieszczenie nie zostaje pokryte w całości, a tym bardziej każdy punkt nie jest pokryty przez dwie kamery. Zwiększenie wartości parametru *alpha* do 50 skutkuje wzrostem liczby kamer i wzrostem redundancji dla pojedynczego punktu, ponieważ dodawanie kamery jest wtedy mało kosztowne. Po zwiększeniu wartości parametru *alpha* do 100 widoczna jest podobna zależność. Można również zauważyć, że pomimo niskiego kosztu użycia kamery, w pomieszczeniu występują miejsca, w których punkty są pokryte tylko przez 1 kamerę. Dzieje się tak, ponieważ koszt użycia dodatkowej kamery do zysku z redundancji jest zbyt duży.




(a) $alpha = 5$

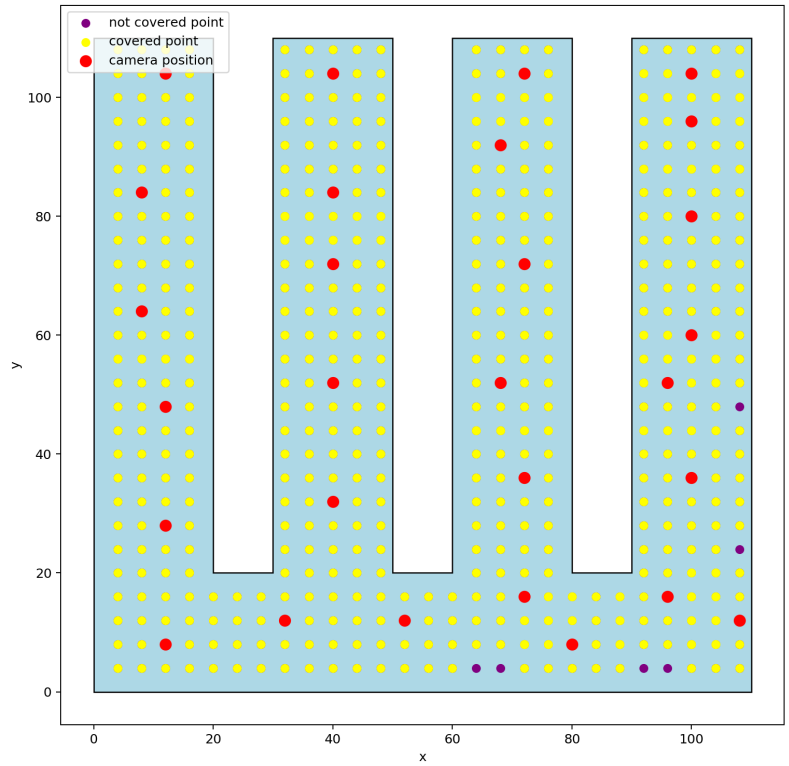



(b) $alpha = 50$

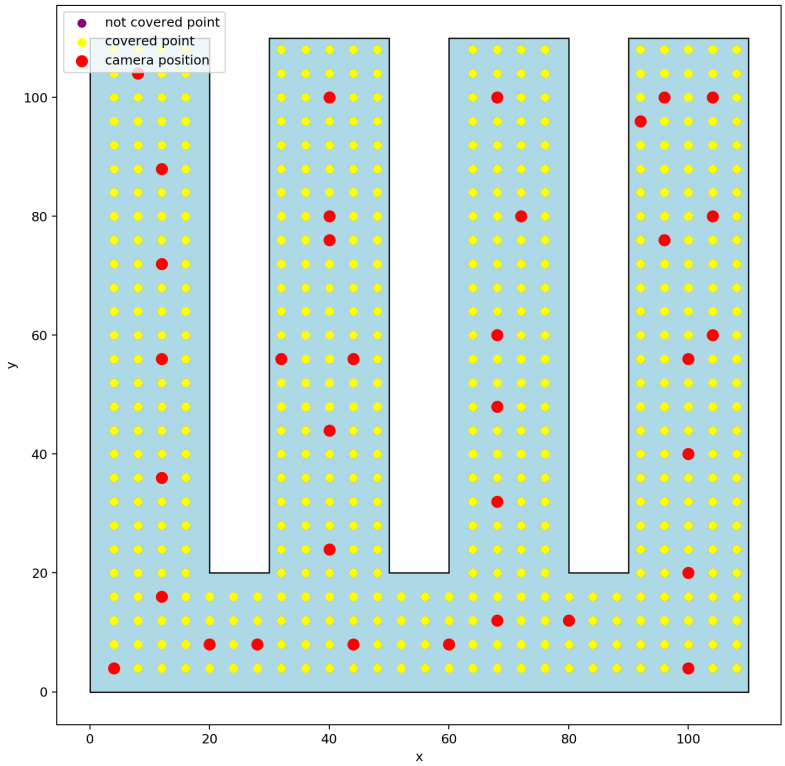



(c) $alpha = 100$

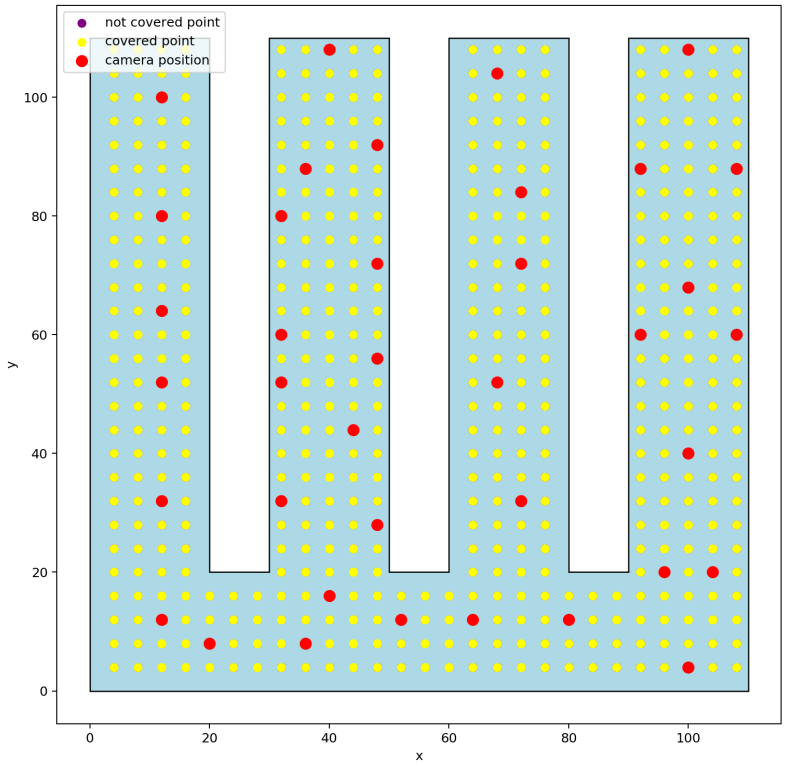
Rysunek 4: Wpływ parametru *alpha* na wartość funkcji celu.




(a) $alpha = 5$




(b) $alpha = 50$



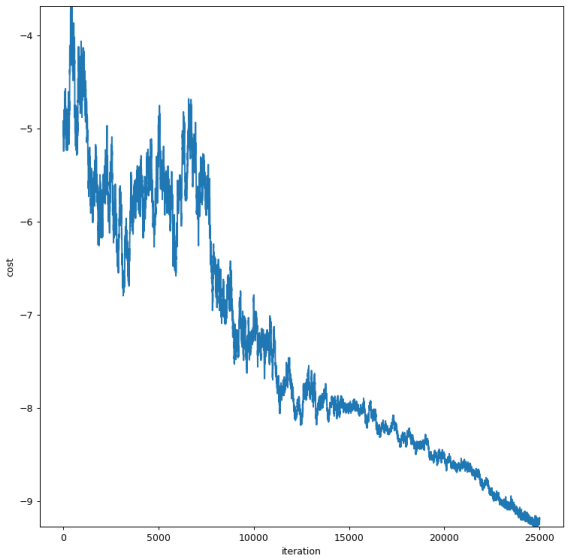

(c) $alpha = 100$

Rysunek 5: Wpływ parametru *alpha* na końcowe rozmieszczenie kamer

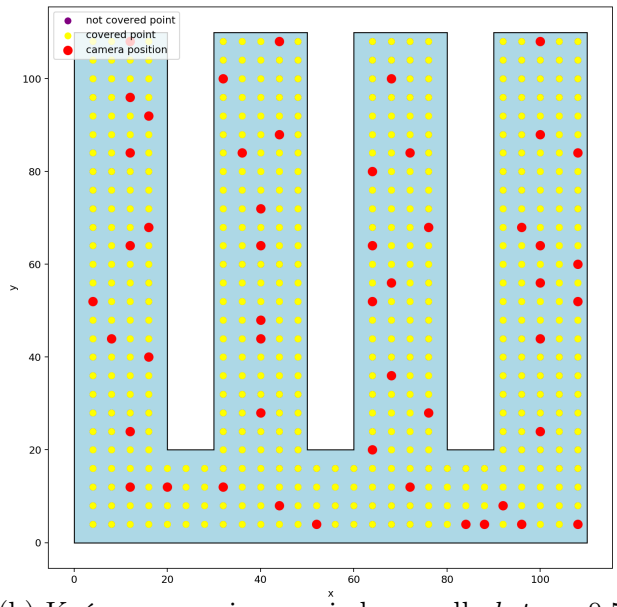
## 9.5 Wpływ wartości parametru *beta* na wartość funkcji celu i końcowe rozmieszczenie kamer

W badaniu sprawdzaliśmy wpływ wartości parametru *beta* na wygląd wykresu przedstawiającego wartość funkcji celu dla 25000 iteracji oraz wpływ tego parametru na końcowe rozmieszczenie kamer w pomieszczeniu. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

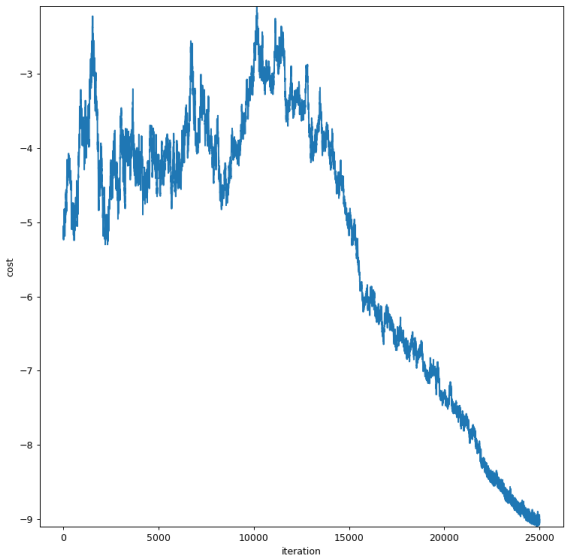
```
"t_max": 200.0,
"t_min": 0.01,
"alpha": 10,
"beta": <zmienna>,
"r_min": 2,
"num_iterations": 25000,
"num_updates" : 100,
"camera_move_method": "local",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```



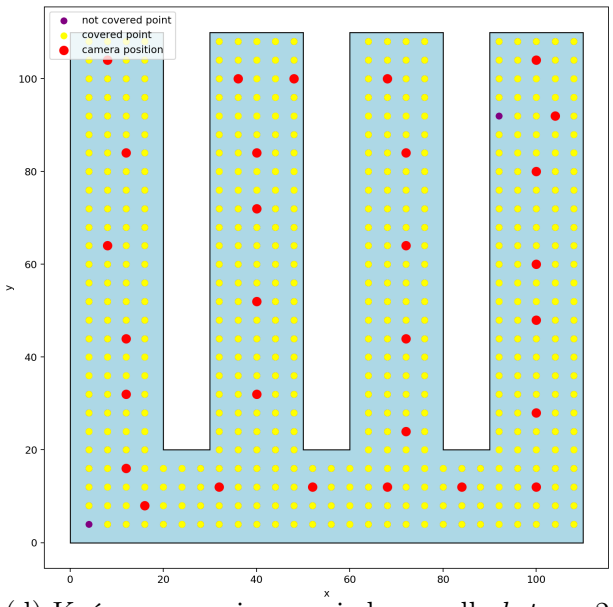

(a) Wartość funkcji celu dla $beta = 0.5$



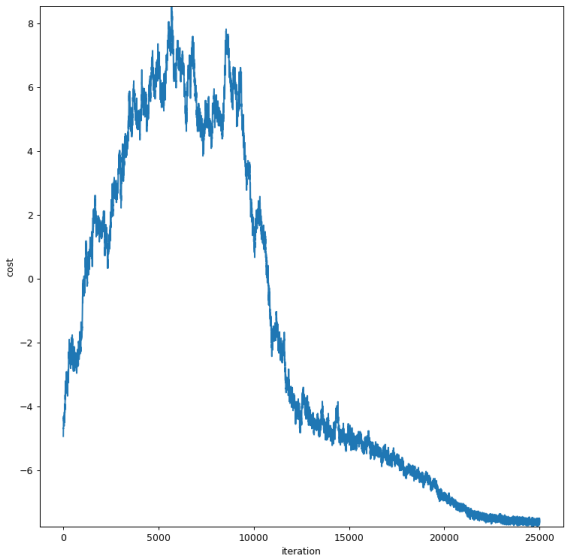

(b) Końcowe rozmieszczenie kamer dla $beta = 0.5$



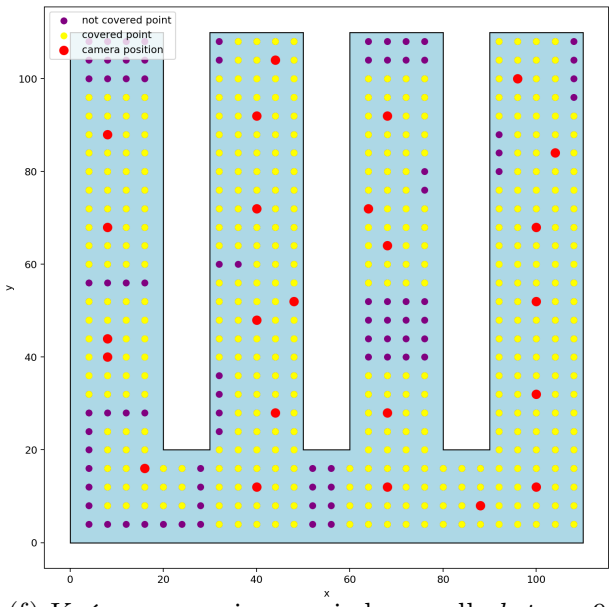

(c) Wartość funkcji celu dla $beta = 2$




(d) Końcowe rozmieszczenie kamer dla $beta = 2$




(e) Wartość funkcji celu dla $beta = 9$




(f) Końcowe rozmieszczenie kamer dla $beta = 9$

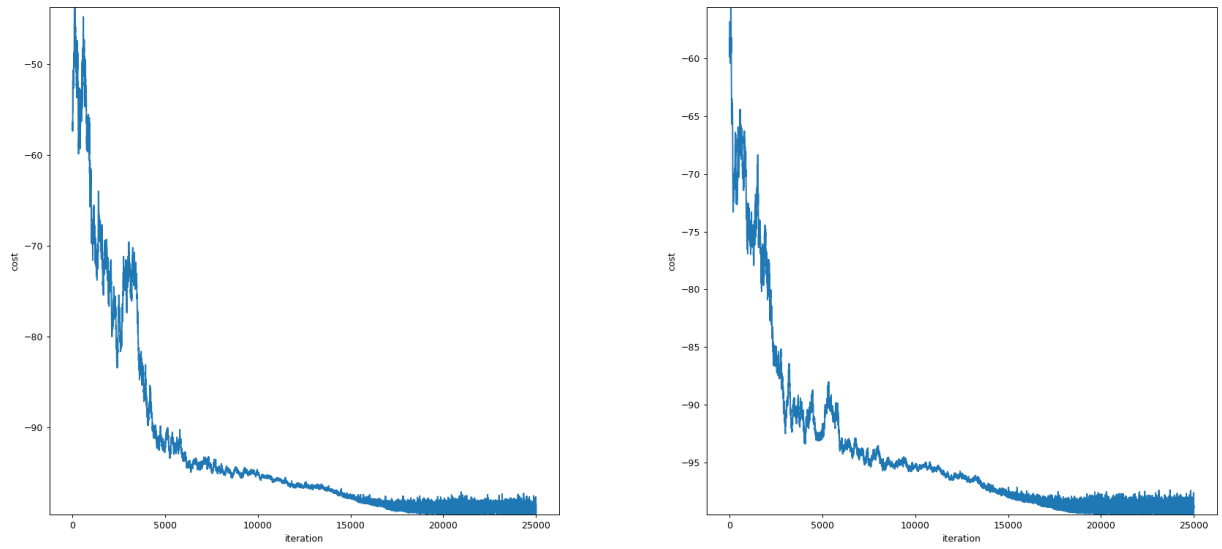
Rysunek 6: Wpływ parametru *beta* na wartość funkcji celu i końcowe rozmieszczenie kamer.

Badanie zostało przeprowadzone na trzech wartości parametru *beta* (rys. 6). Parametr $r_{min}$ miał stałą wartość równą 2, co oznacza, że każdy punkt z wnętrza pomieszczenia powinien być pokryty przez co najmniej dwie kamery. Niska wartość *beta*, czyli niska wartość kary za użycie nadmiarowej kamery sprawiła, że użytych zostało za dużo kamer, co widać gołym okiem na rys. 6b. Zwiększenie wartości *beta* do 2 spowodowało, że użytych zostało mniej kamer niż za pierwszym razem, a ich rozmieszczenie wydaje się być bardziej sensowne. Dla *beta* = 9 zaprezentowane rozwiązanie końcowe jest dalekie od optymalnego. Widocznych jest dużo punktów nie pokrytych przez żadną kamerę.

## 9.6 Wpływ wartości parametru *camera_move_method* na wartość funkcji celu

W badaniu sprawdzaliśmy wpływ wartości parametru *camera_move_method*, czyli sposobu w jaki przesuwana jest kamera, na wygląd wykresu przedstawiającego wartość funkcji celu. Wartości pozostałych parametrów były stałe - zostały one zaprezentowane poniżej:

```
"t_max": 50.0,
"t_min": 0.001,
"alpha": 100,
"beta": 1,
"r_min": 1,
"num_iterations": 25000,
"num_updates" : 10,
"camera_move_method": "local|random",
"camera_side" : 20,
"r_count_method": "average",
"density" : 4
```




(a) *camera_move_method* = "local" (b) *camera_move_method* = "random"

Rysunek 7: Wpływ sposobu przesuwania kamery na wartość funkcji celu.

Wyniki badania dla dwóch dozwolonych wartości parametru *camera_move_method* zaprezentowane zostały na rys. 7. Wykresy są do siebie podobne, jednak można zauważyć pewną różnicę

w pobliżu 5000 iteracji. W przypadku, gdy przemieszczamy kamerę w losowe miejsce, istnieje większa szansa na gwałtowną zmianę wartości funkcji celu niż wtedy, gdy kamera jest przesuwana o jedną jednostkę w dowolnym kierunku. Na wykresie objawia się to większymi „skokami" funkcji celu (rys. 7a). Gwałtowność zmian wartości funkcji celu w sąsiednich stanach spowodowana może być nie tylko zmianą położenia jednej z kamer, ale również dodaniem lub usunięciem kamery. Można więc założyć, że gdy parametr *camera_move_method* ma wartość "local", to gwałtowną zmianę wartości funkcji celu podczas przejścia do kolejnego stanu powodują dwa czynniki: dodanie i usunięcie kamery. Jeśli natomiast kamera będzie przesuwana w losowe miejsce, to przesunięcie kamery staje się trzecim czynnikiem wpływającym na występowanie gwałtownych zmian wartości funkcji celu. Kombinacja wartości pozostałych parametrów użytych w doświadczeniu sprawiła, że sama wartość parametru *camera_move_method* nie miała wpływu na końcową wartość funkcji celu, która przyjęła wartość ok. -95.