Algorytmy metaheurystyczne

Lista 2

Piotr Syga

5 kwietnia 2020

Termin: 2020-04-26, 09:59:59

1 Cel listy

Celem listy jest praktyczne przećwiczenie metaheurystyk opratych na symulowanym wyżarzaniu. Dobór parametrów (np. temperatura początkowa, tempo wychładzania czy też poziom akceptacji) należy do autora programu. W czasie rozwiązywania listy autor powinien rozpoznać jaki wpływ na działanie programu (czas działania, wymagania pamięciowe, osiągany rezultat, podatność na utknięcia w lokalnym minimum, ...) mają poszczególne parametry.

2 Wymagania formalne

Rozwiązanie listy powinno zostać umieszczone w ścieżce `amh/l2/` (case sensitive) głównego katalogu studenta na repozytorium svn—`https://156.17.7.16/`. Rozwiązanie każdego zadania powinno znajdować się w odpowiednim folderze `zi/`, dla zadania `i`. Poza plikami źródłowymi rozwiązania, w tym samym folderze należy umieścić `makefile`, który po wpisaniu polecenia `make` utworzy w bieżącym folderze plik wykonywalny `main`. Wszystkie dane wejściowe podawane są na standardowym wejściu, na standardowym wyjściu powinno znajdować się jedynie rozwiązanie w ustalonym formacie. Program będzie uruchamiany poprzez przekierowanie danych wejściowych i wyjściowych z/do plików: `./main <In >Out` (lub `./main <In >Out 2>Err`). Program **nie może** wykorzystywać obliczeń równoległych, w szczególności zabronione jest testowanie kilku rozwiązań jednocześnie.

3 Zadania

1. Napisz program, który za pomocą symulowanego wyżarzania znajdzie minimum funkcji Salomona zadanej wzorem

$$f(\bar{x}) = 1 - \cos(2\pi\sqrt{\sum_{i=1}^{4} x_i^2}) + 0.1\sqrt{\sum_{i=1}^{4} x_i^2}$$

   In: Piątka liczb całkowitych `t x1 x2 x3 x4` oddzielonych spacją.
   `t` – maksymalna liczba sekund, która może wykonywać się program w tym uruchomieniu, $\bar{x} =$ (`x1`, `x2`, `x3`, `x4`) – jest rozwiązaniem początkowym (podane liczby zawsze są całkowite, program natomiast może je interpretować jako dowolny typ liczbowy).

   Out: 5 liczb typu `double` oddzielonych spacją, z czego cztery pierwsze to $\bar{x}$, natomiast piąta to wartość odpowiedniej funkcji w punkcie $\bar{x}$.

2. Napisz program, który dla przedstawionej macierzy liczb całkowitych (zakres wartości między 0 a 255, można przyjąć typ `uint8` lub odpowiednik) M, znajdzie najbliższą macierz M', spełniającą poniższe ograniczenia

   - wykorzystujemy co najwyżej 8 różnych wartości liczbowych: 0, 32, 64, 128, 160, 192, 223, 255,
   - macierz wynikowa M' składa się z bloków $a \times b$, takich że $a \geqslant k$ i $b \geqslant k$, wewnątrz jednego bloku wszystkie liczby mają równą wartość,
   - odległość między macierzami M i M', rozmiaru $n \times m$, liczymy jako $d_{\mathrm{MSE}}(M,\, M') = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(M(i,\, j) - M'(i,\, j)\right)^2$.

   In: Dane wejściowe składać się będą z $n+1$ linii. W pierwszej linii będą umieszczone, oddzielone spacją liczby całkowite `t`, n, m, k, gdzie `t` jest limitem czasu, jak w zadaniu 1, natomiast n i m oznaczają wymiary macierzy początkowej M, a k jest parametrem odpowiadającym za rozmiar bloków w macierzy wynikowej M'. W kolejnych n liniach będą znajdowały się wartości kolejnego wiersza macierzy (dokładnie m liczb oddzielonych spacjami). Przykładowe dane wejściowe można znaleźć tu i tutaj.

   Out: Na standardowym wyjściu znaleźć się powinna jedynie odległość pomiędzy M a M'. Na standardowym wyjściu błędów powinna być jedynie macierz M', przedstawiona jako n wierszy, w każdym dokładnie m liczb całkowitych z określonego przedziału, oddzielonych spacją.

   Uwaga: Łatwo zauważyć, iż zadanie jest pewnym uproszczeniem przybliżania obrazu z ograniczeniami, natomiast nie jest wymagany żaden sposób wizualizacji wyniku za pomocą bitmap.

3. Napisz program, który będzie symulował poruszanie się agenta po kracie (wycinek $\mathbb{Z}^2$). Celem jest dotarcie agenta do wyznaczonego punktu, przy założeniach, że w każdym kroku może poruszyć się o 1 w lewo, o 1 w prawo, o 1 w górę albo o 1 w dół. Program powinien za pomocą symulowanego wyżarzania generować kolejne sekwencje kroków, tak by dotarcie do celu zajęło jak najmniej rund. Jeśli agent dotrze do celu przed wykonaniem wszystkich kroków, liczymy liczbę wykonanych kroków, jeśli po wykonaniu całej wygenerowanej sekwencji kroków, agent nie dotarł do celu - kontynuuje z miejsca, w którym wylądował a długość wykonanej sekwencji wlicza się do bieżącego rozwiązania albo zaczyna nową iterację.

   In: Dane wejściowe składać się będą z $n+1$ linii. W pierwszej linii będą umieszczone, oddzielone spacją liczby całkowite `t`, n i m, gdzie `t` jest limitem czasu, jak w zadaniu 1, natomiast n i m oznaczają wymiary kraty (odpowiednio liczba wierszy i kolumn w labiryncie, który będzie chciał opuścić agent). W kolejnych n liniach będą znajdowały się cyfry tworzące labirynt. Między cyframi **nie będzie** spacji. Możliwe cyfry:

   - **0** – standardowe, puste pole, po którym agent może się poruszać
   - **1** – ściana, która nie może zostać pokonana (Uwaga: ściany mogą znajdować się również wewnątrz labiryntu, **nie ma** wymagania by przynajmniej jednym sąsiadem `1` była `1`; ściany mogą całkowicie blokować dostęp do części labiryntu, pod warunkiem, że istnieje ścieżka z pozycji startowej agenta do przynajmniej jednego wyjścia.).
   - **5** – symbol agenta, oznaczający jego pozycję początkową (Uwaga: nie ma konieczności wizualizacji kolejnych kroków).
   - **8** – symbol wyjścia, oznaczający pozycję celu, na który agent powinien dotrzeć (Uwaga: w danej instancji może być więcej niż jeden symbol `8`, wszystkie `8` znajdują się on na obrzeżu).

   Uwaga: Agent nie zna swojej pozycji początkowej, ani pozycji celu. Można założyć, że agent potrafi rozpoznać rodzaj pola (cyfrę) swoich czterech sąsiadów oraz, że zna n oraz m.
   Przykładowe dane wejściowe można znaleźć tu, tutaj i tutaj. Przykłady z listy pierwszej są również prawidłowymi przykładami dla listy 2.

   Out: Na standardowym wyjściu znaleźć się powinna jedynie łączna liczba kroków k od pozycji startowej do celu, wykonana w wybranym rozwiązaniu. Ostatnią linią na standardowym wyjściu błędów powinien być k-elementowy ciąg znaków `U`, `D`, `R`, `L` oznaczający kolejne wybrane kierunki w rozwiązaniu, gdzie litery kodują odpowiednio krok w górę, krok w dół, krok w prawo i krok w lewo.

Uwaga: Zadanie 3. będzie rozbudowywane na liście 3, rozbudowana wersja zadania będzie również celem projektu.