Dokumentacja cząstkowa

Adrian Nadulny
Tomasz Sałacki
repozytorium: https://github.com/ANadulny/WMH

1. Cel projektu:

   Celem projektu jest zaprojektowanie i zaimplementowanie heurystycznego algorytmu przeszukiwania z tabu przy pomocy, którego uzyskamy rozwiązanie układanki Sudoku.

2. Problem:

   Rozwiązanie układanki sudoku, przy możliwych różnych początkowych stopniach trudności (przy założeniu, że układanka jest rozwiązywalna)
   Dobranie odpowiednich wartości parametrów wielkości pamięci (długo i krótkoterminowej) żeby uzyskać satysfakcjonujące rozwiązania.
   Uwzględnienie kryterium aspiracji - wyjątek od regułym, gdzie wykonujemy ruch pomimo, iż jest on na liście tabu.

3. Definicje
   a. komórka - pusta lub zawierająca cyfrę od 1 do 9
   b. satysfakcjonujące rozwiązanie - rozwiązanie, które nie posiada żadnych konfliktów
   c. tabu - jest to pamięć krótkoterminowa zakazanych ruchów. Wszystko co znajduje się na niej nie powinno być brane podczas generowania kolejnego rozwiązania w iteracjach algorytmu. Zapobiega powrotowi do ostatnio odwiedzonych rozwiązań.
   d. pamięć długoterminowa - lista zliczająca jakie stany i ile razy je odwiedzono w ciągu ostatnich n iteracji. Prowadzi przeszukiwanie w pożądany kierunek poprzez wskazywanie, że brane pod uwagę rozwiązanie było już wielokrotnie analizowane w przeszłości, co zapobiega nawrotom do tego samego miejsca.
   e. kryterium aspiracji - wyjątek w przypadku którego ruch zabroniony, który daje znacznie lepsze rozwiązanie od tego co oferuje nam sąsiedztwo danej iteracji będzie dopuszczony do realizacji.
   f. blok (w literaturze możliwy termin podkwadrat) - segment (3x3) komórek wypełniony unikalnymi cyframi
   g. ruch - jest zamianą miejscami komórek z liczbami w obrębie tego samego bloku pod warunkiem, że żadna z tych liczb nie jest liczbą umieszczoną oryginalnie na planszy w danych wejściowych.
   h. konflikt - powtarzające się cyfry w tym samym wierszu, kolumnie lub bloku
   i. kryterium stopu - otrzymanie stanu, gdzie nie występują konflikty lub osiągnięcie maksymalnej ilości iteracji

4. Metoda rozwiązania problemu:

   Algorytm przeszukiwania tabu z zaimplementowaniem dwóch list:
   i. lista tabu - pamięć krótkoterminowa, wszystko co znajduje się na niej nie powinno być brane podczas generowania kolejnego rozwiązania w iteracjach algorytmu

ii. pamięć długoterminowa - lista zliczająca ile, jakich ruchów wykonano w ciągu ostatnich n iteracji

b. Szczegółowy opis algorytmu:
   i. Wczytanie danych o stanie początkowym z pliku i zamienienie pustych miejsc na zera
   ii. Utworzenie pustej listy tabu i pamięci długoterminowej
   iii. Wypełnienie zer dowolną dostępną cyfrą w obrębie każdego bloku, tak żeby cyfry nie powtarzały się i występowały od 1 do 9
   iv. Sprawdzenie kryterium stopu jeśli nie spełnione to wykonujemy poniższe kroki
   v. Wygenerowanie wszystkich możliwych ruchów i stanów do których prowadzą należących do sąsiedztwa stanu obecnego.
   vi. Znalezienie stanu w sąsiedztwie, który jest najbardziej obiecujący z racji na minimalizację funkcji celu
   vii. Stany w sąsiedztwie rozpatrywane są w kolejności według wzrostu wartości funkcji celu. Aby stan został wybrany muszą być spełnione następujące warunki
      1. Znajduje się w tabu, ale spełnione jest kryterium aspiracji
      2. Nie znajduje się w tabu (w tym przypadku, zostanie dodany do tabu)
      3. Częstotliwość jego występowania w pamięci długoterminowej jest mniejsza od zadanej wartości

      Wybrany stan jest stanem dla którego będzie generowane sąsiedztwo w następnej iteracji
   viii. Powrót do punktu IV

c. Testowanie algorytmu i sposób nanoszenia zmiany dla parametrów wpływających na osiągany wynik
   i. sprawdzenie czasu i ilości iteracji wykonywania algorytmu dla różnych parametrów wejściowych tj. długości listy tabu oraz pamięci długoterminowej
   ii. dobranie parametrów wielkości list pamięci odbywałoby się na podstawie analizy rezultatów z wielu przebiegów dla danych parametrów początkowych

5. Środowisko:

   Java

6. Założenia implementacyjne:

   struktury danych
   i. klasy
      1. **Board** - obsługuje planszę sudoku (tablica 2d komórek)
      2. **Cell** - komórka planszy (informacje o wartości i czy jest to wartość początkowa)
      3. **CellsList** - lista komórek reprezentująca wiersz lub kolumnę
      4. **Position** - współrzędne xy komórki
      5. **Movement** - para pozycji, które mają zostać zamienione miejscami
      6. **TabuList** - przechowuje kolejkę FIFO par stan i ruch dla ruchów zakazanych do wykonania

7. **MemoryList** - przechowywanie listy odwiedzonych stanów wraz z częstotliwością ich odwiedzania w trakcie działania algorytmu
8. **SudokuSolver** - klasa, która zajmuje się wykonywaniem algorytmu znajdowania rozwiązania
9. **SudokuReader** - wczytywanie sudoku ze wskazanych plików wejściowych

ii. główne metody
1. **solveSudoku** - odpowiedzialna za zwrócenie rozwiązania łamigłówki sudoku
2. **calculateNumberOfConflictedPosition -** wylicza ile występuje konfliktów na planszy
3. **generateAllMovements** - generuje wszystkie możliwe (czyli dozwolone) ruchy dla danego stanu
4. **fillZeroesWithNumbers -** wypełnienie zerami brakujących komórek sudoku dla wczytywanego stanu początkowego
5. **movementListForSubgrid -** generuje ty listę ruchów dla danego bloku

iii. kryteria zatrzymania algorytmu
1. Otrzymanie wyniku 0 z metody calculateNumberOfConflictedPosition, która wyznacza funkcję celu. Osiągnięcie takiego wyniku, oznacza, że nie występuje żaden konflikt na planszy.

iv. sytuacje wyjątkowe wraz ze sposobem ich obsługi
1. stan do którego chcemy przenieść się w kolejnej iteracji zbyt często pojawia się w pamięci długoterminowej i przekroczona jest dla niego ustalona maksymalna częstotliwość. Jeżeli zajdzie taka sytuacja, to pomijamy taki stan i szukamy kolejnego równie dobrego, dla którego nie zajdzie opisana powyżej sytuacja

v. dane wejściowe
1. plansza sudoku zależna od poziomu trudności

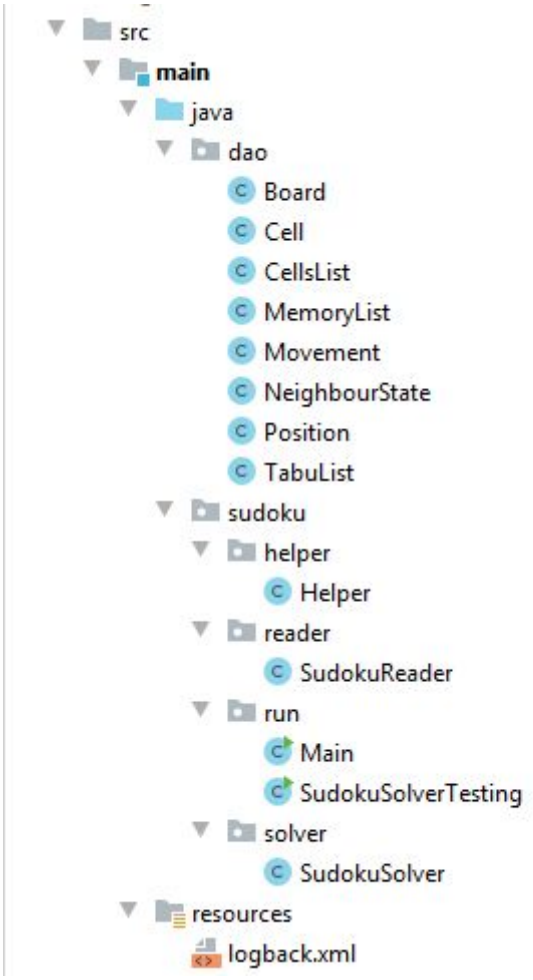
vi. parametry algorytmu
1. maksymalna liczba iteracji
2. wielkość tabu
3. wielkość pamięci długoterminowej
4. kryterium aspiracji - minimalna różnica między tabu, a najlepszym stanem z sądziedztwa żeby kryterium aspiracji zostało spełnione
5. maksymalna częstotliwość stanu w pamięci długoterminowej zezwalająca na przejście do danego stanu

7. Faza implementacji:
a. zrealizowano już
i. przygotowano w plikach .txt po 10 stanów początkowych dla 3 różnych poziomów trudności wraz z rozwiązaniami do każdego z tych stanów. Dane zostały wzięte z gazety zawierającej łamigłówki Sudoku.
ii. utworzono strukturę do przechowywania sudoku wraz możliwością wczytania stanu początkowego do niej

iii. przygotowano testy wczytujące stany początkowe i sprawdzające czy wczytywanie danych do struktury dobrze działa oraz czy pokrywają się cyfry w stanie początkowym z cyframi odpowiadającymi im w rozwiązaniach
iv. po załadowaniu stanu początkowego, puste pola są wypełniane losowymi liczbami z zachowaniem zasady, że w danym bloku żadna z liczb się nie powtarza
v. generowanie wszystkich możliwych ruchów dla danego stanu, stanów które powstaną po ich wykonaniu oraz wyliczenie jak wiele konfliktów w nich występuje

b. aktualna struktura kodu wraz z jego działaniem

i.



ii. działa już:
1. wczytywanie sudoku z plików wejściowych
2. wygenerowanie stanu początkowego z danych wejściowych
3. znajdowanie listy sąsiedztwa wraz z wartością funkcji celu dla danego stanu
4. logowanie kolejnych kroków iteracji algorytmu
5. obsługa kolejnych iteracji z uwzględnieniem listy tabu
6. algorytm jest w stanie na chwilę obecną znaleźć rozwiązanie

c. dalsze plany realizacji projektu
i. Zaimplementowanie pamięci długoterminowej
ii. Testowanie poprawności algorytmu oraz zmierzenie jego skuteczności (ilość iteracji, czas działania programu) dla różnych poziomów trudności