Seminarium: Algorytmy heurystyczne
IIUWr 2010/2011

Przemysław Gospodarczyk

Przeszukiwanie z tabu (06.04.11)

opracowanie

Wersja	Zmiany	Autor	Data
1.0	Pierwsza wersja	Przemysław Gospodarczyk	2011-04-05

Spis treści

1. Teoria **3**
1.1. Historia Tabu Search i nowe trendy . . . 3
1.2. Definicja Tabu Search i ogólna idea . . . 3
1.3. Tabu Search – schemat ogólny algorytmu . . . 4
1.4. Struktury sąsiedztwa . . . 4
1.5. Kryterium aspiracji . . . 5
1.6. Wykorzystanie pamięci długoterminowej, pojęcia dywersyfikacji i intensyfikacji 5

2. Probabilistic Tabu Search **5**

3. Reactive Tabu Search **6**

4. Tabu Cycle Method **6**

5. Iterated Tabu Search **7**
5.1. Algorytm . . . 7
5.2. Zapamiętywanie rozwiązań . . . 7
5.3. Wybór kandydatów do zoptymalizowania . . . 8
5.4. Rekonstrukcja rozwiązania . . . 8

6. Opis problemu QAP **8**
6.1. Opis problemu i specyfikacja . . . 8
6.2. Źródło danych . . . 9

7. Zastosowanie połączenia Tabu Search z algorytmem ewolucyjnym do rozwiązania problemu QAP **9**
7.1. Algorytm . . . 9
7.1.1. Schemat ogólny zastosowanego algorytmu . . . 9
7.1.2. Inicjowanie populacji . . . 10
7.1.3. Tabu Search jako heurystyka usprawniająca . . . 11
7.1.4. Wybór rodziców i generowanie nowego zbioru rozwiązań . . . 11
7.1.5. Zamiana pokolenia . . . 11
7.1.6. Własne usprawnienia . . . 11
7.2. Obsługa programu . . . 12
7.2.1. Kompilacja i uruchamianie . . . 12
7.2.2. Wywołanie i parametry algorytmów . . . 12
7.3. Testy . . . 12
7.3.1. Oznaczenia . . . 13
7.3.2. Komentarz do testów . . . 13

1. Teoria

1.1. Historia Tabu Search i nowe trendy

Fred Glover w roku 1977 przedstawił pracę dotyczącą m.in. wykorzystania pamięci krótkotrwałej i długotrwałej w przeszukiwaniu lokalnym. Pamięć długotrwała miała służyć do zapamiętania najbardziej atrakcyjnych elementów przestrzeni przeszukiwań podczas, gdy pamięć krótkotrwała służyła do pamiętania ostatnich ruchów i miała ulegać nadpisywaniu w kolejnych iteracjach przeszukiwania.
Wyżej wymienione pomysły stały się podstawą do działania Tabu Search i w roku 1986 Fred Glover wydał pierwszą pracę dotyczącą tej heurystyki.
Co ciekawe Michael Hansen, również w 1986 roku wydał swoją pracę na temat podobnej heurystyki. Heurystyka o nazwie Steepest Ascent Mildest Descent Heuristic powstała niezależnie od pracy Freda Glovera.
W ciągu lat powstało bardzo wiele różnych usprawnień, o które można wzbogacić Tabu Search. Niektóre z nich opisałem w tej pracy. Ostatnie trendy to hybrydyzacja algorytmów heurystycznych, która zazwyczaj przejawia się dodawaniem Tabu Search do algorytmów ewolucyjnych, które rozwijają się bardzo sprawnie i dają coraz lepsze rezultaty. Szczególnie techniki generujące już w pierwszych iteracjach informacje specyficzne o problemie i wykorzystujące je w praktyce (lepsze niż losowe rozwiązanie startowe, krzyżowanie fragmentów najlepszych rozwiązań znalezionych przez TS, wiedząc, że pewne pozycje w wektorze rozwiązań powinny być wypełnione w określony sposób lub zależą od siebie) są obecnie najbardziej wykorzystywane. Nie bez znaczenia dla rozwoju Tabu Search jest również możliwość zrównoleglania (każdy procesor niezależnie może modyfikować dane mu rozwiązanie).

1.2. Definicja Tabu Search i ogólna idea

Przeszukiwanie z tabu jest **metahuerystyką** do rozwiązywania problemów optymalizacyjnych, opartą na iteracyjnym przeszukiwaniu przestrzeni rozwiązań, wykorzystując sąsiedztwo pewnych elementów tej przestrzeni oraz zapamiętując przy tym przeszukiwaniu ostatnie ruchy, dopóki nie zostanie spełniony warunek końcowy. Ruchy zapisywane są w postaci **atrybutów przejścia** (parametry opisujące jednoznacznie wykonany ruch) na liście (zwanej też czasami zbiorem) tabu. Obecność danego ruchu na **liście tabu** jest tymczasowa (zazwyczaj na określoną liczbę iteracji od ostatniego użycia) i oznacza, że danego ruchu nie można wykonać przez określoną liczbę iteracji – chyba, że ruch spełnia tzw. **kryterium aspiracji** (ang. *aspiration criterion*). Lista tabu ma za zadanie wykluczyć (w praktyce jednak jedynie zmniejszyć) prawdopodobieństwo zapętleń przy przeszukiwaniu i zmusić algorytm do przeszukiwania nowych, niezbadanych regionów przestrzeni przeszukiwań.
Przeszukiwanie z tabu zmniejsza nieco szybkość zbieżności poprzez zamienianie danego rozwiązania z najlepszym jego sąsiadem, bez względu na to, czy sąsiad jest od niego lepszy, czy gorszy. Pozytywnym aspektem tego podejścia jest jednak możliwość szybszego wyjścia z regionu przyciągania ekstremum lokalnego, które niekoniecznie musi być globalnie optymalne (w szczególności, może być czasami bardzo odległe od optymalnego).
Zazwyczaj warunkiem końcowym przeszukiwania jest zadana z góry liczba iteracji do wyko-

nania, brak zmiany globalnie najlepszego rozwiązania w określonej liczbie iteracji z rzędu (co może być również motywacją do wygenerowania nowego rozwiązania startowego i rozpoczęcia algorytmu od początku z pewną wiedzą nabytą w poprzednim wykonaniu) lub jeżeli osiągniemy zadaną z góry wartość funkcji celu.
Pojęcie metaheurystyki zostało wymyślone przez Freda Glovera w roku 1986 i oznacza heurystykę nadrzędną (co sugeruje podrzędność innych w stosunku do tej), która dzięki swojej ogólności potrafi sterować innymi, odpowiednio dopasowanymi na potrzeby danego problemu optymalizacyjnego. Tabu Search jest więc ideą, a nie konkretnym algorytmem rozwiązującym konkretny problem optymalizacji.

1.3. Tabu Search – schemat ogólny algorytmu

```
//losowe rozwiązanie startowe
s = best = RANDOM_SOLUTION()
//inicjowanie listy tabu
t = []
//pętla trwa dopóki nie zostaną spełnione określone warunki końcowe
WHILE NOT(TERMINATION_CONDITION)
{
   //wykonanie kroku
   s = SELECT(NEIGHBORS(s),t))
   //aktualizowanie listy tabu
   t = UPDATE_TABU(s,t)
   //najlepsze rozwiązanie zapamiętujemy
   if (f(best) < f(s))
   {
           best = s
   }
}
return best
```

1.4. Struktury sąsiedztwa

Ze względu na przestrzeń rozwiązań należy zdefiniować relację sąsiedztwa na parach elementów tej przestrzeni, która obejmuje całą dziedzinę przestrzeni przeszukiwań. Definicja relacji zależy oczywiście od przestrzeni i formatu rozwiązań (np. rozwiązaniami mogą być wektory binarne, wektory liczb rzeczywistych, permutacje zbioru liczb naturalnych itp...).
Przykładowo, jeżeli rozważamy przestrzeń permutacji zbioru n elementowego to możemy rozważyć 3 możliwe strategie przejścia dla permutacji $\pi =< \pi(0), \pi(1), ..., \pi(n-1) >$:

- insert(i,j) – przeniesienie j-tego elementu na pozycję i-tą,
- swap(i,j) – zamiana miejscami i-tego elementu z j-tym,

- invert(i,j) – odwrócenie kolejności w podciągu zaczynającym się na i-tej pozycji i kończącym się na pozycji j-tej.

W powyższych przejściach argumenty i,j nazywamy atrybutami przejścia, które jednoznacznie pozwalają identyfikować przejście.

1.5. Kryterium aspiracji

Czasami zabranianie pewnych ruchów, mimo, że nie prowadzi do zapętleń, to może spowodować ogólną stagnację przy przeszukiwaniu lub wręcz całkowicie uniemożliwić ruch (sytuacja, gdy wszyscy sąsiedzi są na liście tabu).
Spełnienie kryterium aspiracji pozwala na złamanie tabu, czyli wykonanie ruchu pomimo, że jego atrybuty znajdują się na liście ruchów zabronionych. Najprostszym i najczęściej stosowanym kryterium aspiracji jest zezwolenie na wykonanie ruchu, jeżeli prowadzi on bezpośrednio do uzyskania najlepszego znanego globalnie rozwiązania. Inne ciekawsze, ale znacznie bardziej skomplikowane kryteria aspiracji opierają się na wyspecjalizowanych metodach badających możliwości powstania cyklu po wykonaniu danego ruchu (co udało się efektywnie zaimplementować naukowcom w latach 1989, 1991).

1.6. Wykorzystanie pamięci długoterminowej, pojęcia dywersyfikacji i intensyfikacji

Podstawowym pomysłem strategii Tabu Search jest wykorzystanie pamięci długoterminowej do przechowywania najlepszego globalnie rozwiązania, które na koniec algorytmu zostaje zwrócone jako wynik działania heurystyki.
Istnieje wiele innych pomysłów na wykorzystanie pamięci długoterminowej przy przeszukiwaniu przestrzeni rozwiązań, których cechą wspólną jest chęć utworzenia statystyki o korzystnych ruchach z przeszłych iteracji. Statystykę można wykorzystać do bardziej inteligentnych i wyspecjalizowanych strategii, które pozwalają na zaspokajanie dwóch istotnych kryteriów:

- intensyfikacji, czyli zagęszczenia próbkowania obszaru, jeżeli w tym obszarze znajdują się dobre rozwiązania;

- dywersyfikacji, czyli rozproszenia przeszukiwania, aby nie pominąć wszystkich podobszarów.

Pomysłem na egzekwowanie powyższych pomysłów może być też funkcja kary, która zależy od częstotliwości wykonywania danego ruchu i zaburza wartości funkcji celu.

2. Probabilistic Tabu Search

Metoda na ogół mniej skuteczna niż klasyczny Tabu Search, ale dużo wydajniejsza i przydatna, gdy funkcja celu jest skomplikowana i liczy się długo oraz, gdy przestrzeń przeszukiwania jest duża (oraz, co za tym idzie, być może sąsiedztwo).
Probabilistic Tabu Search sprawdza tylko losowy podzbiór sąsiadów (zamiast wszystkich).

Ponadto randomizacja jest silnym mechanizmem anty-cyklowym, co pozwala skracać listę tabu.
W związku z tym, że łatwo przeoczyć w ten sposób dobre rozwiązania, mimo że są one blisko, warto co kilka iteracji przeprowadzać kilka iteracji klasycznego Tabu Search (Glover i Laguna, 1993).
Ewentualnie po zakończeniu działania Probabilistic Tabu Search, można włączyć klasyczny Tabu Search na kilka iteracji, aby zoptymalizować otrzymane rozwiązanie, tak by było przynajmniej optimum lokalnym.
Pełny opis metody można znaleźć w [5].

3. Reactive Tabu Search

Metoda pozwala dynamicznie sterować długością listy tabu w trakcie działania algorytmu za cenę pamięciożerności i zwiększonej złożoności obliczeniowej. Pamiętając wszystkie dotychczasowe rozwiązania wraz z numerem iteracji, w pamięci długotrwałej, algorytm sprawdza powtarzalność uzyskiwanych rozwiązań. Jeżeli rozwiązania zaczynają się powtarzać, to algorytm w kolejnych iteracjach wolniej redukuje listę (co powoduje zwiększenie jej długości) niedopuszczając do powstania cyklów i "uciekając" z regionu przyciągania ekstremum lokalnego.
Przypadek, gdy listę można skrócić zachodzi, gdy rozwiązania nie powtarzają się.
Aby zmniejszyć złożoność, stosuje się haszowanie i kompresję (do rozróżnienia m rozwiązań wystarczy `log(m)` bitów).
Pełny opis metody można znaleźć w [6].

4. Tabu Cycle Method

Metoda dzieli w trakcie działania listę tabu na grupy, gdzie k-ta grupa zawiera elementy dodane w określonym przedziale iteracji.
Ruchy w grupie najmłodszej nr. 0, zwanej *Buffer Group* są zabronione na stałe. Algorytm pozwala na łamanie tabu dla starszych grup, z częstotliwością zależną od współczynnika $TC(k) = n - k + 1$, gdzie k jest numerem grupy, a n liczbą grup. Współczynnik $TC(k)$ mówi o tym, co ile iteracji wolno łamać tabu w grupie k-tej.
Ponadto wprowadza się współczynnik $CC(k)$ inkrementowany o 1 w każdej iteracji, w której nie następuje łamanie tabu w grupie k-tej. Jeżeli nastąpiło złamanie tabu, to $\forall_{h \geqslant k} CC(h) = CC(h)–TC(h)$. W zależności od wyżej wymienionych współczynników, poszczególnym grupom nadawane są następujące statusy:

- OFF — grupie nigdy nie wolno łamać tabu, przypadek dla $CC(k) < TC(K)$;
- ON – grupie wolno łamać tabu co $TC(K)$ iteracji, przypadek $CC(k) \geqslant TC(K)$;
- FREE — na grupę nie jest nałożone żadne tabu, przypadek, gdy grupa jest ON i każda starsza od niej też jest ON.

Jeżeli dozwolony ruch dla grupy k-tej nie jest wykonany to: $CC(k) = min(CC(k)+1, aTC(k))$, gdzie a jest odpowiednio dobranym współczynnikiem, który pilnuje jak dużą wielokrotność TC można osiągnąć (czasami $CC(k)$ staje się tak duży, że grupy są ON lub wręcz FREE zbyt długo). Pełny opis metody można znaleźć w pracy autorstwa Glovera i Laguny z 1997 roku.

5. Iterated Tabu Search

5.1. Algorytm

Uproszczony schemat ogólny algorytmu w pseudokodzie (w szczególności pewne metody nie zostały jawnie zdefiniowane i algorytm należy samodzielnie dospecyfikować, ze względu na specyfikę danego problemu):

```
//losowe rozwiązanie startowe
s = RANDOM_SOLUTION()
//wywołanie klasycznego Tabu Search na rozwiązaniu startowym
s' = TABU_SEARCH(s)
//zapamiętujemy najlepsze dotychczasowe rozwiązanie jako s*
s* = s'
//pętla trwa dopóki nie zostaną spełnione określone warunki końcowe
WHILE NOT(TERMINATION_CONDITION)
{
   //wybieranie rozwiązania do zoptymalizowania
   s = CANDIDATE_ACCEPTANCE(s', s'', s*, s, ...)
   //rekonstrukcja wybranego rozwiązania
   s' = RECONSTRUCTION(s)
   //wywołanie klasycznego Tabu Search na wybranym rozwiązaniu
   s'' = TABU_SEARCH(s')
   //najlepsze rozwiązanie zapamiętujemy
   if (f(s*) < f(s''))
   {
           s* = s''
   }
}
return s*
```

Oryginalna wersja omówionego w tej pracy algorytmu pochodzi z pracy [8].

5.2. Zapamiętywanie rozwiązań

Algorytm ITS zawsze zapamiętuje najlepsze znane do tej pory rozwiązanie jako `s*`. Reszta rozwiązań, która okazuje się słabsza, a w szczególności pewne rozwiązania pośrednie, również mogą być warte zapamiętywania (w zależności od strategii wyboru kandydatów do zoptymalizowania), ale trzymanie tych rozwiązań nie jest obowiązkowe.

5.3. Wybór kandydatów do zoptymalizowania

Wybór, którego należy dokonać to pewien kompromis między intensyfikacją a dywersyfikacją przeszukiwania.
Najprostszy pomysł polega na wyborze najlepszego znanego rozwiązania (`s*` w schemacie algorytmu) i może prowadzić (szczególnie przy słabszej metodzie rekonstrukcji) do intensyfikacji. Pomysł może sprzyjać również szybszej zbieżność.
Inne pomysły, które wymagają pamiętania rozwiązań pośrednich, jak np. losowe wybieranie może prowadzić do dywersyfikacji.
Często stosuje się też metodę WYA (skrót od ang. „where are you") polega na wyborze kolejnych optimów z poprzedniej iteracji, bez względu na ich jakość z globalnego punktu widzenia.
Wszystkie „inteligentniejsze" metody wyboru rozwiązań wymagają znacznie bardziej wyspecjalizowanych metod, które dotyczą bardziej zaawansowanych algorytmów ewolucyjnych np. BOA (ang. *Bayesian optimization algorithm*), ECGA (ang. *Extended Compact Genetic Algorithm*), PBIL (ang. *Population Based Incremental Learning*). Pierwsze dwa dają już po kilku iteracjach pewną specyficzną wiedzę na temat zadania, pozwalają zlokalizować zależności między genami i wykorzystać uzyskaną wiedzę do dokonywania bardziej świadomych wyborów. Omawianie tych algorytmów przekracza znacznie temat tej pracy.

5.4. Rekonstrukcja rozwiązania

Wybór metody rekonstrukcji rozwiązania to kolejny kompromis między intensyfikacją a dywersyfikacją przeszukiwania.
Z jednej strony rekonstrukcja musi być na tyle silna, aby umożliwić skuteczną "ucieczkę" z regionu przyciągania ekstremum lokalnego. Z drugiej strony zależy nam na tym, aby nie tracić wiedzy o dobrych rozwiązaniach (jeżeli zaburzymy rozwiązanie zbyt mocno to algorytm stanie się zbyt losowy).

6. Opis problemu QAP

6.1. Opis problemu i specyfikacja

Kwadratowy problem przydziału (ang. *Quadratic assignment problem – QAP*) to jeden z najważniejszych NP-trudnych problemów optymalizacji kombinatorycznej. Przedstawiona poniżej interpretacja tego problemu jest jedną z wielu znanych.

Dysponujemy zbiorami `N` fabryk i `N` lokacji oraz dwoma kwadratowymi macierzami rozmiaru `N` (macierz `d` odległości między każdą parą lokacji i macierz `f` wag między każdą parą fabryk), gdzie `N` jest parametrem algorytmu (liczbą naturalną).

Zadanie polega na przypisaniu każdej z fabryk do jednej z lokacji (każdą do innej) tak, aby zminimalizować sumę iloczynów dystansów między lokacjami i odpowiadających wag. W praktyce chodzi o minimalizację pewnej funkcji celu:

$$\min_{p \in \Pi} \sum_{i=1}^{N} \sum_{j=1}^{N} d[i][j] * f[p[i]][p[j]] \tag{1}$$

gdzie `p` jest wektorem przypisania.
Wektor przypisania jest w rzeczywistości permutacją liczb naturalnych z zakresu [0,N), gdzie na i-tym polu odpowiadającym i-tej lokacji znajduje się numer fabryki, która jest przypisana do tej lokacji.

6.2. Źródło danych

Wszystkie zestawy danych testowych w postaci plików o rozszerzeniu `.DAT` pochodzą z serwisu QAPLIB - A Quadratic Assignment Problem Library (adres [2]). Struktura pliku z danymi składa się z trzech części:

- liczba naturalna `N`;
- poprzedzona pustym wierszem zawartość macierzy `f` (w każdej z N linijek jeden wiersz macierzy);
- poprzedzona pustym wierszem zawartość macierzy `d` (w każdej z N linijek jeden wiersz macierzy).

Serwis udostępnia ponadto najlepsze znane rozwiązania dla wszystkich udostępnionych testów.

7. Zastosowanie połączenia Tabu Search z algorytmem ewolucyjnym do rozwiązania problemu QAP

7.1. Algorytm

Oryginalna wersja omówionego w tej pracy algorytmu pochodzi z pracy [1], wersja opisana w tej pracy została zaimplementowana na nowo, interpretując i rozszerzając pomysły z wersji oryginalnej.

7.1.1. Schemat ogólny zastosowanego algorytmu

Uproszczony schemat ogólny algorytmu w pseudokodzie (w szczególności, w kodzie źródłowym nazwy poszczególnych metod i ich parametry są inne):

```
//czyta dane z pliku
READ_DATA(file)
//inicjuje obie populacje (zróżnicowaną i najlepszą)
INIT_BOTH_POPULATIONS(DIV_POP, BEST_POP)
//heurystyka Tabu Search
```

```
BEST_POP_UPDATE(BEST_POP)
//warunkiem końcowym jest osiągnięcie zadanej liczby iteracji
FOR (i = 0; i < MAX-ITER; i++)
{
   //jeżeli w poprzedniej iteracji nie dodano żadnego nowego rozwiązania
   if (!added)
   {
           //inicjujemy na nowo zróżnicowaną populację
           INIT_DIV_POPULATION(DIV_POP)
   }
   // krzyżowanie osobników (krok zrównoleglony, osobno dla każdej pary)
   CHILDREN = CROSSOVER(DIV_POP, BEST_POP)
   // podmiana starego pokolenia na nowe
   (DIV_POP, BEST_POP) = REPLACEMENT(CHILDREN, DIV_POP, BEST_POP)
   //heurystyka Tabu Search
   BEST_POP_UPDATE(BEST_POP)
}
```

7.1.2. Inicjowanie populacji

Generujemy startowy zbiór rozwiązań wykorzystując wiedzę o typie problemu. Pewna z góry ustalona liczba mówi nam ile chcemy rozwiązań najlepszej jakości, a ile najbardziej różnorodnych (one rozszerzają nam przestrzeń poszukiwań i zmniejszają prawdopodobieństwo zbieżności do ekstremum lokalnego). Można stosować wyspecjalizowane heurystyki, ulepszające zbiór startowy.

Wykorzystano tzw. metodę Glovera, która gwarantuje pewien poziom zróżnicowania w zbiorze rozwiązań (w przeciwieństwie do podejścia czysto losowego). Wygenerowane przez metodę wektory są poprawne i nie trzeba ich naprawiać.
Jako ziarno wykorzystujemy 1 losowy, poprawny wektor x. Na podstawie ziarna tworzymy nowe rozwiązanie.
Dane:

- b – naturalne, mniejsze od N;

- s – pozycja startowa, na początku równa b.

Algorytm:
Rozwiązanie x' budujemy dodając najpierw element `x[s]`, potem kolejno `x[s + b], x[s + 2b], ..., x[s + rb]`, gdzie r jest tak dobrane, że `s+rb` nie przekracza N. Kiedy dojdziemy do końca ziarna, to pozycja startowa zmniejsza się o 1 i cała operacja się powtarza aż nie zapełnimy całego wektora x' (musimy odwiedzić wszystkie pozycje w x).

7.1.3. Tabu Search jako heurystyka usprawniająca

Dla najlepszej populacji wygenerowanych osobników stosujemy metodę przeszukiwania tabu Erica Taillarda dla QAP.
Heurystyka oblicza możliwe zamiany lokacji 2 firm, wykorzystując prostą listę tabu (ruchy niedozwolone). Aby zmusić algorytm do przeszukiwania niezbadanych fragmentów przestrzeni, wymieniane są ze sobą tylko te elementy, które nie były wymieniane przez kilka ostatnich iteracji (na podstawie listy tabu). Jeżeli jednak, zabroniona zmiana doprowadza do wyprodukowania najlepszego nieznanego do tej pory przez algorytm rozwiązania, to zamiana się dokonuje, mimo obecności na liście tabu (ang. *aspiration criterion*).

7.1.4. Wybór rodziców i generowanie nowego zbioru rozwiązań

Wszystkie rozwiązania ze zbioru łączymy w pary (metoda każdy z każdym). Stosujemy metodę path-relinking.
Wektorami wiodącymi, zostają te, które są lepsze w swojej parze (lepiej optymalizują f. celu), potomek jest inicjowany wektorem niewiodącym. Iterujemy wektory. Jeżeli na napotkanej pozycji wektory rodziców nie różnią się, to odpowiadająca pozycja w potomku jest pozostawiana. Jeżeli na napotkanej pozycji wektory rodziców różnią się oraz pozycja ta w potomku nie była wcześniej zmieniana to do potomka przepisywana jest wartość z wektora wiodącego (pod warunkiem, że nie zwiększa wartości funkcji celu). Jeżeli pozycja była już rozważana to nie ulega zmianie (algorytm nigdy nie robi zamiany wstecz).

7.1.5. Zamiana pokolenia

Niech R1 to zbiór najlepszych rozwiązań oraz niech R2 to zbiór najbardziej zróżnicowanych rozwiązań.
Nowe rozwiązanie wchodzi do R1, jeżeli jest lepsze niż najgorsze z tego zbioru i nie ma jego kopii ani w R1 ani w R2. Nowe rozwiązanie wchodzi do R2, jeżeli jest bardziej zróżnicowane niż najmniej zróżnicowane z tego zbioru i nie ma jego kopii ani w R1 ani w R2.
Różnorodność rozwiązania x jest zdefiniowana jako dystans między tym rozwiązaniem, a wszystkimi rozwiązaniami z R1.

$$\forall_{x^1 \in R_1} d(x, x^1) = \sum_{i=1}^{N} |x - x^1| \tag{2}$$

$$D(x, R_1) = \sum_{x^1 \in R_1} d(x, x^1) \tag{3}$$

7.1.6. Własne usprawnienia

Populacja zróżnicowana jest inicjowana na nowo, jeżeli w ciągu pięciu kolejnych iteracji nie znaleziono nowego minimum.
Różnorodność rozwiązania x jest zdefiniowana jako dystans między tym rozwiązaniem, a

wszystkimi rozwiązaniami **z R1 oraz z R2**. Zapobiega to zbieżności populacji zróżnicowanej do określonych wartości, bo nowe przedziały przeszukiwania rozwiązania zależą teraz także od populacji najlepszej.

7.2. Obsługa programu

7.2.1. Kompilacja i uruchamianie

Program został napisany w całości w języku `C++` przy użyciu zestawu narzędzi programistycznych Dev-C++ w wersji 4.9.9.2.
Nie gwarantuję, że przy innej wersji zintegrowanego środowiska programistycznego, a w szczególności innej wersji kompilatora języka `C++` program będzie działał poprawnie. Aplikacja była testowana pod systemem Windows XP Professional z zainstalowanym dodatkiem Service Pack 2 i jest przeznaczona wyłącznie dla systemów operacyjnych Microsoft Windows.
Kod źródłowy implementacji znajduje się w całości w pliku `main.cpp`.

7.2.2. Wywołanie i parametry algorytmów

Przed uruchomieniem algorytmu można ustawić następujące parametry (jako stałe zdefiniowane na początku kodu źródłowego):

- wielkość populacji zróżnicowanej i najlepszej (stała `P`, domyślnie `P = 50`) – łączna wielkość populacji wynosi więc `2P`;
- liczba iteracji (stała `MAX_ITER`, domyślnie `MAX_ITER = 500`).

Pozostałe parametry dostarczane są wraz z plikiem z danymi, jego struktura została omówiona w jednym z poprzednich działów tej pracy.
Przykładowe wywołanie skompilowanego programu:
`Projekt.exe < plik.dat`, gdzie `plik.dat` to plik z danymi o odpowiedniej strukturze.

7.3. Testy

Wyniki algorytmu							
Test	Rozmiar	MAX_ITER	MIN_ITER	P	T	%best_known	%opt
Wil100	100	50	38	2*70	5	1.6%	3.4%
Inst100	100	25	12	2*100	5	0.05%	brak danych
Sko100c	100	35	22	2*150	5	2.6%	5.73%
Sko100f	100	50	28	2*150	5	1.99%	6.43%
Esc128	128	20	8	2*110	10	0%	96.86%
Tho150	150	75	57	2*110	10	1.8%	6.3%
Tai150b	150	40	17	2*100	5	1.9%	12.66%
Inst200	200	50	23	2*100	10	0.34%	brak danych
Tai256c	256	80	42	2*120	5	0.55%	2.03%

7.3.1. Oznaczenia

Test – nazwa testu.
N – rozmiar testu.
MAX_ITER – liczba wszystkich iteracji, jakie wykonał algorytm.
MIN_ITER – minimalna liczba iteracji, jakich potrzebuje algorytm do znalezienia najlepszego rozwiązania.
P – suma rozmiarów populacji.
T – liczba iteracji bez poprawy najlepszego rozwiązania, która skutkuje wygenerowaniem populacji zróżnicowanej na nowo.
%best_known – procent o jaki gorsze jest rozwiązanie algorytmu od najlepszego znanego rozwiązania.
%opt – procent o jaki gorsze jest najlepsze znane rozwiązanie od optymalnego.

7.3.2. Komentarz do testów

Badanie pokazało, że zastosowanie Tabu Search jako wspomagania innych heurystyk do rozwiązywania NP-trudnych, kombinatorycznych problemów takich jak QAP znajduje swoje uzasadnienie w praktyce. Algorytm we wszystkich opisanych testach znalazł rozwiązania niewiele gorsze od najlepszych znanych (w jednym przypadku udało się wyrównać najlepszy znany wynik). Ponadto na podstawie wyników ze strony QAP można zauważyć, że nie istnieje obecnie ani jeden algorytm heurysytyczny, który radziłby sobie wzorowo ze wszystkimi testami (najlepsze znane rezultaty są osiągane przez wiele implementacji, różnych algorytmów, nie tylko ewolucyjnych i Tabu Search).

Literatura

[1] T. James, C. Rego, F. Glover *Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem*, IEEE Computer Society 2005;

[2] *http://www.seas.upenn.edu/qaplib/inst.html (ostatni dostęp 05.04.2011)*;

[3] *http://www.soften.ktu.lt/ gintaras/qproblem.html (ostatni dostęp 05.04.2011)*;

[4] M.Laguna *Exploration of Metaheuristic Optimization: Tabu Search, Scatter Search and Path Relinking* ;

[5] F.Glover, M.Laguna *Tabu Search* ;

[6] R.Battiti, G. Tecchiolli *The Reactive Tabu Search* , ORSA Journal on Computing Vol.6, No. 2 (1994);

[7] M.Laguna *Implementing and Testing the Tabu Cycle and Conditional Probability Methods*, Leeds School of Business University of Colorado Boulder,(2005);

[8] A. Misevicius, A. Lenkevicius, D. Rubliauskas *Iterated Tabu Search: An Improvement To Standrard Tabu Search*, Information Technology and Control, 2006, Vol. 35, No. 3.