METODY SZTUCZNEJ INTELIGENCJI

Pod powyższym pojęciem będziemy rozumieli różne metody (algorytmy) z zakresu teorii gier, planowania, systemów ekspertowych, teorii algorytmów, itp. mające na celu ***symulowanie inteligentnego zachowania komputera***, zbliżającego jego sposób postępowania do ludzkiego.

Więcej informacji:

1. L. Bolc, J. Cytowski, „Metody przeszukiwania heurystycznego", PWN, 1989.
2. P. Wróblewski, „Algorytmy, struktury danych i techniki programowania", Helion.
3. N.J. Nilsson, „Principles of Artificial Intelligence", Springer-Verlag, 1982.

Gry dwuosobowe

- ***kompletna wiedza*** o sytuacji (stanie planszy) na każdym etapie gry,
- ***symetryczne*** role graczy,
- reguły gry znane ***z wyprzedzeniem***.

Strukturą danych (reprezentującą stan i przebieg gry) może być *drzewo*:

- każdy **węzeł** opisuje jeden z możliwych stanów gry,
- **krawędzie** wychodzące z węzłów **na danym poziomie** drzewa reprezentują wszystkie możliwe ruchy danego gracza (jako odpowiedzi na wszystkie możliwe ruchy przeciwnika),
- **ścieżka** w drzewie (od korzenia do jednego z liści) reprezentuje jeden z możliwych przebiegów gry.

Struktura pojedynczego węzła zależy od charakteru (stopnia skomplikowania) gry.

Rozmiar drzewa (liczba węzłów) zależy od liczby możliwych ruchów graczy (a więc pośrednio również od stopnia skomplikowania gry).

Jeśli gracz A wykona określony ruch (przejście w drzewie do określonego węzła podrzędnego), to **gracz B może wybrać jeden z ruchów** dostępnych z wybranego węzła, przy czym musi on być:

- **najkorzystniejszy dla B** (kryterium *zdrowego rozsądku*),
- **zgodny z regułami** gry (kryterium *poprawności*).

Z punktu widzenia jednego z graczy, gra może zakończyć się wygraną, przegraną lub remisem.

Zadaniem algorytmu sztucznej inteligencji jest takie przeprowadzenie gry, aby zaproponować jednemu z graczy (komputerowi) ***strategię wygrywającą***.

Zatem algorytm taki powinien zawierać **dwa typy funkcji**:

1. ***ewaluacja*** bieżącego stanu gry (pod kątem przewagi jednej ze stron) i wygenerowanie na tej podstawie liczby rzeczywistej (porównanie dwóch stanów gry sprowadza się do porównania dwóch liczb rzeczywistych),

2. ***decyzja*** o wyborze kolejnego ruchu, podejmowana na podstawie ewaluacji bieżącego stanu gry (i ewentualnie kilku stanów następnych, znanych na podstawie wygenerowania całości lub części drzewa).

Strategie przeszukiwania (wybrane)

- algorytm A*,
- mini-max,
- algorytm cięć $\alpha - \beta$,
- SSS*.

Algorytm *mini-max*

Pomaga znaleźć **najlepszy ruch** (z punktu widzenia gracza A), **analizując sytuację od końca** gry (od najniższego możliwego poziomu drzewa).

W każdym kroku zakładamy (zgodnie ze zdrowym rozsądkiem), że **gracz A** próbuje **zmaksymalizować** swoje szanse na wygraną, podczas gdy w następnym ruchu **gracz B** będzie starał się te szanse **zminimalizować** (tzn. zmaksymalizować swoje szanse wygrania).

Jeśli analiza całego drzewa nie jest możliwa (zbyt duży koszt obliczeń), algorytm bierze pod uwagę tylko pewną określoną liczbę poziomów.

Jeśli najniższy analizowany poziom dotyczy ruchów **przeciwnika** (gracza B), to wybierany jest węzeł z wartością **minimalną** (w przeciwnym wypadku ten z wartością maksymalną).

Algorytm cięć $\alpha - \beta$

Jest to rozszerzenie algorytmu *mini-max*.

Idea sprowadza się do **pomijania analizy tych węzłów**, które dostarczyłyby **gorszy ruch niż już zbadane** (odcinanie niektórych gałęzi drzewa).

W efekcie mamy algorytm o **krótszym czasie działania** niż *mini-max*.

Jego **efektywność** ściśle zależy od tego, w jakiej **kolejności** węzły będą analizowane:

- Jeśli najpierw będą analizowane „**dobre**", to potencjalnie **więcej** kolejnych węzłów zostanie **wyeliminowanych** i czas działania będzie krótszy.
- Idealnie więc by było analizować węzły w odpowiedniej kolejności, co jednak zwiększa **czasochłonność**.
- W praktyce, stosuje się różne szybkie metody **heurystyczne** (nie obciążające całej procedury czasowo, a jednak pozwalające zwiększyć liczbę wyeliminowanych węzłów).

Algorytmy metaheurystyczne

Jest to cały szereg różnych algorytmów optymalizujących zadaną funkcję celu (przy określonych ograniczeniach), bazujących na **przeszukiwaniu dostępnych rozwiązań w sposób *inteligentny***, tak aby znaleźć rozwiązanie o jak najmniejszej (największej) wartości funkcji celu.

Metody te różnią się sposobem przeszukiwania przestrzeni rozwiązań (oczywiście chodzi o to, żeby w jak najmniejszej liczbie kroków znaleźć rozwiązanie lepsze od najlepszego dotychczas znalezionego).

Sposoby przeszukiwania czerpią inspirację m.in. z:
hartowania metali, doboru naturalnego (ewolucji, genetyki), systemów immunologicznych, systemów neuronowych, zachowania się organizmów „stadnych" (roje, kolonie), zjawiska elektromagnetyzmu.

Sąsiedztwo (otoczenie) $N(\pi)$ *danego rozwiązania* π *w przestrzeni permutacji – zbiór permutacji uzyskanych z* π *poprzez wykonanie* **wszystkich możliwych ruchów** *(zaburzeń) danego typu.*

Rodzaje ruchów:

- Ruch typu **„wstaw (i, j)"** (*insert*) – wstawienie elementu z pozycji i-tej na pozycję j-tą ($i, j = 1, \ldots, n,\ i \neq j$) w danej permutacji.

- Ruch typu **„zamień (i, j)"** (*swap*) – zamiana miejscami elementu z pozycji i-tej z elementem z pozycji j-tej w danej permutacji.

- Ruch typu **„odwróć (i, j)"** (*invert*) – odwrócenie w danej permutacji ciągu elementów na pozycjach od i-tej do j-tej.

Symulowane wyżarzanie

ang. ***Simulated annealing (SA)*** – Scott Kirkpatrick (1980).

Algorytm SA nawiązuje do **termodynamicznego procesu wyżarzania metali**, w którym próbka metalu poddawana jest procesowi studzenia w celu osiągnięcia pożądanych cech (twardości, sprężystości, itp.). W procesie tym następuje minimalizacja energii materiału. Dokładniej, temperatura jest redukowana powoli w celu dojścia do stanu **równowagi energetycznej** przy zachowaniu maksymalnej entropii.

Idea działania SA sprowadza się do wygenerowania trajektorii poszukiwań $x^0, x^1, \ldots$ (gdzie każde kolejne rozwiązanie x^{j+1} jest wybierane z sąsiedztwa rozwiązania x^j) w taki sposób, aby wyjść z ekstremum (np. minimum) lokalnego lub aby go w ogóle uniknąć. W efekcie, trajektoria poszukiwań algorytmu SA jest prowadzona w „statystycznie dobrym" kierunku.

Algorytm SA rozpoczyna swoje działanie od pewnego ***rozwiązania początkowego*** x^0 (np. losowego) i sukcesywnie porusza się po rozwiązaniach sąsiednich.

W danej iteracji i, z sąsiedztwa aktualnego rozwiązania $N(x^j)$ wybierane jest ***rozwiązanie zaburzone*** x' w sposób losowy*. Niech $\Delta = K(x') - K(x^j)$.

Mogą zachodzić dwie sytuacje:

1. $\Delta \leq 0$ (czyli $K(x') \leq K(x^j)$),

2. $\Delta > 0$ (czyli $K(x') > K(x^j)$).

*W przypadku, gdy rozwiązania są reprezentowane przez **permutacje**, to rozwiązanie zaburzone może być wygenerowane przez pojedynczy ruch *wstaw(k,l)*, *zamień(k,l)*, lub *odwróć(k,l)* dla losowych wartości k oraz l.

Ad. 1. **($\Delta \leq 0$)** x' jest akceptowane jako nowe rozwiązanie bez dodatkowych warunków, tzn.

$$x^{(j+1)} = x'.$$

Ad. 2. **($\Delta > 0$)** x' jest akceptowane z prawdopodobieństwem $P = \min\{1, e^{-\Delta/T_i}\}$, gdzie $\boldsymbol{T_i}$ jest ***temperaturą*** w iteracji i, tzn.

$$x^{(j+1)} = x', \text{ jeśli } R < e^{-\Delta/T_i}$$

dla pewnej losowo wygenerowanej liczby R z przedziału $[0, 1)$.

Temperatura, T_i, jest parametrem regulujacym proces akceptacji i powinna maleć w procesie poszukiwań wg określonego ***schematu studzenia***, np:

- schemat geometryczny: $T_{i+1} = \lambda_i T_i$, $1 > \lambda_i > 0$,
- schemat logarytmiczny: $T_{i+1} = \frac{T_i}{1+\lambda_i T_i}$, $\lambda_i > 0$.

W każdej iteracji i, proces generowania (i ewentualnego akceptowania) nowego rozwiązania zaburzonego jest powtarzany s razy dla danej wartości temperatury T_i.

Parametry s, λ_i, ***temperaturę początkową*** T_0 oraz ***końcową*** T_k należy dobrać eksperymentalnie,

przy czym wartość T_k powinna być bliska 0 (np. $T_k = 0{,}0001$) oraz mniejsza od T_0.

Wartość $\boldsymbol{s}$ można przyjąć:

1 (zapewnienie szybkiego czasu działania SA),
lub n, lub n^2, lub $n^2/2$, lub $n^2/4$, itp.,

gdzie n — rozmiar instancji problemu (np. liczba zadań dla problemu szeregowania).

Dla $\lambda_\mathbf{i} = \lambda = const$ możemy wyliczyć:

- $\lambda = \sqrt[N]{\frac{T_k}{T_0}}$ dla schematu geometrycznego oraz
- $\lambda = \frac{T_0 - T_k}{N T_0 T_k}$ dla schematu logarytmicznego.

Inny sposób wyznaczania λ_i:

$$\lambda_i = \frac{\ln(1+\delta)}{3\sigma_i},$$

gdzie $\delta \in [0, 1-10, 0]$, σ_i – odchylenie standardowe wartości kryterium dla wszystkich s rozwiązań generowanych w danej temperaturze T_i, tzn.

$$\sigma_i = \sqrt{\frac{\sum_{q=1}^{s} K_q^2(x^j)}{s} - \left(\frac{\sum_{q=1}^{s} K_q(x^j)}{s}\right)^2}.$$

Uwaga!!! Należy testować, czy $\sigma_i = 0$ (oznacza to, że w danej temp. wszystkie rozwiązania są takie same).

Sposób wyznaczenia T_0:

Poczynając od x^0 wygeneruj k' kolejnych rozwiązań próbnych, z których każde jest bezwarunkowo akceptowane. Wyznacz

$$\Delta_{\max} = \max_{1 \le i \le k'} \left(K(x^{i+1}) - K(x^i) \right) \quad \text{oraz} \quad T_0 = -\Delta_{\max}/ln(p),$$

gdzie $p \approx 0,9$. Wartość k' można przyjąć równe n, n^2, itp.

Kryterium stopu:

- osiągnięcie temperatury końcowej, $T_i = T_k$,
- wykonanie określonej ilości iteracji, $i = MaxIter$,
- określony czas działania algorytmu,
- kombinacja powyższych.

Uwaga!!! Podczas procesu poszukiwań należy zapamiętywać najlepsze znalezione rozwiązanie.

Schemat metody SA

```
procedure Symulowane_wyżarzanie;
begin
    wyznacz(x^0);
    zainicjuj(T_0,s);
    i := 0;
    x := x^0;
    repeat
        for m := 1 to s do
            begin
                x' :=zaburzenie(x);
                if K(x') ≤ K(x) then x := x';
                else if random(0,1) < e^{-Δ/T_i} then x := x';
            end;
        wyznacz_temperaturę(T_i);
        i := i + 1;
    until kryterium_stopu;
end;
```

Poszukiwanie z zabronienieniami (zakazami)

ang. ***Tabu Search (TS)*** – Fred Glover (1989-90).

Algorytm TS to metoda przeszukiwania zbioru rozwiązań naśladująca **proces poszukiwania wykonywany przez człowieka**. Jej podstawowe cechy to:

- wybieranie lokalnie najlepszego rozwiązania,
- omijanie już odwiedzanych rozwiązań lub tych, które przypominają już odwiedzone rozwiązania.

Algorytm TS rozpoczyna swoje działanie od pewnego **rozwiązania początkowego** x^0 i sukcesywnie porusza się po rozwiązaniach sąsiednich, przeglądając każdorazowo całe sąsiedztwo rozwiązania bieżącego.

W danej iteracji i, z sąsiedztwa aktualnego rozwiązania $N(x^i)$ wybierane jest „niezakazane" rozwiązanie $x^{(i+1)}$ takie, że

$$K(x^{(i+1)}) = \min_{x \in N(x^i)} \{K(x)\}.$$

Wybrane rozwiązanie jest zapamiętywane i w dalszym procesie poszukiwań jest **zakazane** (ma status tabu).

Do zapamiętywania rozwiązań odwiedzonych najczęściej wykorzystuje się **pamięć krótkoterminową** zwaną listą tabu.

Na liście tabu zapamiętujemy pewną liczbę: ostatnio odwiedzonych rozwiązań, pewnych ich parametrów, lub ruchów do nich prowadzących.

Lista ta jest kolejką typu LIFO, tzn. nowododawane rozwiązanie zastępuje najstarsze.

Lista tabu często ma stałą i niewielką długość (7-10 pozycji). Choć spotyka się implementacje z listą o zmiennej długości.

Przykład

Ze względu na to, że przejście z rozwiązania x^i do rozwiązania $x^{(i+1)}$ można najczęściej opisać pewnym „ruchem" (np. wstaw, zamień), zatem często zakazuje się wykonywania określonych ruchów.

Przykładowo, jeżeli mamy permutację

$$\pi^i =< 2, 3, 6, 4, 1, 5 >$$

i wykonamy ruch *zamień*$(k, l) = (3, 5)$ to otrzymamy :

$$\pi^{(i+1)} =< 2, 3, 1, 4, 6, 5 > .$$

Na liście tabu możemy zapamiętać parę $(\pi^i(k), \pi^i(l)) = (6, 1)$ i zakazane będą wszystkie permutacje, w których element 6 występuje przed elementem 1, np. $< 6, 1, 2, 3, 4, 5 >$, $< 6, 2, 3, 4, 1, 5 >$, $< 2, 3, 4, 6, 5, 1 >$, itd., czyli wszystkie ruchy prowadzące do takich permutacji.

Niektóre metody zabraniające ruchów lub odwiedzania rozwiązań mogą uniemożliwić przejście do rozwiązań jeszcze nie odwiedzonych a bardzo obiecujących. Aby dać takim rozwiązaniom szansę stosuje się tzw. **funkcję aspiracji** – rozwiązanie zabronione (tj. będące na liście tabu) *jest* akceptowane, jeśli wartość funkcji aspiracji osiąga dla danego rozwiązania określoną wartość.

Przykładowe funkcje aspiracji:

1. Jeżeli rozwiązanie jest **lepsze od jakiegokolwiek wcześniejszego** rozwiązania to poziom aspiracji jest przekroczony.

2. (dla listy zapamiętującej atrybuty ruchów) Jeżeli atrybut przejścia

$$x^i \rightarrow x^{(i+1)}$$

jest na liście tabu to poziom aspiracji jest przekroczony gdy

$$\boldsymbol{K(x^i) \leq K(x^{(i+1)}).}$$

Parametry algorytmu TS, które należy określić to:

- sposób wyznaczania rozwiązania początkowego (np. losowo),
- rodzaj sąsiedztwa,
- postać i długość listy tabu,
- funkcję aspiracji (opcjonalnie),
- kryterium stopu (np. wykonanie określonej liczby iteracji bez poprawy funkcji celu (z określoną dokładnością), całkowita ilość iteracji, czas działania, itp.).

Dla niektórych wariantów metody TS wykazano teoretyczną zbieżność do rozwiązania optymalnego.

Schemat metody TS

procedure Tabu_search;
begin
wyznacz(x^0);
$x^* := x^0$; {x^* - najlepsze dotychczasowe rozwiązanie}
$K^* := K(x^0)$; {K^* - wartość f. kryterialnej dla x^*}
inicjuj_listę_tabu; {jako pustą}
$i := 0$;
repeat
M_i := liczba_elementów($N(x^i)$);
$K' := \infty$; {K' - wart. najlepszego rozw. znaleziona podczas przeszukiwania $N(x^i)$}
for $j := 1$ **to** M_i **do**
begin
x:=kolejny_element($N(x^i)$);
if ($K(x) < K'$) **AND** (przejście $x^i \to x$ nie jest na liście tabu **OR**
przejście $x^i \to x$ jest na liście tabu ale spełnia kryterium aspiracji)
then $K' := K(x)$; $x^{i+1} := x$;
end;
if $K' < K^*$ **then** $x^* := x^{i+1}$; $K^* = K'$;
dodaj przejście $x^i \to x^{i+1}$ na koniec listy tabu
(usuwając przejście z początku listy jeśli jest pełna);
$i := i + 1$;
until kryterium_stopu;
end;

Poszukiwanie genetyczne (ewolucyjne)

ang. ***Genetic Algorithms (GA)*** – Z. Michalewicz (1992).

Algorytmy GA wykorzystują mechanizm **dostosowywania się żywych organizmów do otaczającego ich środowiska** na drodze ewolucji.

W kontekście problemów optymalizacyjnych:

- Miarą przystosowania się ***osobnika*** (tj. rozwiązania x danego problemu) jest wartość tzw. ***funkcji przystosowania (adaptacji)*** i może nią być wartość funkcji celu $K(x)$.
- Każdy osobnik posiada pewien zbiór ***genów (chromosomów)***, który go jednoznacznie charakteryzuje (kodowanie rozwiązania w określony sposób przy pomocy zbioru atrybutów, np. poprzez permutację).

W każdej iteracji, i, algorytmu GA dysponujemy pewną ***populacją*** osobników (tj. podzbiorem rozproszonych rozwiązań $W \subset X$) i poddajemy ją kolejno procesowi: reprodukcji, krzyżowania, mutacji i przeżycia (selekcji).

Reprodukcja:

Polega na wybraniu k osobników z populacji do tzw. ***puli rodzicielskiej*** ($k \leq |W|$). Każdy z osobników może zostać wybrany kilka razy.

Metody wyboru:

- **losowa** – jednakowe prawdopodobieństwo wybrania każdego z osobników:

$$p_x = \frac{1}{|W|} \quad \text{dla } x \in W,$$

- **„k najlepszych”** – każdy z osobników jest wybierany dokładnie raz – jeśli ma odpowiednio wysoką wartość funkcji przystosowania, lub nie jest wybierany wcale,

- **ruletki** – prawdopodobieństwo wybrania każdego z osobników, p_x, jest proporcjonalne do wartości jego funkcji przystosowania, np.

$$p_x = \frac{K(x)}{\sum_{j \in W} p_j} \quad \text{dla} \quad x \in W.$$

Krzyżowanie:

Proces wyznaczania nowych rozwiązań na podstawie osobników z puli rodzicielskiej. W ramach krzyżowania, osobniki rodzicielskie kojarzone są w pary (np. losowo lub osobniki sąsiadujące). Każdy nowy osobnik, x', wyznaczany jest z danej pary rodziców, x^l oraz x^m, przy pomocy określonego ***operatora krzyżowania***, przejmując (częściowo) cechy (chromosomy) obu rodziców.

Najpopularniejsze rodzaje krzyżowania na reprezentacji permutacji:

Jednopunktowe:

- permutacje rodzicielskie x^l, x^m dzielimy na 2 części w losowej (tej samej) pozycji (tzw. ***punkcie krzyżowania***);
- jednego potomka, x', otrzymujemy wstawiając pierwszą część x^m na początek x^l, a drugiego, x'', poprzez wstawienie pierwszej części x^l na początek x^m;
- następnie usuwane są z obu nowopowstałych permutacji powtarzające się indeksy (w celu otrzymania rozwiązań dopuszczalnych) tak aby zachować w całości nowowstawiony fragment.

Przykład:

$$
\begin{aligned}
x^l: &\quad < 1,4,6,3 \mid 7,8,2,5,9 > \\
x^m: &\quad < 9,7,5,2 \mid 1,3,4,6,8 > \\
&\quad \Downarrow \\
x': &\quad < 9,7,5,2 \,\|1,4,6,3,[7],8,[2],[5],[9] > \\
x'': &\quad < 1,4,6,3 \,\|9,7,5,2,[1],[3],[4],[6],8 > \\
&\quad \Downarrow \\
x': &\quad < 9,7,5,2,1,4,6,3,8 > \\
x'': &\quad < 1,4,6,3,9,7,5,2,8 >
\end{aligned}
$$

Dwupunktowe – losowane są dwa punkty krzyżowania i wymieniane są fragmenty permutacji pomiędzy tymi punktami (oczywiście z usuwaniem powtarzających się indeksów).

Wielopunktowe – wymieniamy fragmenty chromosomów pomiędzy punktami 1 i 2, 3 i 4, itd.

Jednostajne – dla każdej pozycji decydujemy losowo, w sposób niezależny i z jednakowym prawdopodobieństwem, czy zamienić miejscami odpowiednie indeksy, czy też pozostawić je bez zmian.

Z częściowym odwzorowaniem (PMX).

Z porządkowaniem (OX).

Mutacja:

- Powoduje sporadyczne, losowe (z małym prawdopodobieństwem, np. $\mathbf{0,001 \div 0,1}$) zmiany w chromosomach osobników.
- Proces ten ma na celu zabezpieczenie przed utratą różnorodności genetycznej populacji (zbyt dużo osobników sąsiednich – z jednego obszaru).
- W przypadku reprezentacji osobników przy pomocy permutacji, może to być losowy ruch typu **zamień**, **wstaw** lub **odwróć**.

Selekcja (przeżycie):

Proces wyboru, spośród rodziców i potomków, osobników do kolejnej fazy (iteracji algorytmu). Do kolejnej fazy mogą przechodzić np. **wszystkie osobniki** (rosnąca liczność populacji), mogą przechodzić tylko **osobniki potomne**, lub **najlepszych** $|\mathbf{W}|$ **osobników** spośród zarówno osobników rodzicielskich jak i potomnych (w obu ostatnich przypadkach mamy do czynienia ze stałą licznością populacji).

Parametry algorytmu GA, które należy określić to:

- wielkość populacji,
- sposób wyznaczania populacji początkowej (np. losowo),
- funkcja przystosowania (np. funkcja celu),
- metoda reprodukcji,
- rodzaj i prawdopodobieństwo, p_k, krzyżowania genetycznego,
- rodzaj i prawdopodobieństwo, p_m, mutacji,
- metoda selekcji.

Schemat metody GA

procedure Algorytm_genetyczny;
begin
wylosuj N osobników do populacji początkowej W^0;
$K^* := \min_{x \in \mathcal{W}^0} K(x)$; {$K^*$ - wart. f. kryt. dla najlepszego dotychczasow. rozwiązania}
$x^* := \{x : K(x) = K^*\}$; {$x^*$ - najlepsze dotychczasowe rozwiązanie}
$i := 0$;
repeat { Pętla tworząca nowe populacje }
wyznacz populację rodzicielską z W^i;
j := 1;
repeat { Pętla wykonująca operacje na parze osobników z populacji }
wyznacz osobniki rodzicielskie (x^l, x^m) z populacji W^i;
wygeneruj liczby losowe(R_k, R_m^1, R_m^2);
if $R_k \leq p_k$ **then** (x', x'') := krzyżowanie(x^l, x^m);
if $R_m^1 \leq p_m$ **then** x' := mutacja(x');
if $R_m^2 \leq p_m$ **then** x'' := mutacja(x'');
wstaw (x', x'') do populacji W^i;
j := j + 2;
until $j > k$;
{ Poniżej zapisanie najlepszego rozwiązania }
if $K^* > \min_{x \in \mathcal{W}^i} K(x)$ **then** $K^* := \min_{x \in \mathcal{W}^i} K(x)$; $x^* := \{x : K(x) = K^*\}$;
W^{i+1} := selekcja(W^i);
$i := i + 1$;
until kryterium_stopu;
end;

Poszukiwanie mrówkowe

ang. ***Ant Search (AS)*** – Marco Dorigo (1992).

Systemy mrówkowe wykorzystują sposób zachowania się kolonii mrówek do rozwiązania problemów optymalizacji kombinatorycznej. Ich cechą charakterystyczną jest współpraca autonomicznych osobników o niskich zdolnościach (mała orientacja w terenie, niska inteligencja, itp.) – pojedyncze mrówki – do osiągania zaawansowanych celów (np. wyszukiwanie najkrótszych tras z mrowiska do źródeł pożywienia) – **inteligencja grupowa**.

W przypadku kolonii mrówek, środkiem komunikacji (współpracy) jest pozostawiany przez nie ***feromon***.

Natomiast pojedyncza mrówka porusza się chaotycznie (losowo), biorąc pod uwagę tylko poziom feromonu, a jej celem jest znalezienie pożywienia (i zaniesienie go do mrowiska).

Feromon to substancja zapachowa o następujących właściwościach:

- Jest pozostawiana przez każdą mrówkę na przebytej przez nią trasie – ***ścieżka feromonowa***.
- Przejście kolejnej (lub tej samej) mrówki tą samą trasą (lub jej fragmentem) powoduje **wzrost stężenia feromonu** na danym odcinku.
- Mrówki preferują poruszanie się po ścieżkach o **wysokim stężeniu feromonu**, gdyż oznacza to, że wiele mrówek tamtędy już przeszło, a więc jest duże prawdopodobieństwo, że ścieżka jest właściwa (np. prowadzi do pożywienia). Zatem coraz więcej mrówek porusza się po często uczęszczanych trasach.
- Feromon **ulatnia się** z czasem, w związku z czym na mało uczęszczanych trasach stężenie feromonu maleje. Zatem coraz mniej mrówek przechodzi po trasach mało uczęszczanych.
- W efekcie zachodzi **dodatnie sprzężenie zwrotne** – dobrymi drogami podąża coraz więcej mrówek.

W kontekście **optymalizacji kombinatorycznej**:

- Najwygodniej stosować systemy mrówkowe do problemów, które można reprezentować w postaci grafów (np. problem komiwojażera).
- Wtedy rozwiązanie może być reprezentowane jako ścieżka w grafie i konstruowane stopniowo, co odzwierciedla poruszanie się mrówki.
- Skonstruowanie pełnego rozwiązania oznacza dojście mrówki do celu (pożywienia). Wartość funkcji celu skonstruowanego rozwiązania będzie długością ścieżki przebytej przez daną mrówkę (zatem jej minimalizacja jest równoważna celowi mrówki, czyli minimalizacji długości przebytej drogi).
- Możemy „puszczać"jedną lub wiele mrówek, jednocześnie lub jedna po drugiej. **Efektywność** systemu mrówkowego (czas działania i jakość dostarczonego rowiązania) zależy wprost od przyjętej metody.

Przykład:

Algorytm mrówkowy do rozwiązania Problemu Komiwojażera

Problem Komiwojażera (PK) – przypomnienie

Dane:

n – liczba miast, $n \in \mathbf{Z}^+$,

$c_{ji},\ i,j \in \{1,\ldots,n\},\ i \neq j$ – odległość między miastem i a miastem j,

$$c_{ji} = c_{ij},\ c_{ji} \in \mathbf{R}^+.$$

Zadanie:

Znaleźć permutację miast π^*, dla której

$$TC(\pi^*) = \sum_{j=1}^{n-1} c_{\pi^*(j)\pi^*(j+1)} + c_{\pi^*(n)\pi^*(1)} \longrightarrow \min.$$

Zachowanie mrówki przy rozwiązywaniu PK:

- Mrówka rozpoczyna trasę z dowolnego miasta.
- Prawdopodobieństwo przejścia do innego miasta zależy od:

  1. odległości do tego miasta,
  2. intensywności feromonu na trasie do tego miasta.

- Trasa mrówki musi spełniać te same wymagania co trasa komiwojażera, tzn. przechodzić przez każde z miast tylko raz. Aby to zagwarantować każda mrówka musi mieć listę już odwiedzonych miast.
- Po odwiedzeniu przez mrówkę wszystkich miast otrzymujemy rozwiązanie. Po zakończeniu trasy należy uaktualnić feromon na wszystkich drogach między miastami (zwiększyć na trasie mrówki, zmniejszyć poza) i można wypuszczać w trasę **następną mrówkę***.

*Poziom feromonu można także aktualizować po każdym kroku mrówki. Mrówki można wtedy wypuszczać równolegle

Q - ilość feromonu zostawiana przez pojedynczą mrówkę pomiędzy dwoma miastami,

$\Delta\tau_{ij} = mQ$ - zmiana intensywności feromonu pomiędzy miastem i a j wywołana przejściem m mrówek,

$\tau_{ij}(t)$ - intensywność feromonu pomiędzy miastem i a j w chwili t,

w - współczynnik wysychania feromonu w jednostce czasu.

Stosując powyższe oznaczenia otrzymujemy formułę wyrażającą **zmianę intensywności feromonu** pomiędzy miastami i a j **w jednostce czasu**:

$$\tau_{ij}(t+1) = w \cdot \tau_{ij}(t) + \Delta\tau_{ij}.$$

Decydując się na podjęcie podróży do kolejnego miasta mrówka kieruje się trzema informacjami:

1. **Widocznością** miasta, do którego może się udać z miasta, w którym aktualnie jest. **Widoczność** miasta j z miasta i jest definiowane jako odwrotność odległości z miasta i do miasta j: $\eta_{ij} = \frac{1}{c_{ij}}$.

2. **Intensywnością feromonu** na drodze do miasta, do którego może się udać w danej chwili. **Intensywność feromonu** na drodze z miasta i do miasta j w iteracji t jest dana przez $\tau_{ij}(t)$.

3. **Listą** miast już odwiedzonych.

Najprostsza możliwa formuła uwzględniająca powyższe informacje i umożliwiająca wyliczenie **prawdopodobieństwa przejścia** z miasta i do miasta j **przez mrówkę** k jest następująca:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}(t) \cdot \eta_{ij}}{\sum\limits_{l \in D_k} \tau_{il}(t) \cdot \eta_{il}} & \text{jeżeli } j \in D_k, \\ 0 & \text{w przeciwnym wypadku,} \end{cases}$$

gdzie D_k jest zbiorem miast nieodwiedzonych jeszcze przez mrówkę k.

Następująca zmiana wzoru pozwala na **większą kontrolę** pomiędzy sugerowaniem się widocznością miasta a popularnością trasy wśród mrówek:

$$p_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in D_k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} & \text{jeżeli } j \in D_k, \\ 0 & \text{w przeciwnym wypadku,} \end{cases}$$

gdzie parametry α i β są dobierane przez użytkownika. Wysokie α powoduje, że mrówki kierują się głównie na ścieżki feromonowe. Niskie α zbliża poszukiwanie mrówkowe do algorytmu zachłannego.

Schemat metody AS

procedure Algorytm_mrówkowy;
begin
    zainicjuj(m,Q,w);
    $K^* := \infty$; {K^* - wart. f. kryt. dla najlepszego dotychczasow. rozw.}
    $x^* := \emptyset$; {x^* - najlepsze dotychczasowe rozwiązanie}
    rozmieść m mrówek losowo;
    $t := 0$;
    repeat { Pętla realizująca pojedyncze przejście m mrówek }
        for $i := 1$ **to** m **do**
            begin
                mrówka(i) konstruuje rozwiązanie x^i;
                if $K^* > K(x^i)$ **then** $K^* := K(x^i)$; $x^* := x^i$;
            end;
        uaktualnij $\tau_{ij}(t)$;
        $t := t + 1$;
    until kryterium_stopu;
end;

W problemie komiwojażera skojarzenie długości trasy z odległością od źródła pożywienia jest oczywiste. W ogólności przy rozwiązywaniu problemów kombinatorycznych przechodzenie trasą jest równoznaczne z konstruowaniem rozwiązania. Przykłady:

- Kolejne kroki na trasie to dobieranie kolejnych przedmiotów do plecaka w **problemie plecakowym**.

- Kolejne kroki na trasie przy rozwiązywaniu **problemu szeregowania** mogą być dołączaniem kolejnych zadań do częściowego uszeregowania.

Długość całej trasy jest proporcjonalna do wartości funkcji celu dla danej trasy (reprezentującej rozwiązanie).