ALHE

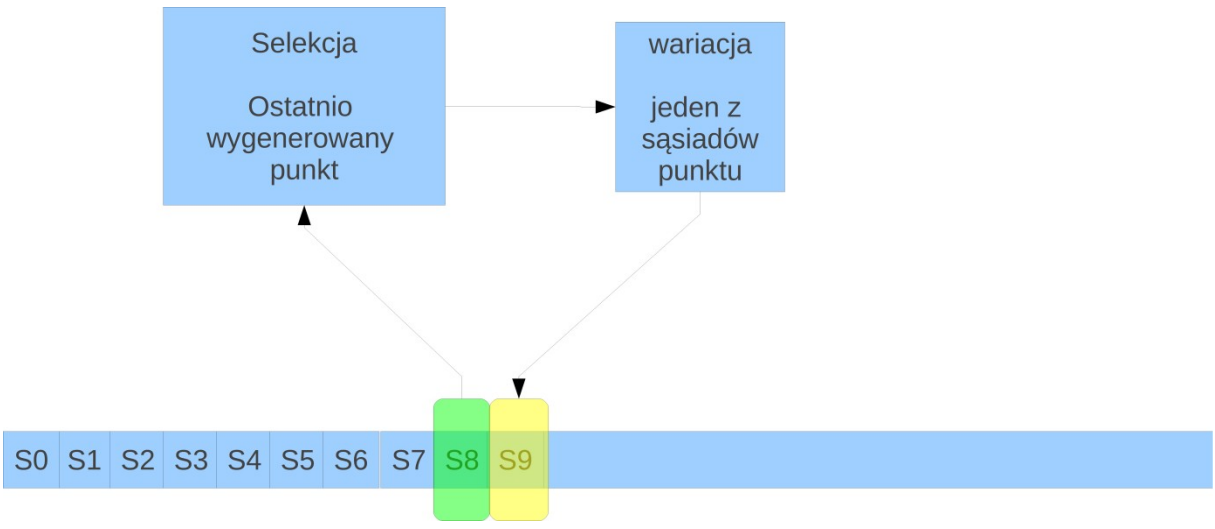
prof. Jarosław Arabas
semestr 15Z

Wykład 5 – Błądzenie przypadkowe, Algorytm wspinaczkowy, Przeszukiwanie ze zmiennym sąsiedztwem, Tabu, Symulowane wyżarzanie

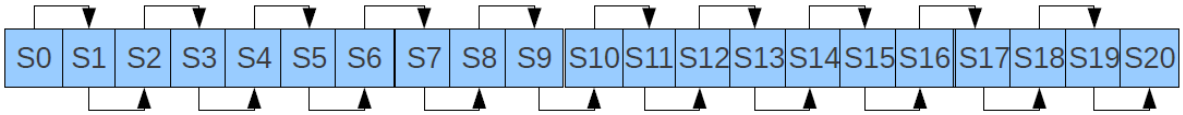
1. Błądzenie przypadkowe:

$$
\begin{aligned}
&\textit{algorytm}\ \textbf{\textit{błądzenie przypadkowe}}\\
&H \leftarrow init(s_0)\\
&\textbf{while}\ !\ stop\\
&\quad \boldsymbol{x} \leftarrow selLast(H)\\
&\quad \boldsymbol{y} \leftarrow selRandom(N(\boldsymbol{x}))\\
&\quad H \leftarrow H \cup \{\boldsymbol{y}\}
\end{aligned}
$$

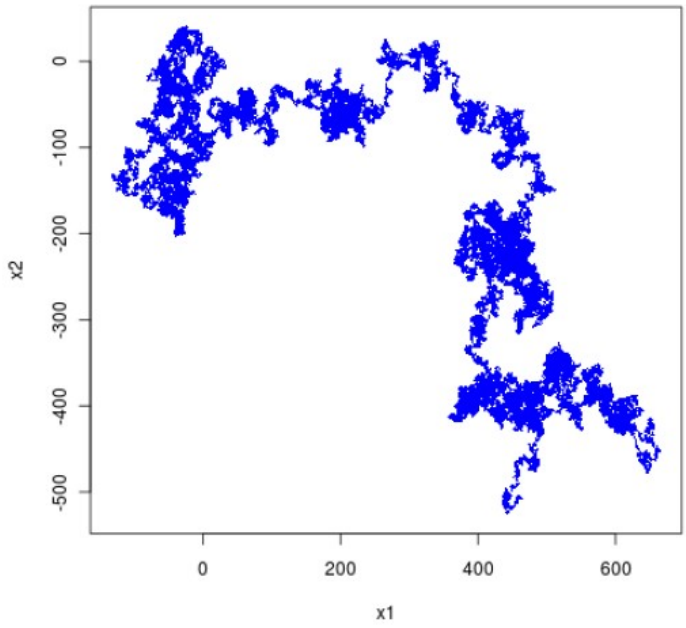
Pierwszym krokiem jest wprowadzenie do logu punktu startu (S_0), losowanego z całej przestrzeni rozwiązań. Selekcja zawsze wybiera najnowszy punkt z historii. Wariacja generuje losowo wybrane rozwiązanie z sąsiedztwa wyselekcjonowanego punktu, gdzie dla każdego sąsiada istnieje takie samo prawdopodobieństwo wyboru. Każdy punkt wygenerowany przez wariację jest akceptowany, niezależnie czy jest lepiej dopasowany czy tez nie. Błądzenie przypadkowe nie posiada naturalnego warunku kończącego.



Oto przedstawiona jest historia generowanych rozwiązań. Strzałki między punktami S_x oraz S_y oznaczają, że punkt S_y jest lokalną modyfikacją punktu S_x. Zawsze nowo wylosowany punkt staje się nowym roboczym, niezależnie od tego czy jest lepszy czy też nie. Okno historii może być minimalnie wielkości 1 .

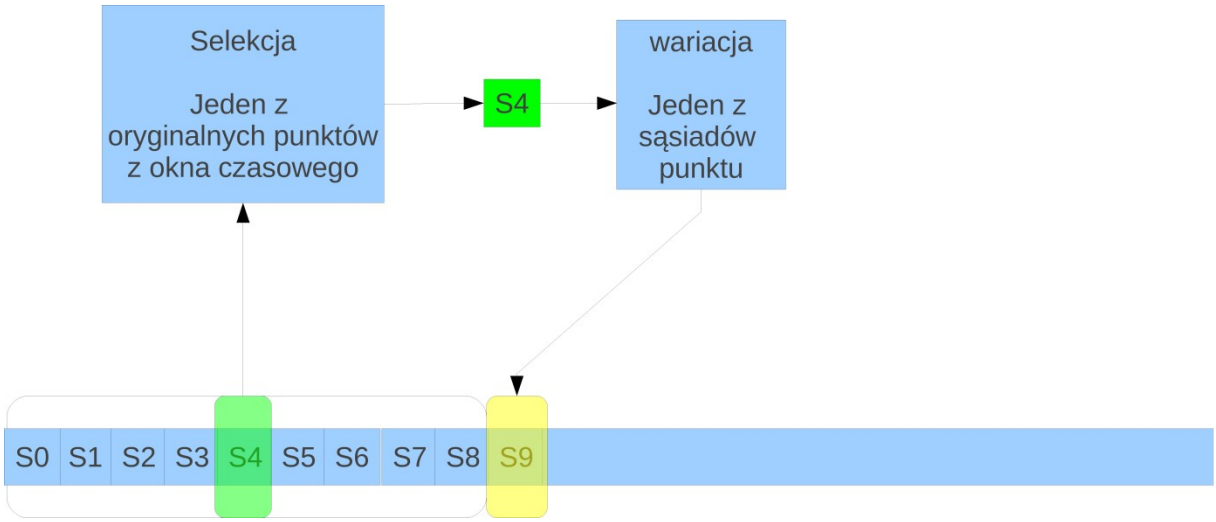


W szczególności może się zdarzyć, że punkt S_0 może być tym samym rozwiązaniem co punkt S_2. Nic nie stoi na przeszkodzie, aby dany algorytm po wyjściu z pewnego rozwiązania zaraz do niego powrócił. Poniżej został przedstawiony poglądowy diagram, który przedstawia przykładowy ślad algorytmu dla przestrzeni dwuwymiarowej.

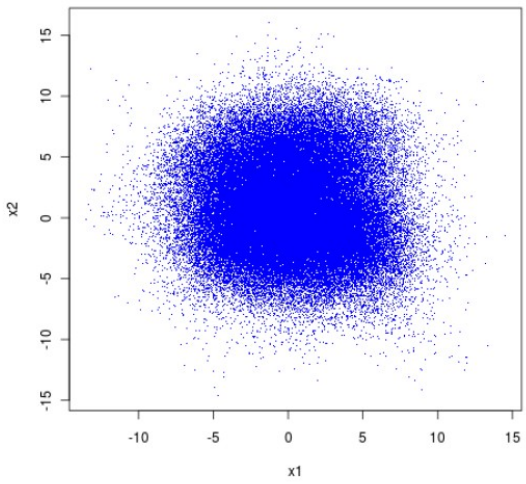


Teoria błądzenia przypadkowego ma zastosowanie w zagadnieniach obsługi masowej i kolejek. Jest również wykorzystywana w teorii ruchów Browna.

2. Rozszerzony algorytm błądzenia przypadkowego.

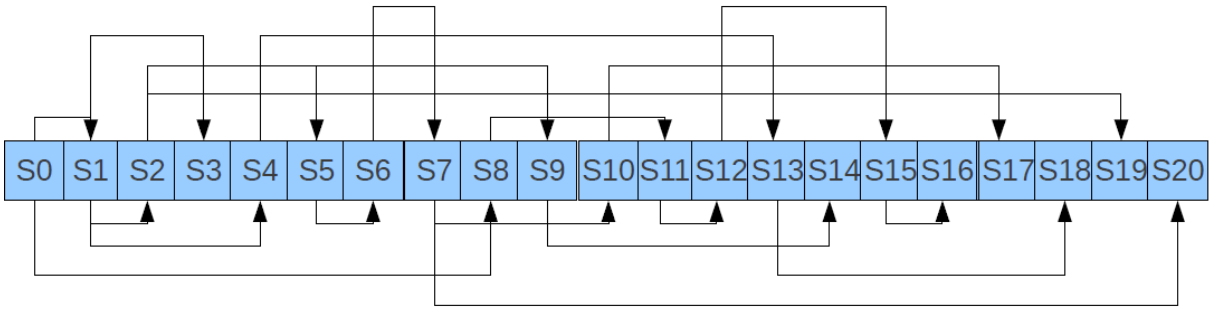


Zmodyfikowano selekcje, która wybiera teraz punkt poprzez losowanie z całego okna historii. Ta mała zmiana wpływa znacząco na ślad algorytmu:



Jest tak, ponieważ w tym przypadku punkt losowany później jest w pewnym stopniu zależny od całej historii, a nie tylko od poprzedniego punktu jak we wcześniejszym przypadku.

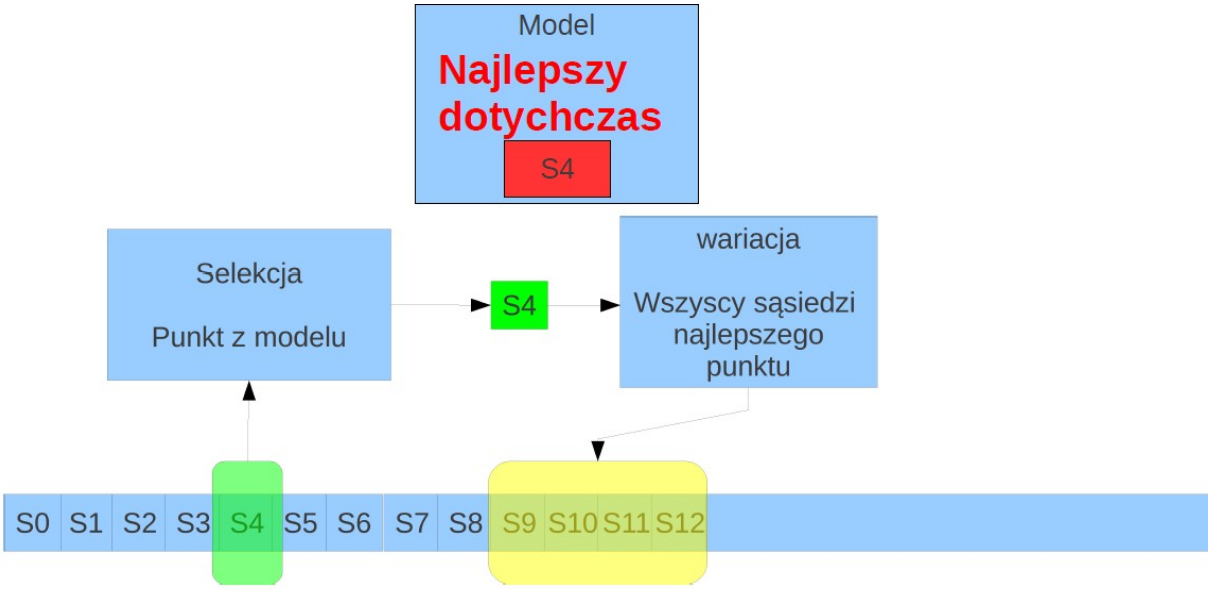
Oto log dla przykładowego wywołania algorytmu. Tym razem nie można ograniczyć okna w żaden sposób.



3. Algorytm wspinaczkowy

algorytm ***wspinaczkowy***
$H \leftarrow init(s_0)$
$\boldsymbol{x} \leftarrow selBest(H)$
while $!\, stop$
  $Y \leftarrow N(\boldsymbol{x})$
  $\boldsymbol{y} \leftarrow selBest(Y)$
  if $q(\boldsymbol{y}) > q(\boldsymbol{x})$
    $\boldsymbol{x} \leftarrow \boldsymbol{y}$
  $H \leftarrow H \cup Y$

Algorytm zaczyna działanie od wyboru z logu rozwiązania o najwyższej funkcji celu. Następnie generujemy całe jego sąsiedztwo i sortujemy je pod względem dopasowania. Rozwiązanie, które jest najbardziej dopasowane zostaje przyrównane do dotychczasowego najlepszego punktu i jeśli jest większe – podmienia je. Potem następuje aktualizacja logu i proces zaczyna się od nowa.



Algorytm wspinaczkowy w pewnym sensie modeluje zachowanie osoby, która w ciemności sprawdza nogą powierzchnie w celu omijania dołków. Może jedynie sprawdzać powierzchnie wokół siebie. Dany algorytm posiada zawsze jeden punkt roboczy. Zawsze generujemy wszystkich sąsiadów i wybieramy najlepszego. Kończymy, gdy w sąsiedztwie nie ma już lepszego.

W niektórych przypadkach niemożliwe jest wygenerowanie całego sąsiedztwa lub też sąsiadów jest za dużo i stosowanie tego algorytmu traci sens.
Wtedy możemy dyskretyzować sąsiedztwo (np. próbkowanie na siatce) lub wybrać punkty przez losowanie (zgodnie z ustalonym rozkładem prawdopodobieństwa).

4. Algorytm wspinaczkowy ze zmiennym sąsiedztwem. (VNS)

Wadą algorytmu wspinaczkowego z pełnym przeglądem sąsiedztwa jest brak mechanizmu ochrony przed utknięciem w maksimum/minimum lokalnym, dlatego też powstał algorytm ze zmiennym sąsiedztwem. Realizuje się to poprzez dokładanie krawędzi w grafie przestrzeni przeszukiwań.

```
algorytm VNS
H ← init(s_0)
x ← selBest(H)
while ! stop
  k ← 1
  repeat
    Y ← N_k(x)
    H ← H ∪ Y
    y ← selBest(Y)
    k ← k + 1
  until (q(y) > q(x) ∨ k > K)
  if (k > K) exit
  x ← y
```

x – punkt roboczy
K – parametr metody

$$N_1(\boldsymbol{x}) \subset N_2(\boldsymbol{x}) \subset \ldots \subset N_K(\boldsymbol{x})$$

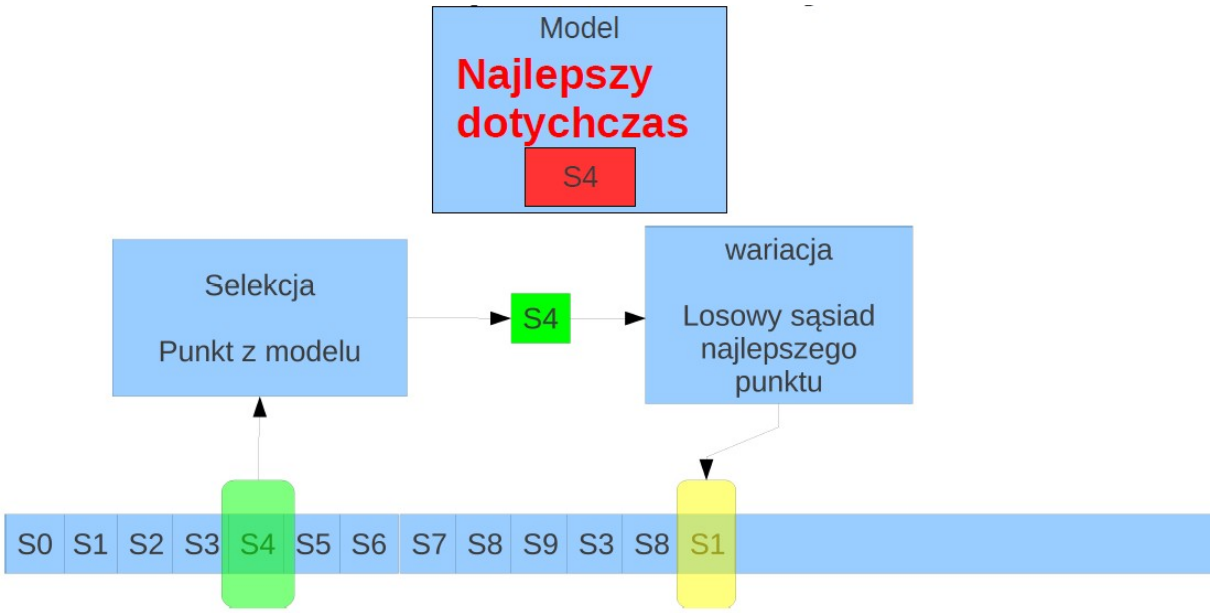
Zasadnicza zmiana w ogólnym działaniu algorytmu pojawia się, jeśli po przejrzeniu najbliższego sąsiedztwa nie znajdziemy lepszego rozwiązania. Wtedy następuje krok wymiany sąsiedztwa poziomu k na poziom k+1. Pętla wewnętrzna algorytmu działa dopóki nie znajdzie się lepsze rozwiązanie lub sprawdziliśmy już sąsiedztwo stopnia K, które definiuje zasięg naszego poszukiwania.

Tak naprawdę mechanizm zabezpiecza nas przed utknięciem w ekstremum lokalnym z sąsiedztwem poziomu K.

5. Stochastyczny algorytm wspinaczkowy

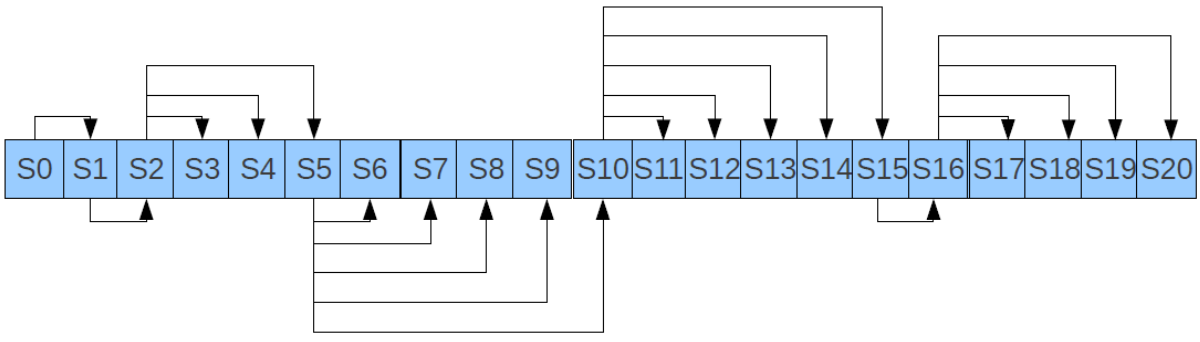
```
algorytm wspinaczkowy
H ← init(s_0)
x ← selBest(H)
while ! stop
  y ← selRandom(N(x))
  if q(y) > q(x)
    x ← y
  H ← H ∪ {y}
```

W stochastycznym algorytmie wspinaczkowym wariacja generuje punkt z obszerniejszego sąsiedztwa przez losowanie bez zwracania. Jeśli jest lepszy to do niego przechodzimy i to on staje się punktem roboczym. Jeśli nie – losujemy jeszcze raz i po każdym losowaniu dodajemy wylosowany punkt do logu.

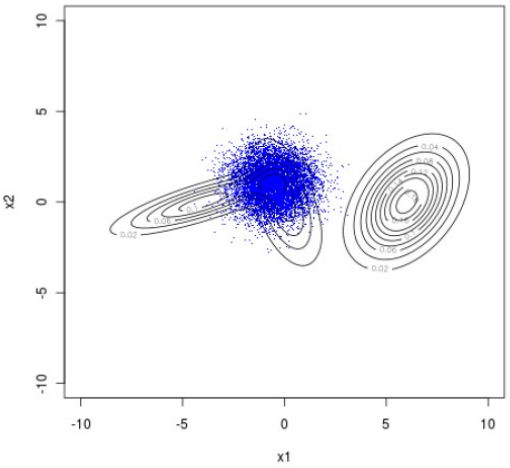


Jak zatrzymać algorytm?

Wraz z liczbą losowań zakończonych porażką maleje szansa trafienia lepszego punktu. Po pewnym czasie możemy zakończyć proces dalszego losowania.



Tak wygląda historia generowanych rozwiązań, gdzie strzałki między punktami S_x oraz S_y oznaczają, że punkt S_y jest lokalną modyfikacją punktu S_x.



Przedstawiony został ślad algorytmu, czyli – zbiór wszystkich wygenerowanych punktów w pojedynczym uruchomieniu.

Linie na wykresie symbolizują warstwice, które w tym wypadku łączą rozwiązania o tej samej wartości funkcji celu.

6. Algorytm symulowanego wyżarzania

algorytm ***symulowane wyżarzanie***

$H \leftarrow init(s_0)$

$\boldsymbol{x} \leftarrow selBest(H)$

while $!\, stop$

  $\boldsymbol{y} \leftarrow selRandom(N(\boldsymbol{x}))$

  if $q(\boldsymbol{y}) > q(\boldsymbol{x})$

    $x \leftarrow y$

  else if $rand() < p_a$

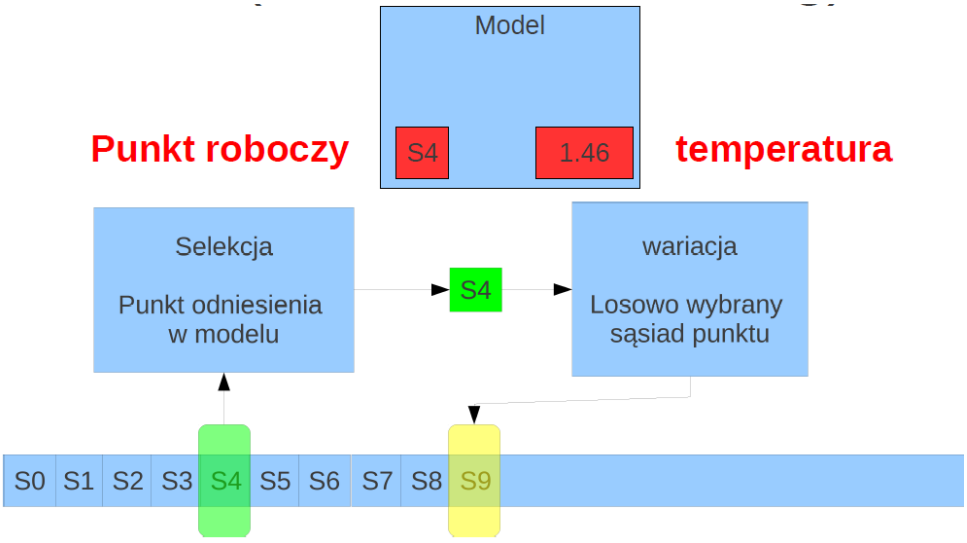
    $x \leftarrow y$

  $H \leftarrow H \cup \{\boldsymbol{y}\}$

x – punkt roboczy
q – funkcja celu maksymalizowana
T - temperatura

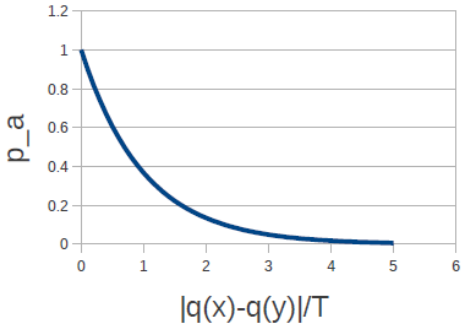
$$p_a = \exp\left(-\frac{|q(\boldsymbol{y}) - q(\boldsymbol{x})|}{T}\right)$$

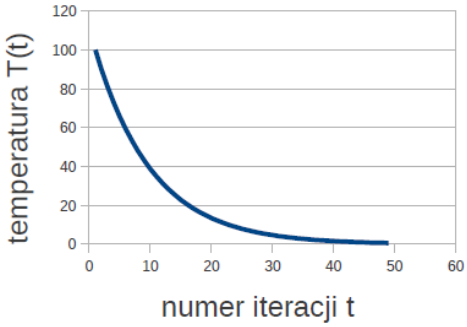
Symulowane wyżarzanie jest w zachowaniu bardzo podobne do algorytmu wspinaczkowego, jednak w tym wypadku nie zależy nam aż tak bardzo na osiąganiu coraz to lepszych wyników. Po wylosowaniu punktu gorzej dopasowanego, na podstawie dodatkowego losowania decydujemy czy go przyjąć czy też odrzucić. Prawdopodobieństwo akceptacji p_a jest zależne od różnicy pomiędzy wynikiem funkcji celu dla punktu roboczego oraz dla punktu, którego indeks jest aktualnie przechowywany w modelu. Istnieje również inny czynnik który wpływa na tę wartość – temperatura (T).



Temperatura jest funkcją energii wewnętrznej z jaka startuje nasz algorytm. Jest punktem odniesienia, który mówi nam o tym jak duża różnica jest pomiędzy wylosowanym rozwiązaniem a tym które jest aktualnie w modelu. W miarę działania algorytmu staje się ona coraz

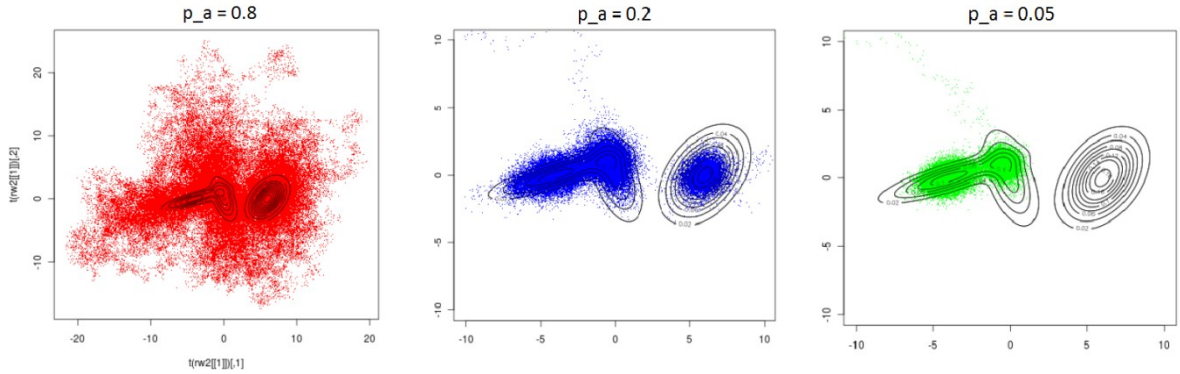
mniejsza. Początkowo algorytm jest bardziej nastawiony na eksplorację, chętniej akceptuje gorsze wyniki, natomiast w późniejszej fazie stawiamy na eksploatację, dlatego algorytm będzie prowadził dłuższe przeszukiwanie lokalne.





Temperatura w końcowej fazie algorytmu powinna być obniżana do wartości bliskiej 0. Na powyższych wykresach widać dokładnie w jaki sposób zmienia się prawdopodobieństwo akceptacji w miarę wykonywania kolejnych iteracji algorytmu.

Poniżej zostały przedstawione ślady algorytmu dla różnego początkowego prawdopodobieństwa akceptacji:



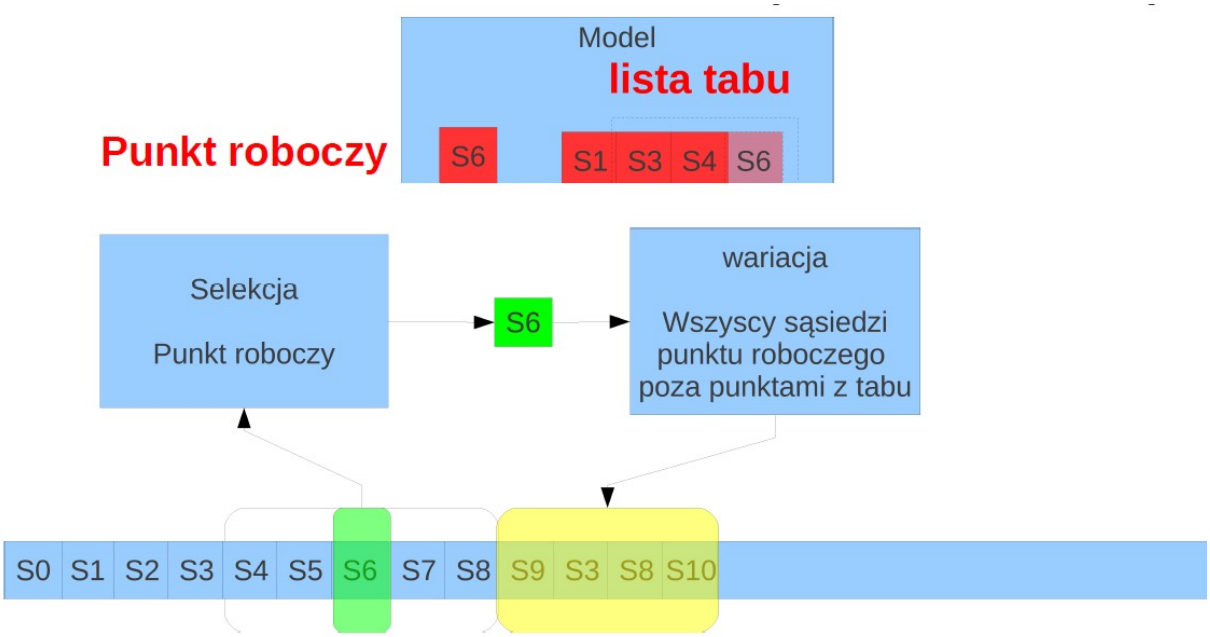
Należy również zaznaczyć że symulowane wyżarzanie przez swoją „pobłażliwość” może być metodą bardzo powolną.

7. Przeszukiwanie z tabu

Istnieje również inny mechanizm, który nie pozwoli zatrzymać się metodzie w ekstremum lokalnym. Jest on oparty na uszczuplaniu przestrzeni przeszukiwań o znalezione punkty (usuwanie krawędzi z przestrzeni przeszukiwań).

algorytm ***przeszukiwanie z tabu***
$T \leftarrow \emptyset$
$H \leftarrow init(s_0)$
$x \leftarrow selBest(H)$
while *! stop*
  $Y \leftarrow N(x) \setminus T$
  $x \leftarrow selBest(Y)$
  define T_j *for delection*
  $T \leftarrow T \cup \{x\} \setminus (T_j)$
  $H \leftarrow H \cup Y$

Początkowo lista tabu jest pusta. Po wyborze najlepszego punktu sąsiadującego z punktem S_0, wrzucamy go do listy tabu. Następnie aktualizujemy log i ponownie generujemy sąsiedztwo nowo wybranego punktu. Przy czym jeśli w danym sąsiedztwie znajduje się rozwiązanie umieszczone aktualnie na liście tabu- nie bierzemy go pod uwagę. Oczywiście po pewnej określonej liczbie iteracji punkty z listy tabu zostają usunięte. Dany algorytm nie ma naturalnego warunku zakończenia. Jego wadą jest również to, że trzeba przetrzymywać w pamięci listę tabu.



Mechanizm tabu można utworzyć na dwa sposoby:

a) Wariancja nie może wygenerować punktu, który jest na liście tabu
b) Selekcja nie może wybrać punktu, który jest na liście tabu

Lista tabu również może być zorganizowana na wiele sposobów:

a) Kolejka FIFO
b) Losowy dostęp
c) Kolejka priorytetowa wg:
   - funkcji celu
   - szacowanej funkcji celu
   - podobieństwa
   - wiedzy dziedzinowej

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). W tej sytuacji możliwe jest złamanie tabu.

VNS i tabu zapobiegają stopowaniu algorytmów w ekstremum lokalnym przez modelowanie grafu przestrzeni przeszukiwań (dodawaniu lub usuwanie z niego krawędzi). Można również zmodyfikować samo działanie algorytmu, jak jest to pokazane w algorytmie stymulowanego wyżarzania.

8. Stabuizowane symulowane wyżarzanie

Mechanizm tabu można równie dobrze dołączyć do stymulowanego wyżarzania jak i stochastycznego wzrostu. Przypadek pierwszy możemy zaobserwować poniżej:

algorytm ***symulowane wyżarzanie z tabu***

$T \leftarrow \emptyset$

$H \leftarrow init(s_0)$

$x \leftarrow selBest(H)$

while $!\, stop$

$Y \leftarrow N(x) \setminus (N(T_1) \cup N(T_2) \cup .... N(T_k))$

$\boldsymbol{y} \leftarrow selRandom(Y)$

if $q(\boldsymbol{y}) > q(\boldsymbol{x})$

$x \leftarrow y$

else if $rand() < p_a$

$x \leftarrow y$

$define\ T_j\ for\ delection$

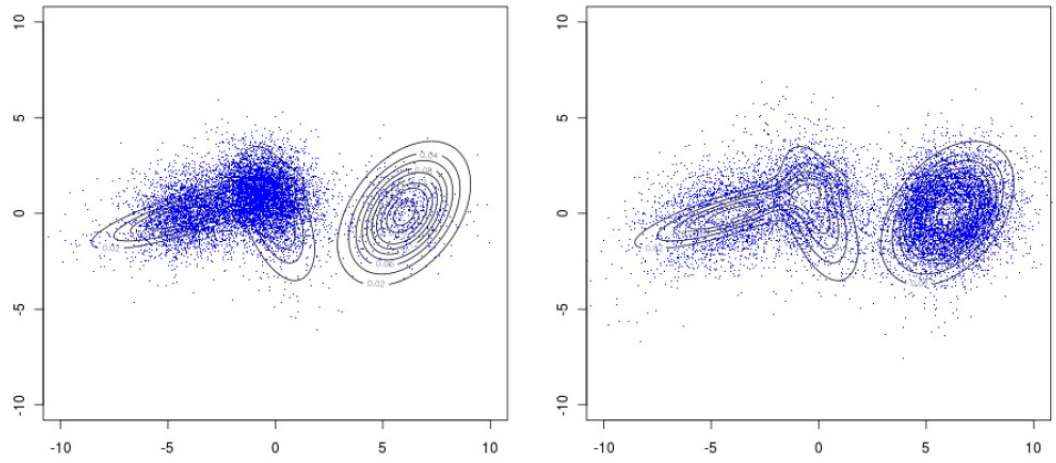
$T \leftarrow T \cup \{x\} \setminus (T_j)$

$H \leftarrow H \cup \{y\}$

T – kolejka tabu

Z tego pseudokodu możemy wywnioskować, że do wariancji dochodzi już niepełne sąsiedztwo punktu roboczego. Jest ono pomniejszone o sąsiedztwa rozwiązań obecnych aktualnie na liście tabu.

Porównanie działania tego algorytmu z i bez mechanizmu tabu:

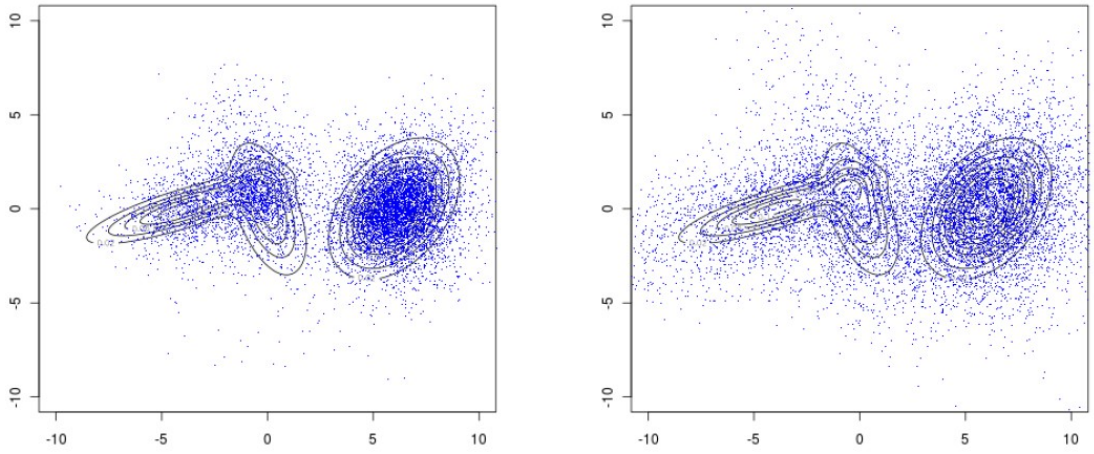


Który z tych rysunków jest z tabu (FIFO 10)?

p_a=0.05

Dla $p_a = 0.05$ algorytm bez tabu zauważalnie dłużej oscylował wokół ekstremum lokalnego widocznego po lewej stronie płaszczyzny, lecz w końcu dotarł również do ekstremum globalnego. Ślad algorytmu po dodaniu listy tabu zmienił się znacząco w okolicach ekstremów lokalnych.

Zmniejszyła się liczba generowanych tam punktów w stosunku do dalszych okolic tych ekstremów. Jest to spowodowane tym, że zdecydowana większość punktów na liście tabu pochodzi z sąsiedztwa ekstremów.



Który z tych rysunków jest z tabu (FIFO 10)?

p_a=0.2

Po zwiększeniu prawdopodobieństwa akceptacji można zauważyć, że algorytm bez tabu poradził sobie znacznie lepiej. (dla $p_a = 0.05$ było bardzo trudno zejść z pierwszego napotkanego ekstremum lokalnego). Natomiast algorytm wykorzystujący tabu znacznie się rozproszył.

Przygotował:

Kamil Kacperski