Omówienie – zestaw 6., część 2.[1]

Ad. 2. Symulowane wyżarzanie w zastosowaniu do problemu komiwojażera (*algorytm udostępniony w osobnym pliku: "Algorytmy do projektu nr 6"*)

Metoda symulowanego wyżarzania była omawiana również na kursie metod numerycznych.

Drugie zadanie polega na rozwiązaniu **problemu komiwojażera** dla dwustu zadanych punktów na dwuwymiarowej płaszczyźnie. Współrzędne (x, y) punktów zostały zamieszczone na UPeL-u w pliku tekstowym `input.dat`. Wszystkich punktów w pliku jest $n = 200$.

W problemie komiwojażera celem jest znalezienie **cyklu Hamiltona o najmniejszej całkowitej wadze** (długości) w **grafie ważonym**. Graf jest **pełny**: między każdymi dwoma wierzchołkami istnieje krawędź (między każdymi dwoma punktami istnieje bezpośrednie połączenie; celem komiwojażera jest odwiedzenie jednokrotnie każdego punktu – np. miasta – i powrót do punktu początkowego). **Wagę cyklu** (u nas: długość) definiujemy jako sumę długości kolejnych krawędzi, składających się na cykl, natomiast długością krawędzi (u, v) jest **odległość euklidesowa** między punktami u i v. Problem jest nietrywialny, ponieważ liczba wszystkich możliwych cykli Hamiltona wynosi $\frac{(n-1)!}{2}$, więc jest znacząca już dla niewielkiej liczby wierzchołków[2]. Nie istnieje rozwiązanie w czasie wielomianowym, dlatego często stosuje się podejście **heurystyczne**, które w stosunkowo krótkim czasie daje pewne **przybliżenie rozwiązania**. Nie mamy jednak gwarancji, że znaleziony cykl faktycznie będzie najkrótszym możliwym cyklem.

Jednym z rozwiązań problemu komiwojażera jest algorytm **symulowanego wyżarzania**, należący do rodziny metod **Monte Carlo**. Zanim do niego przejdziemy w podpunkcie (**c**), prześledzimy prostszą wersję algorytmu typu Monte Carlo (**a**), którą będziemy stopniowo rozbudowywać, rozważając pośredni algorytm: **Metropolisa–Hastingsa** (**b**).

Poniższe algorytmy zostaną podane w wersji rozwiązującej zagadnienie minimalizacji jednowymiarowej funkcji $f(x)$ – celem jest znalezienie **minimum funkcji**. Późniejsze uogólnienie problemu minimalizacji funkcji na zagadnienie komiwojażera okaże się naturalne.

(a) Metoda Monte Carlo

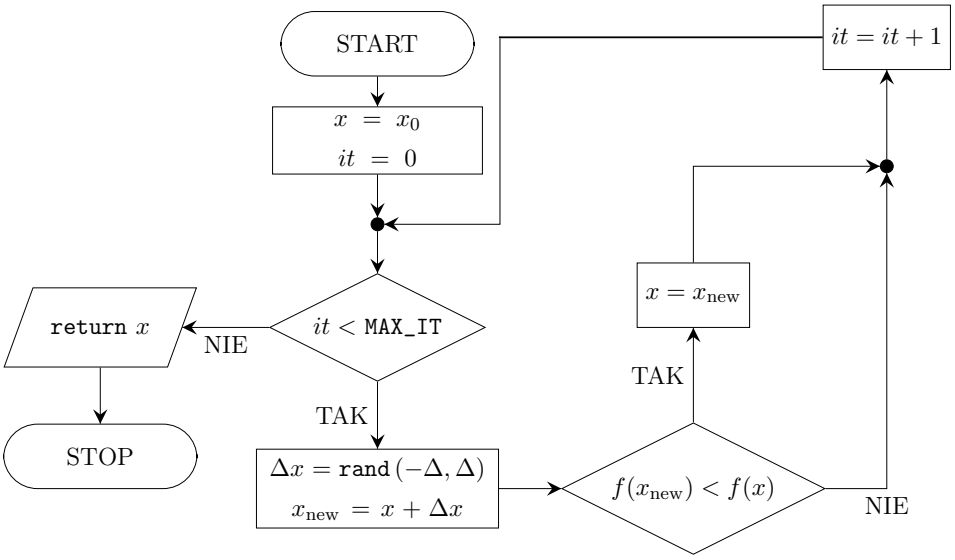
Metody Monte Carlo pozwalają na modelowanie – poprzez **losowania** – procesów zbyt złożonych do rozwiązania analitycznego. Pomimo, że wejściowy problem jest deterministyczny, będziemy rozwiązywać go w sposób **probabilistyczny**.

Algorytm rozpoczniemy od zadanej startowej współrzędnej x_0. Jednym z najprostszych podejść typu Monte Carlo do zagadnienia minimalizacji funkcji $f(x)$ jest wielokrotne **losowanie niewielkiego kroku** Δx, o który przesuwamy aktualne przybliżenie x. Jeśli wartość funkcji w nowym punkcie jest mniejsza niż wartość funkcji w poprzednim, to akceptujemy

[1] W razie jakichkolwiek uwag do niniejszego dokumentu (choćby literówek) proszę o kontakt: `Elzbieta.Strzalka@fis.agh.edu.pl`

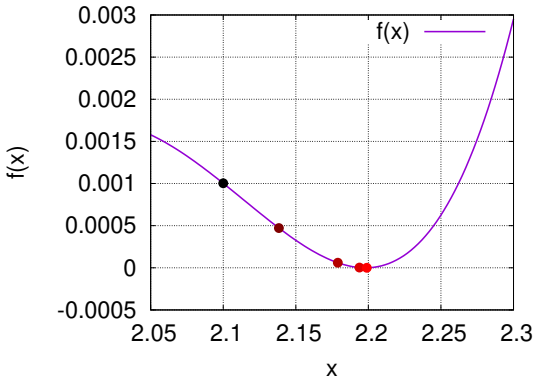
[2] W naszym pliku wejściowym znajduje się $n = 200$ punktów, co daje liczbę możliwych cykli o 374 cyfrach.

nowy punkt jako nowe przybliżenie minimum funkcji. Algorytm przerywamy np. po ustalonej maksymalnej liczbie iteracji. Przybliżenie z ostatniej iteracji zostaje uznane za znalezione minimum funkcji. Algorytm przedstawiono na schemacie z rys. 1.

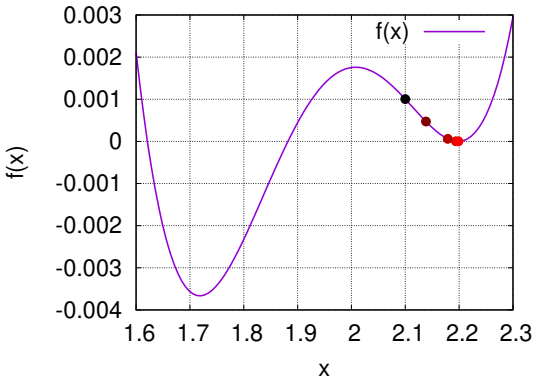


Rysunek 1: Schemat algorytmu **Monte Carlo** – wyszukiwanie minimum funkcji $f(x)$. x_0 jest punktem początkowym; `MAX_IT` to maksymalna liczba iteracji; Δ – wartość definiująca przedział $[-\Delta, \Delta]$; z którego losowane jest aktualne przesunięcie Δx.

Przykład działania algorytmu zilustrowano na rys. 2. Algorytm ma istotną wadę: najczęściej pozwala na znalezienie tylko **minimum lokalnego**, a nie **globalnego**. Punkt o wyższej wartości funkcji nigdy nie zostanie wybrany jako nowe przybliżenie rozwiązania, więc algorytm nie jest w stanie wydostać się z okolic minimum lokalnego.



(a) Przedział wykresu zawężony do $x \in [2.05, 2.3]$. Algorytm pozwolił na znalezienie przybliżonego minimum w okolicy współrzędnej $x = 2.2$.



(b) Wykres z rys. (a) rozszerzony do przedziału $x \in [1.6, 2.3]$: znalezione rozwiązanie okazuje się być tylko **minimum lokalnym**.

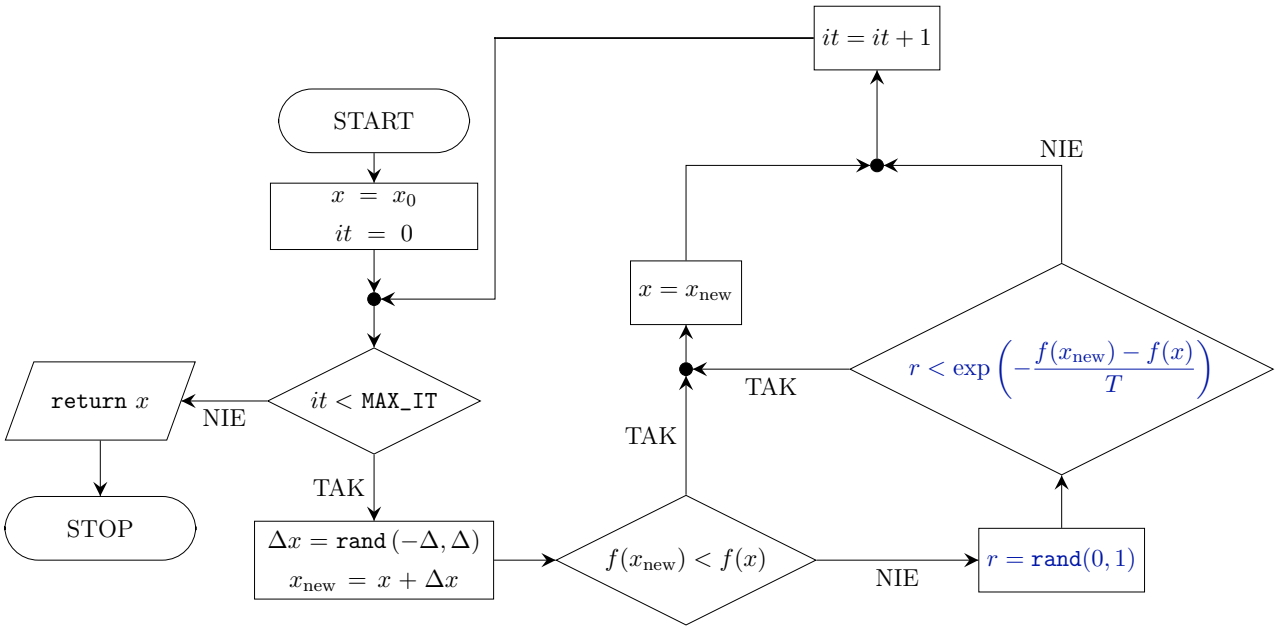
Rysunek 2: Ilustracja działania kilku iteracji algorytmu **Monte Carlo** ze schematu z rys. 1 dla przykładowej funkcji $f(x)$; punkt startowy $x_0 = 2.1$. Kolejne zaakceptowane przybliżenia rozwiązania są zaznaczone punktami. Kolory punktów zmieniają się wraz z iteracjami: od czarnego dla pierwszej iteracji do czerwonego dla ostatniej.

(b) Algorytm Metropolisa–Hastingsa

Problem braku możliwości wyjścia z okolic minimum lokalnego rozwiązuje się poprzez modyfikację algorytmu w przypadku, gdy nie jest spełniona nierówność $f(x_{\text{new}}) < f(x)$. Co prawda x_{new} wydaje się być **gorszym przybliżeniem**, ale **akceptujemy je z pewnym niezerowym prawdopodobieństwem**. Taki zabieg może pozwolić na wydostanie się z okolic minimum lokalnego i w konsekwencji znalezienie minimum globalnego. W **algorytmie Metropolisa–Hastingsa** prawdopodobieństwo akceptacji gorszego wyniku jest dane **rozkładem Boltzmanna**, który w naszym zastosowaniu przybiera postać:

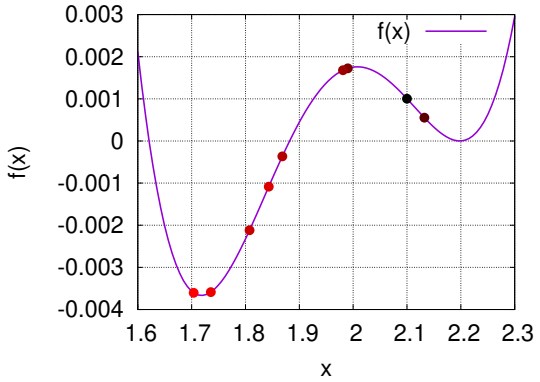
$$\exp\left(-\frac{f(x_{\text{new}}) - f(x)}{T}\right). \tag{1}$$

Stały parametr T jest nazywany **temperaturą** przez nawiązanie do układu, który fluktuuje termicznie. Schemat algorytmu Metropolisa–Hastingsa zamieszczono na rys. 3.

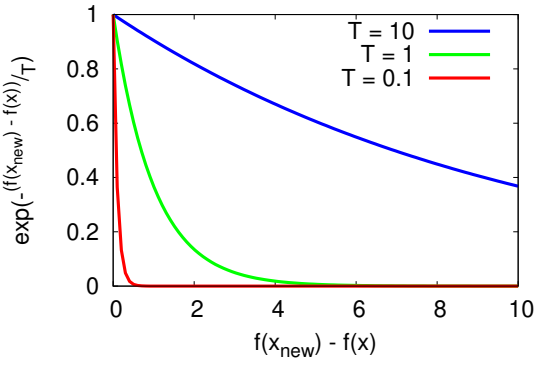


Rysunek 3: Schemat algorytmu **Metropolisa–Hastingsa** – wyszukiwanie minimum funkcji $f(x)$. x_0 jest punktem początkowym, T – zadaną temperaturą; `MAX_IT` to maksymalna liczba iteracji; Δ – wartość definiująca przedział $[-\Delta, \Delta]$; z którego losowane jest aktualne przesunięcie Δx. Wartość r jest losowana z przedziału $[0, 1]$.

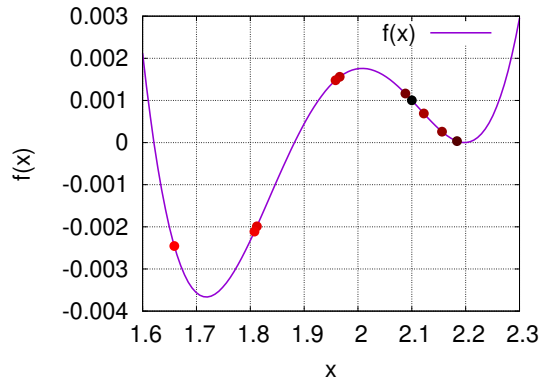
Niezerowa **temperatura** może pozwolić na znalezienie minimum globalnego (jak na rys. 4(a)). Jednak przebieg rozkładu Boltzmanna (zaprezentowany na rys. 4(b)) powoduje, że im większa wejściowa temperatura, tym większe prawdopodobieństwo akceptacji "gorszego" przybliżenia, a co za tym idzie – również wyskoczenia poza okolice minimum globalnego. Z kolei zbyt niska temperatura może nie być wystarczająca, żeby wydostać się z okolic minimum lokalnego. Z tego powodu odpowiedni dobór temperatury jest kluczowy w algorytmie Metropolisa–Hastingsa.



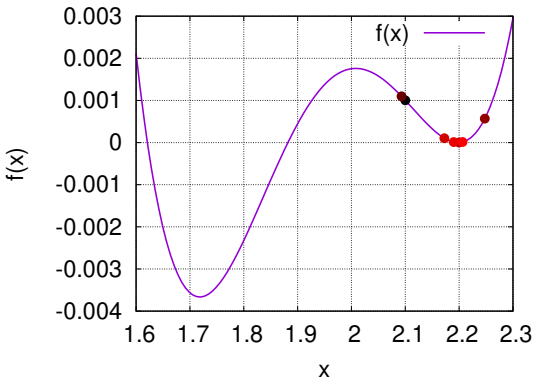
(a) Przypadek 1.: Algorytm zakończył działanie w pobliżu **minimum globalnego**.



(b) **Rozkład Boltzmanna** w funkcji różnicy $f(x_{\text{new}}) - f(x)$.



(c) Przypadek 2.: wejściowy parametr T został ustawiony **zbyt duży**, kolejne przybliżenia "przeskakują" minimum globalne.



(d) Przypadek 3.: parametr T został ustawiony **zbyt mały**, kolejne przybliżenia nie są w stanie wydostać się z okolic minimum lokalnego.

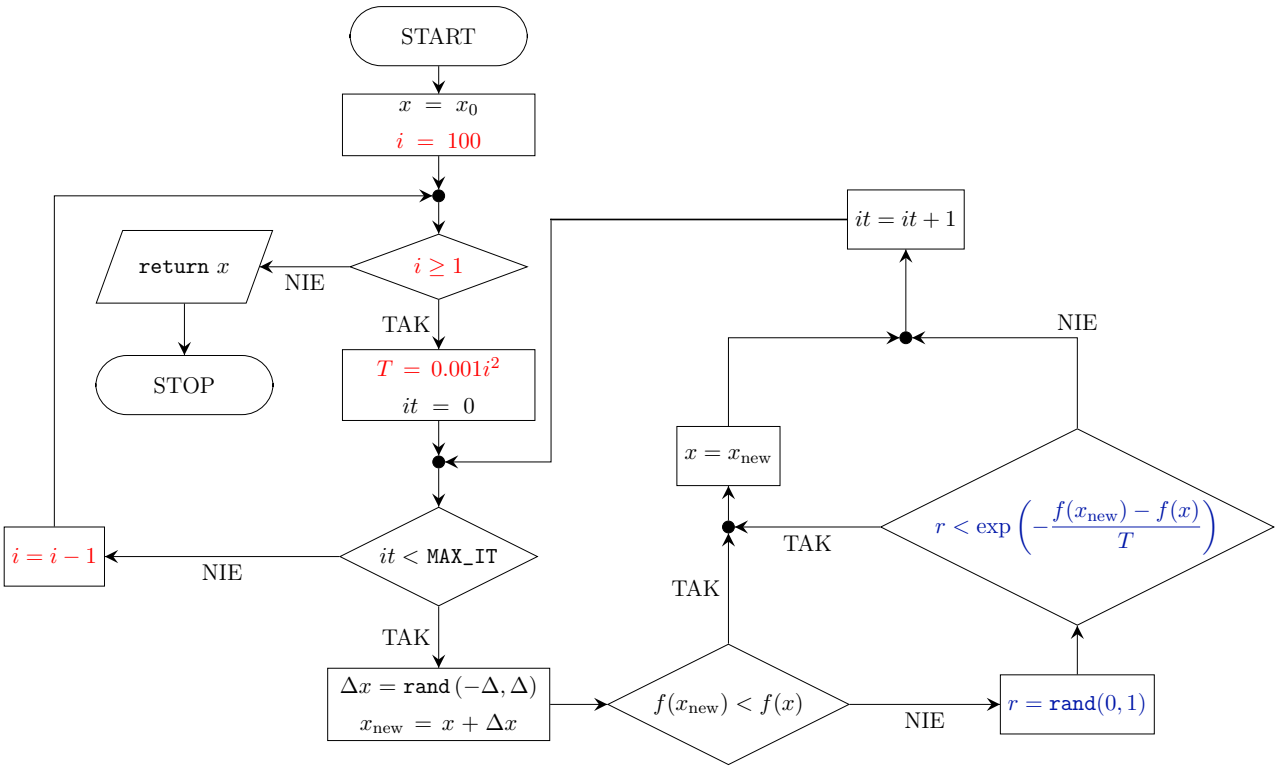
Rysunek 4: (a), (c), (d): Ilustracja działania kilku iteracji algorytmu **Metropolisa–Hastingsa** według schematu z rys. 3. dla przykładowej funkcji $f(x)$; punkt startowy $x_0 = 2.1$. Kolejne zaakceptowane przybliżenia rozwiązania są zaznaczone punktami. Kolory punktów zmieniają się wraz z iteracjami: od czarnego dla pierwszej iteracji do czerwonego dla ostatniej. Wykres (b) prezentuje rozkład Boltzmanna dla trzech różnych wartości bezwymiarowej temperatury T. *Uwaga: przedstawione wykresy jedynie ilustrują problemy w działaniu algorytmu, pomijając wpływ przedziału* $[-\Delta, \Delta]$. *W praktycznym zastosowaniu przedział należy dostosować do rozwiązywanego problemu*

(c) Symulowane wyżarzanie

Problem wyboru odpowiedniej temperatury zostaje rozwiązany w algorytmie **symulowanego wyżarzania**, który jest modyfikacją schematu Metropolisa–Hastingsa o **automatyczny dobór temperatury** poprzez jej stopniowe obniżanie. Dotychczasowe kroki algorytmu zostają objęte dodatkową zewnętrzną pętlą, w której zmienia się temperatura (następuje **schładzanie**). Po zmianie temperatury kolejne iteracje zaczynamy od najlepszego uzyskanego do tej pory przybliżenia.

Schemat z rys. 5. obniża temperaturę zgodnie z przykładową funkcją kwadratową o zadanych

parametrach, jednak sposoby schładzania mogą być różne. Przepis na schładzanie wraz z jego parametrami powinien być wybrany i dostosowany do rozwiązywanego problemu.



Rysunek 5: Schemat **symulowanego wyżarzania** – wyszukiwanie minimum funkcji $f(x)$. x_0 jest punktem początkowym; `MAX_IT` to maksymalna liczba iteracji wykonywanych dla aktualnej temperatury T; Δ – wartość definiująca przedział $[-\Delta, \Delta]$; z którego losowane jest aktualne przesunięcie Δx. Wartość r jest losowana z przedziału $[0, 1]$.

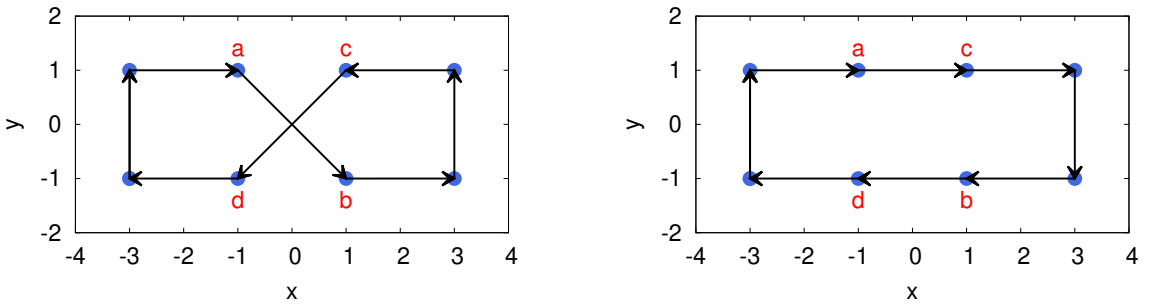
Dzięki automatycznemu obniżaniu temperatury algorytm symulowanego wyżarzania w początkowych, wysokich temperaturach jest w stanie wydostać się z okolic minimów lokalnych, natomiast iteracje wykonywane w niższych temperaturach mają na celu ostateczne udokładnienie przybliżenia przy założeniu, że znajdujemy się już w okolicach minimum globalnego (nie ma jednak gwarancji, że faktycznie tak jest). Często w celu uzyskania najlepszego wyniku algorytm wykonuje się dla **wielu wędrowców jednocześnie**[3]. Idea symulowanego wyżarzania została zaczerpnięta z metalurgicznego procesu usuwania defektów z kryształu podczas jego schładzania.

Przedstawiony **algorytm symulowanego wyżarzania** w zastosowaniu do **problemu komiwojażera** wymaga już niewielu modyfikacji. Celem również jest minimalizacja funkcji: jest nią **funkcja wyznaczająca długość cyklu**. Funkcja ta jest zależna od kolejności n wierzchołków w cyklu; z każdym i-tym wierzchołkiem są związane dwie współrzędne: x_i oraz y_i. Zadane współrzędne wierzchołków znajdują się w pliku `input.dat` na platformie UPeL.

[3]Współrzędną aktualnego przybliżenia minimum funkcji w literaturze często nazywa się położeniem wędrowca. Zaprezentowane w dokumencie schematy zakładają istnienie jednego wędrowca.

Algorytm zaczynamy od **dowolnego początkowego cyklu** P. W każdej iteracji nowy cykl P_{new} wyznaczamy poprzez **operację optymalizacyjną 2-opt**, którą zilustrowano na rys. 6.: zamieniamy losowe dwie krawędzie (a, b) oraz (c, d) na krawędzie (a, c) oraz (b, d). Należy przy tym uważać, żeby nie wylosować krawędzi, które mają wspólny wierzchołek incydentny: mogłoby to doprowadzić do rozspójnienia cyklu.

Następnie zgodnie z algorytmem symulowanego wyżarzania akceptujemy nowy cykl, jeśli jego długość jest mniejsza od długości poprzedniego cyklu. W przeciwnym wypadku akceptujemy go z prawdopodobieństwem zadanym przez rozkład Boltzmanna dla aktualnej temperatury.



(a) Cykl z wylosowanymi krawędziami (a, b), (c, d).

(b) Cykl po zamianie krawędzi na (a, c), (b, d).

Rysunek 6: Ilustracja działania operacji **2-opt**: dla cyklu z rys. (a) wylosowano krawędzie (a, b) oraz (c, d). Nowy cykl wygenerowano przez zamianę tych krawędzi na (a, c) oraz (b, d), jak na rys. (b).