7. kapitola

Kombinatoriálne optimalizačné problémy

7.1 Formulácia základných kombinatoriálnych optimalizačných problémov

Jeden z prvých veľkých úspechov evolučných algoritmov bolo ich použitie pre riešenie notoricky obtiažnych kombinatoriálnych problémov, ktorých CPU čas rastie buď faktoriálovo alebo exponenciálne s rastom dimenzie problému (hovoríme, že sú NP = "**n**on**p**olynomialy" obtiažne [xx]). Budeme študovať dva typy kombinatoriálnych problémov:

Typ 1. Funkcia f je definovaná nad symetrickou grupou S_N zloženou zo všetkých permutácií N objektov

$$f: S_N \to R \tag{7.1}$$

kde permutácie z S_N sú definované ako N-tica celých rôznych čísel

$$P = (p_1, p_2, \ldots, p_N) \tag{7.2}$$

Optimalizačný problém, ktorý je analógom problému (1.2) alebo (1.5), má tvar

$$P_{opt} = \arg \min_{P \in S_N} f(P) \tag{7.3}$$

Riešenie tohto optimalizačného problému je obvykle veľmi obtiažne v dôsledku skutočnosti, že symetrická grupa S_N obsahuje $N!$ permutácií. Z týchto dôvodov môže nastať situácia, že pre veľké N CPU čas zhruba rastie ako $N!$.

Typ 2. Nech R_{set}={1,2,...,r} je množina obsahujúca prvých r celých čísel, jej priamy súčin R_{set}^N obsahuje N-tice

$$\pi = (\pi_1, \pi_2, \ldots \pi_N) \tag{7.4}$$

Jednotlivé komponenty tohto výrazu sú celé čísla z uzavretého intervalu [1,r]. Funkcia

$$f: R_{set}^N \to R \tag{7.5}$$

zobrazuje N-tice π na reálne čísla, analóg optimalizačného problému (1.5) má tvar

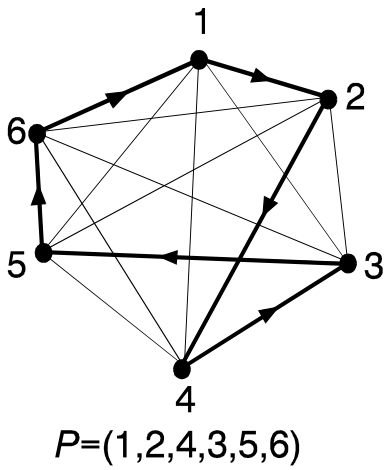
$$\pi_{opt} = arg \min_{\pi \in R_{set}^N} f(\pi) \tag{7.6}$$

V dôsledku toho, že kardinalita množiny R_{set}^N je r^N, optimalizačný problém môže patriť medzi obtiažne s exponenciálnym rastom CPU času pre veľké N. Poznamenajme, že pre r=2 sa tento optimalizačný problém redukuje na obyčajný binárny problém (1.5).

Typ 1 kombinatoriálneho problému bude ilustrovaný známou úlohou obchodného cestujúceho (TSP, Traveling Salesman Problem) [xx]. Jeho grafovo-teoretická formulácia je nasledovná: Nech G je úplný graf obsahujúci N vrcholov a $N(N-1)/2$ hrán (spojov). Každá hrana [i,j] , spájajúca i-ty s j-tym vrcholom (mestom) je ohodnotená kladným číslom $d(i,j)$, ktoré sa interpretuje ako vzdialenosť medzi danými vrcholmi - mestami. Služobná cesta obchodného cestujúceho (Hamiltónov cyklus) navštívi každé mesto len raz, pričom sa vracia do východzieho mesta. Táto cesta je jednoducho určená ako permutácia N objektov - miest (7.2), odpovedá ceste, ktorá je zahájená v meste p_1, potom pokračuje postupne v mestách p_2, p_3, ..., p_N, a na záver sa vráti do východzieho mesta p_1, pozri obr. 7.1. Každej ceste je priradená jej dĺžka

$$f(P) = d(p_1, p_N) + \sum_{i=2}^{N} d(p_{i-1}, p_i) \tag{7.7}$$

Hlavným cieľom úlohy TSP je nájsť takú cestu - permutáciu P_{opt} , ktorá poskytuje minimálnu dĺžku cesty $f_{opt}=f(P_{opt})$.



Obrázok 7.1. Cyklická cesta (Hamiltónov cyklus) na úplnom grafe, ktorý obsahuje 6 vrcholov. Cesta je reprezentovaná permutáciou P=(1,2,4,3,5,6)

Cesta *P* môže byť zmenená na novú cestu *P*' pomocou stochastického operátora mutácie O_{mut} špecifikovaného pravdepodobnosťou P_{mut}. Budeme požadovať, že ak cesta P je permutácia, potom aj nová cesta *P*' je tiež permutácia, a naviac ak pravdepodobnosť $P_{mut}\to 0$, potom permutácie P a *P*' sú identické

$$P \in S_N \Rightarrow P' = O_{mut}(P) \in S_N \tag{7.8a}$$

$$\lim_{P_{mut}\to 0} O_{mut}(P) = P \tag{7.8b}$$

Ako algoritmicky zrealizovať tieto dve podmienky? Jednoduchá metóda algoritmizácie stochastickej transformácie permutácie na permutáciu je vyjadrená pomocou algoritmu 7.1.

```
procedure Permutation_Mutation(input: P; output P');
begin for i:=1 to N do p'i:=pi;
      for i:=1 to N do
      if random<Pmut then
      begin j:=1+random(N);
            aux:=p'i; p'i:=p'j; p'j:=aux
      end;
end;
```

Algoritmus 7.1. Procedúra pre realizáciu stochastickej transformácie permutácie P na permutáciu P', tak aby platili podmienky (7.8a-b). Premenná `random` je náhodné číslo z intervalu [0,1) generované s rovnomernou distribúciou pravdepodobnosti. Podobne, `random(N)` je náhodné celé číslo z intervalu [0,N-1] s rovnomernou distribúciou.

K tomu, aby sa tento typ kombinatotiálnych úloh môhol študovať gneetickým algoritmom, musíme zaviesť ešte operáciu kríženia medzi dvoma permutáciami

$$(P',Q') = O_{cross}(P,Q) \tag{7.9}$$

ktoré zachováva ich charakter permutácie, t.j. P' a Q' sú taktiež permutácie. Študujme dve permutácie *P* a *Q* *n* objektov, vyberme bod kríženia *a* tak, že 1<*a*<*n*

$$P = (p_1,\ldots p_a, p_{a+1},\ldots,p_n) \tag{7.10a}$$

$$Q = (q_1,\ldots q_a, q_{a+1},\ldots,q_n) \tag{7.10b}$$

Keď vymeníme segmenty, ktoré nasledujú za bodom kríženia, dostaneme dva nové objekty

$$\hat{P} = (p_1,\ldots p_a, q_{a+1},\ldots,q_n) \tag{7.11a}$$

$$\hat{Q} = (q_1,\ldots q_a, p_{a+1},\ldots,p_n) \tag{7.11b}$$

Žiaľ, takto vytvorené objekty nemusia byt už permutácie. Môže nastať situácia, že nejaké celé číslo sa v objektoch $\hat{P}$ alebo $\hat{Q}$ vyskytuje dvakrát. Z tohto dôvodu je potrebné aplikovať "opravný proces" [xx] (nazývaný čiastočné priradenie - partial matching), ktorý z objektov $\hat{P}$ a $\hat{Q}$ vytvorí normálne permutácie (poznamenajme, že v literatúre [xx] je popísaných mnoho rôznych druhov kríženia dvoch permutácií a spôsobov ich opráv). Definujme zobrazenie f_{PQ} tak, že komponenty permutácie $\hat{P}$, ktoré boli vymenené, sú zobrazené na komponenty permutácie $\hat{Q}$, ktoré sa tiež zúčastnili výmeny

$$f_{PQ}: p_i \to q_i \quad (\text{pre } i > a) \tag{7.12}$$

Druhé zobrazenie f_{QP} je určené ako inverzné zobrazenie k f_{PQ}, $f_{QP} = f_{PQ}^{-1}$. Pomocou týchto dvoch zobrazení budeme opravovať "pôvodné" časti v $\hat{P}$ a $\hat{Q}$, t.j. tie, ktoré ležia pred bodom kríženia. Ak komponenta p_i (pre $1 \le i \le a$) nepatrí do definičného oboru zobrazenia f_{QP} (to znamená tiež, že p_i sa vyskytuje v $\hat{P}$ práve raz) potom p_i zostáva nezmenené. V opačnom prípade, ak p_i sa nachádza v definičnom obore zobrazenia f_{QP}, potom p_i sa zamení za prvú hodnotu iteračnej schémy $x_{k+1}=f_{QP}(x_k)$, ktorá je mimo definičnej oblasti funkcie f_{QP} (iteračné riešenie je inicializované $x_0=p_i$). Podobný postup je aplikovaný aj pre opravu pôvodného segmentu v $\hat{Q}$, avšak teraz bude použité zobrazenie f_{PQ}.

Pre ilustráciu študujme kríženie medzi dvoma permutáciami

$$P = (3,4,5,1,2,6,10,8,9,7) \text{ a } Q = (1,2,6,10,8,3,4,5,7,9)$$

predpokladajme, že bod kríženia je a=4. Ak vymeníme medzi dvoma permutáciami segmenty po bode kríženia dostaneme

$$\hat{P} = (3,4,5,1,8,3,4,5,7,9) \text{ a } \hat{Q} = (1,2,6,10,2,6,10,8,9,7)$$

Vidíme, že objekty $\hat{P}$ a $\hat{Q}$ nie sú permutácie, pretože obsahujú niektoré indexy dvakrát. K oprave týchto objektov na permutácie zostrojíme zobrazenia f_{QP} a f_{PQ}

$$f_{PQ} = \begin{pmatrix} 2\,6\,10\,8\,7\,9 \\ 8\,3\;\;4\,5\,9\,7 \end{pmatrix} \text{ a } f_{QP} = \begin{pmatrix} 8\,3\;\;4\,5\,9\,7 \\ 2\,6\,10\,8\,7\,9 \end{pmatrix}$$

Objekt $\hat{P}$ opravíme pomocou zobrazenia f_{QP} a objekt $\hat{Q}$ pomocou zobrazenia f_{PQ}. Prvé tri indexy v P' zameníme za $3\to6$, $4\to10$ a $5\to8\to2$. Podobne, prvé indexy 2, 6 a 10 v Q' zameníme za $2\to8\to5$, $6\to3$ a $10\to4$, dostaneme opravené objekty, ktoré už sú permutácie a sú považované za výsledok výmeny medzi permutáciami P a Q

$$P' = (6,10,2,1,8,3,4,5,7,9) \text{ a } Q' = (1,5,3,4,2,6,10,8,9,7)$$

Kombinatoriálne úlohy typu 2 budú ilustrované úlohou známou pod názvom problém rozkladu čísla (NPP, Number Partitioning Problem). Nech $Q=\{q_1,q_2,...,q_N\}$ je množina obsahujúca N kladných reálnych čísel a nech π je zobrazenie Q na množinu $R_{set}=\{1,2,...,r\}$

$$\pi: Q \to R_{set} \tag{7.13}$$

Toto zobrazenie π môže byť jednoznačne vyjadrené ako N-tica $\pi = (\pi_1, \pi_2, \ldots, \pi_N) \in R_{set}^N$. Jej jednotlivé komponenty sú interpretované tak, že celé číslo $\pi_i \in R_{set}$ je priradené objektu $q_i \in Q$. Vzhľadom k tomuto zobrazeniu množina Q môže byť rozložená na disjunktné podmnožiny

$$Q = Q_1 \cup Q_2 \cup \ldots \cup Q_r \tag{7.14a}$$

kde

$$Q_i = \{ q \in Q; \pi(q) = i \} \tag{7.14b}$$

Podmnožina Q_i obsahuje všetky elementy - čísla množiny Q, ktoré sú zobrazené funkciou π na celé číslo i. Definujme účelovú funkciu

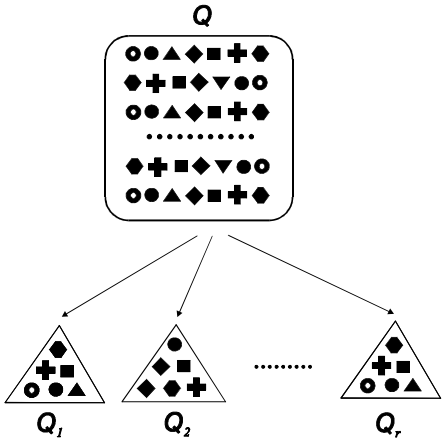
$$f(\pi) = \max_i \sum_{q \in Q_i} q - \min_i \sum_{q \in Q_i} q \tag{7.15}$$

Vyjadruje rozdiel medzi maximálnou a minimálnou sumou čísel z podmnožín $Q_1, Q_2, \ldots, Q_N$. Cieľom NPP úlohy je hľadať také zobrazenie π, ktoré minimalizuje účelovú funkciu f

$$\pi_{opt} = \arg \min_{\pi \in R_{set}^N} f(\pi) \tag{7.16}$$

Ako názorne interpretovať túto úlohu? Predstavme si, že máme hromadu obsahujúcu N vecí, pričom každej veci je priradená cena $q_1,q_2,...,q_N$. Cieľom je rozdeliť túto kopu vecí na r hromád tak, aby

ich ceny (súčty cien jednotlivých vecí patriacich do tej ktorej hromady) vykazovali minimálne rozdiely medzi sebou, pozri obr. 7.2.



Obrázok 7.2. Cieľom NPP úlohy je rozdeliť hromadu (množinu) *Q* na r kôp $Q_1,Q_2,...,Q_r$ tak, aby diferencie medzi ich cenami boli minimálne. Predstavme si partiu *r* zlodejov, ktorá nakradla hromadu Q vecí, pričom každá táto vec má určitú cenu. Po čase si zlodeji chcú nakradnuté veci rozdeliť tak, aby každý dostal veci o približne rovnakej hodnoty. Táto úloha je totožná s úlohou NPP.

K zobrazeniu π môžeme priradiť iné zobrazené π' pomocou operátora mutácie, $\pi'=O_{mut}(\pi)$ podobným spôsobom ako pri binárnej reprezentácii, pozri algoritmus 7.2, pričom pre $P_{mut}\rightarrow 0$ je vytvorené zobrazenie π' totožné s pôvodným zobrazením π.

```
procedure Mapping_Mutation(input:π; output:π' );
begin for i:=1 to N do
      if random<P_mut then
      π'_i :=1+random(r) else π'_i :=π_i;
end;
```

Algoritmus 7.2. Pseudopascalovská procedúra realizujúca mutáciu zobrazenia π na nové zobrazenie π'. Premenné `random` a `random(r)` sú definované podobne ako v algoritme 7.1.

Operácia kríženia pre tento typ chromozómov je veľmi jednoduchá a môže sa priamo stotožniť s podobnou operáciou pre binárne vektory (pozri kapitolu 3.3).

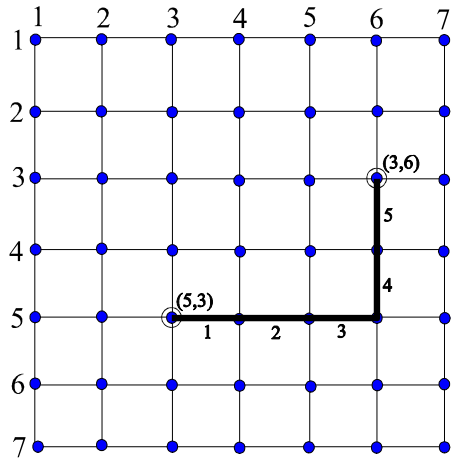
Príklad 7.1. *V tejto kapitole je popísaný prístup k riešeniu kombinatoriálnych úloh pomocou permutácií a n-tíc prirodzených čísel, ktorý sa líši od štandardnej binárnej reprezentácie riešení. Ukážeme, že aj v tomto prípade je binárna reprezentácia stále použiteľná.*

Permutácia N objektov(kombinatoriálne úlohy typu 1) môže byť zadaná ako binárny vektor dĺžky kN, kde k je také celé číslo, ktoré zabezpečuje, aby binárny vektor dĺžky k bol schopný reprezentovať číslo N. Takýto binárny vektor je ľahko pretransformovateľný postupom z kapitoly 1.2 na vektor N celých čísel z intervalu $[0,2^k-1]$*. Potom permutácia P je definovaná tak, že celočíselné komponenty sú usporiadané do nerastúcej alebo neklesajúcej postupnosti. Nech N=5, k=3, binárny vektor (101|011|001|111|101), jeho celočíselná verzia má tvar (5,4,1,7,5) (pozri Tab. 1.1). Nech permutácia P je určená tak, že preusporiada túto N-ticu do rastúcej postupnosti, potom P=(3,2,1,5,4). Nevýhodou tohto prístupu pre kódovanie permutácií je, že rovnaká permutácia je určená rôznymi binárnymi reťazcami, čo môže spôsobovať pomerne malú efektívnosť evolučných algoritmov aplikovaných na riešenie kombinatoriálnych algoritmov. Napíšte program, ktorý binárny vektor pretransformuje na permutáciu N objektov. Kritickým miestom programu bude program pre usporiadanie N-tice čísel klesajúcej alebo rastúcej postupnosti, preto použite nejaký efektívny algoritmus pre usporiadanie (napr. quicksearch).*

Binárna reprezentácia N-tic celých čísel z množiny $R_{set}^{N} = \{1,2,\ldots,r\}^{N}$ (kombinatoriálne úlohy typu 2) môže byť priamo reprezentovaná binárnym vektorom dĺžky kN, kde k je určené podmienkou $r \le 2^{k}-1$. Nech binárny podreťazec dĺžky k reprezentuje reálne číslo z intervalu [1,r], k reálnemu číslu x z tohto intervalu priradíme celé číslo z intervalu [1,r] operáciou $\lfloor x \rfloor$, ktorá sa nazýva "zaokrúhlenie" (napr. $\lfloor 1.1 \rfloor = 1$ a $\lfloor 1.8 \rfloor = 2$. Pre ilustráciu uvažujme N=5, k=3 a r=4 a rovnaký binárny vektor ako v prechádzajúcom príklade (101|011|001|111|101). Pomocou transformácie (1.16) vektor s reálnymi komponentami má tvar (3.143, 2.286, 1.429, 4.000, 3.143). Aplikovaním operácie zaokrúhlenia dostaneme vektor celých čísel (3,2,1,4,3). Napíšte program, ktorý binárny vektor pretransformuje na N-ticu celých čísel z intervalu [1,r].

Príklad 7.2. *Napíšte programy pre vybrané evolučné algoritmy popísané v predchádzajúcich kapitolách, ktoré riešia úlohu obchodného cestujúceho (Traveling Salesman Problem). Použijte oba prístupy na kódovanie cesty, tak priamo pomocou permutácie, ako aj pomocou binárneho reťazca (pozri príklad 7.1). Porovnajte efektívnosť týchto dvoch rôznych prístupov.*

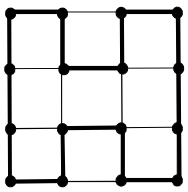
Verifikácia výsledkov pre obchodného cestujúceho nie je jednoduchou záležitosťou. Preto sa navrhujú také príklady, kde sme schopní presne odhadnúť dĺžku optimálnej cesty. Často sa študuje ortogonálne mriežka bodov typu N×N, pričom vzdialenosť medzi jednotlivými bodmi sa počíta pomocou Hammingovej (L_1) vzdialenosti. Na obrázku je znázornená mriežka typu 7×7, ktorá obsahuje 49 bodov. Hammingova vzdialenosť (D_{ij}) medzi dvoma bodmi s riadkovými-stĺpcovými indexami (5,3) a (3.6) je |5-3|+|3-6|=5. Dva susedné body ležiace na rovnakom riadku alebo stĺpci majú jednotkovú vzdialenosť.



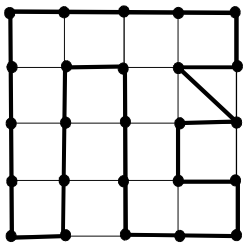
Pre tento špeciálny prípad optimálna vzdialenosť je

$$f_{opt} = \begin{cases} N^2 & (\text{pre párne } N) \\ N^2 + 1 & (\text{pre nepárne } N) \end{cases}$$

Táto formula môže byť jednoducho verifikovaná pomocou matematickej indukcie. Dva jednoduché ilustratívne príklady sú znázornené na nasledujúcich obrázkoch.



N=4, f_{opt}=16



N=5, f_{opt}=26

Optimalizujte parametre metód pre rôzne N=4,5,...,10, tak, aby vypočítaná minimálna dĺžka cesty na ortogonálnej mriežke bodov bola rovná teoretickej minimálnej vzdialenosti skoro so 100% pravdepodobnosťou.

Príklad 7.3*. Napíšte programy pre vybrané evolučné algoritmy, ktoré riešia úlohu rozkladu čísla (Number Partitioning Problem). Použijte oba prístupy na kódovanie zobrazenia π, tak priamo pomocou N-tice celých čísel, ako aj pomocou binárneho reťazca (pozri príklad 7.1). Porovnajte efektívnosť týchto dvoch rôznych prístupov.*

Podobne, ako v predchádzajúcom príklade, navrhneme jednoduchý príklad umožňujúci priamu verifikáciu správnosti (optimálnosti) získaných riešení. Nech úloha je určená parametrami N=100 a

r=10. To znamená, že počet všetkých možných zobrazení π je 10^{100}. K tomu, aby sa nám podarilo určiť optimálne riešenie, predpokladajme, že Q obsahuje číslo 1 až 10 10-krát. Optimálny rozklad Q na 10 podmnožín Q_1, Q_2, ..., Q_{10} je taký, že každá z týchto podmnožín obsahuje čísla 1 až 10 práve 1-krát. Potom cena každej podmnožiny je

$$\sum_{q \in Q_i} q = 1+2+\ldots+10 = 55$$

pre i=1,2,...,10, preto $f(\pi_{opt})=0$. Optimalizujte parametre metód tak, aby získaná hodnota účelovej funkcie bola nulová skoro so 100% pravdepodobnosťou.

***Príklad 7.4**. Napíšte program pre metódu zakázaného hľadania (pozri kapitolu 1.6) aplikovanú na riešenie obchodného cestujúceho (TSP), pričom cesta nech je vyjadrená pomocou permutácie. Transformačné operátory $t \in S$ sú realizované ako permutácie transpozícií dvoch objektov. Operátor t_{ij} pôsobí na permutáciu P ako transpozícia objektov i a j,*

$$t_{ij}P = t_{ij}\left(p_1,\ldots,p_i,\ldots,p_j,\ldots,p_N\right) = P' = \left(p_1,\ldots,p_j,\ldots,p_i,\ldots,p_N\right)$$

pre i<j. Množina S potom obsahuje N(N-1)/2 transformácií. Inverzná transformácia t_{ij}^{-1} k transformácii t_{ij} je totožná s touto transformáciou, $t_{ij}^{-1}=t_{ij}$. Okolie U(P) priradené riešeniu - permutácii P obsahuje N(N-1)/2 elementov

$$U(P) = \left\{P' = tP; \forall t \in S\right\}$$

Ako testovací príklad použite ortogonálnu mriežku bodov - miest, kde sa vzdialenosť počíta pomocou Hammingovej metriky (pozri príklad 7.2). Pokúste sa o optimalizáciu veľkosti "s" zakázaného zoznamu T (tabu listu) a maximálneho počtu iterácií $time_{max}$ pre rôzne hodnoty N dimenzie ortogonálnej mriežky bodov, N=4,5,...,10, tak , aby ste s 95% pravdepodobnosťou dostali korektný výsledok (špecifikovaný v príklade 7.2). Podobné úvahy vykonajte aj pre suboptimálne výsledky, ktoré sa líšia len niekoľko málo jednotiek od optimálneho výsledku. Zistíte, že metóda zakázaného hľadanie poskytuje veľmi rýchle suboptimálne výsledky, ktoré sú veľmi blízke optimálnym.

***Príklad 7.5. M**etóda zakázaného hľadania (a nielen táto metóda) je **č**asto v literatúre ilustrovaná známou jednoduchou kombinatoriálnou úlohou nájsť na šachovnici N×N postavenie N dám tak, aby v každom riadku, stĺpci a diagonále bola umiestnená práve jedna dáma (t.j. neexistuje kolízne postavenie medzi dámami). Zložitosť riešenia tohto problému silne závisí od zvolenej reprezentácie postavenia dám na šachovnici. Jednoduchá reprezentácia postavenia dám je pomocou permutácie $P=(p_1, p_2, \ldots, p_N)$. Potom dámy na šachovnici sú umiestnené takto: v 1. stĺpci na p_1-tom riadku, v 2. stĺpci na p_2-tom riadku, atď. Táto jednoduchá reprezentácia rozmiestnenia dám na šachovnici pomocou permutácií má výhodu v tom, že v každom stĺpci a riadku šachovnice sa nachádza práve jedna dáma. Každej permutácii N objektov priradíme celé nezáporné číslo col(P), ktoré odpovedá počtu diagonálnych kolízií. Riešenie problému je potom taká permutácia P, ktorá má nulový počet kolízií, col(P)=0. Funkcia col(P) je určená vzťahom*

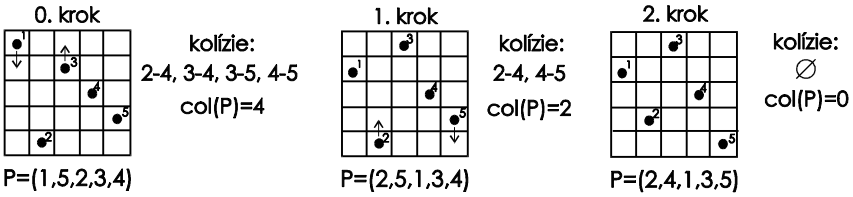
$$col(P) = \sum_{\substack{i,j=1 \\ (i<j)}}^{N} \left[\delta\left(p_i + j - i, p_j\right) + \delta\left(p_i - j + i, p_j\right)\right]$$

kde $\delta(i,j)$ je Kroneckerovo delta, $\delta(i,j)=1$, pre $i=j$, $\delta(i,j)=0$, pre $i \neq j$. Výpočet zmeny počtu kolízií pri prechode od permutácie P k novej permutácii $P'=t_{ij}P$ v rámci metódy zakázaného hľadania je určený formulou

$$col(P') - col(P) = \sum_{\substack{k=1 \\ (k \neq i,j)}}^{N} \left[\delta\left(p_k + i - k, p_j\right) + \delta\left(p_k - i + k, p_j\right) + \delta\left(p_k + j - k, p_i\right) + \delta\left(p_k - j + k, p_i\right)\right.$$
$$\left. - \delta\left(p_k + i - k, p_i\right) - \delta\left(p_k - i + k, p_i\right) - \delta\left(p_k + j - k, p_j\right) - \delta\left(p_k - j + k, p_j\right)\right]$$

Numerická náročnosť výpočtu tejto formule je podstatne menšia, ako predchádzajúcej formule. Preto je výhodnejšie počítať počet kolízií col($t_{ij}P$) pomocou tejto formule, za predpokladu, že počet kolízií col(P) je známy.

Program z predchádzajúceho príkladu 7.4 upravte tak, aby riešil úlohu N dám. Pre ilustráciu tohto algoritmu uvádzame prvé tri iteračné kroky zakázaného hľadania pri riešení úlohy pre N=5. V nultom kroku bolo náhodne vygenerované postavenie dám na šachovnici, ktoré má 4 kolízie. Nová permutácia P v 1. kroku vznikla z predchádzajúcej permutácie použitím transpozície t_{13}, počet kolízií sa znížil na 2. V 2. kroku použitím transpozície t_{25} dostaneme novú permutáciu, ktorá už má nulový počet kolízií, čiže je riešením problému.



***Príklad 7.6**. Napíšte program pre metódu zakázaného hľadania riešiaceho úlohu N dám na šachovnici N×N, pričom permutácia P je binárne reprezentovaná (pozri príklad 7.1). Porovnajte efektívnosť tohto programu s programom z predchádzajúceho príkladu 7.4.*

***Príklad 7.7.** Riešte úlohu obchodného cestujúceho pomocou genetického algoritmu, pričom cesta obchodného cestujúceho je zadaná pomocou permutácie. Ako cvičné príklady riešte konštrukciu optimálnej cesty na ortogonálnej mriežke bodov s Hammingovou vzdialenosťou (pozri príklad 7.2). Pokuste sa navrhnúť novú operáciu kríženia medzi dvoma permutáciami, tak, aby výsledné objekty - potomkovia boli znovu permutáciami.*