INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

**VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA**
**FAKULTA STROJNÍ**


# APLIKOVANÁ INFORMATIKA

## 8. LEKCE – ALGORITMY ŘEŠENÍ KOMBINATORICKÝCH ÚLOH

prof. Ing. Radim Farana, CSc.

**Ostrava 2013**


ISBN 978-80-248-3017-9

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

# OBSAH

# 8 ALGORITMY ŘEŠENÍ KOMBINATORICKÝCH ÚLOH

**OBSAH KAPITOLY:**



**MOTIVACE:**

Řada úloh, které v technické praxi programově řešíme, je popsána jako výběr optimální varianty, která je závislá na konkrétních hodnotách parametrů. K určení, jak kvalitní jsou jednotlivá řešení, je definována ohodnocující funkce neboli účelová funkce (pokud známe pouze funkci, která popisuje kvalitu řešení přibližně, nazýváme ji heuristika). Pokud neznáme vztahy mezi jednotlivými parametry a přitom jak počet parametrů je konečný a jejich obor hodnot je dán množinou hodnot, řešíme úlohu jako kombinatorický problém.

**CÍL:**

Umět popsat kombinatorickou úlohu a definovat její omezení. Seznámit se se základními neheuristickými algoritmy – prohledáváním do šířky a do hloubky a prohledáváním do hloubky s omezenou hloubkou vnoření. Umět použít heuristické algoritmy prohledávání. Seznámit se pravděpodobnostními přístupy a základy genetických algoritmů. Znát principy a použití genetických algoritmů.

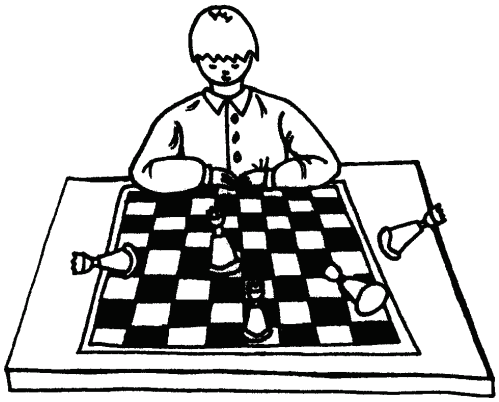
Řada úloh, které v technické praxi programově řešíme, je definována jako výběr ***optimální*** (nejlepší) ***varianty***, která je závislá na konkrétních hodnotách parametrů. K určení, jak kvalitní je které řešení, tedy můžeme definovat ***ohodnocující funkci*** neboli ***účelovou funkci*** (pokud známe pouze funkci, která popisuje kvalitu řešení přibližně, nazýváme ji ***heuristika***):

$$f(p_1, p_2, ..., p_n), \tag{8.1}$$

kde je $f$ - ohodnocující funkce,

$p_i$ - $i$-tý parametr.

Jak vidíme, konkrétní řešení je určeno konkrétními hodnotami jednotlivých parametrů. Kvalita řešení je pak dána hodnotou ***ohodnocující funkce*** (***heuristiky***) pro tyto parametry. Beze ztráty obecnosti můžeme předpokládat, že lepší řešení má vyšší hodnotu ohodnocující funkce. Při řešení problému manipulujeme s hodnotami parametrů tak, abychom dosáhli maximální (optimální) hodnoty ohodnocující funkce. Jedná se tedy o úlohu ***maximalizace ohodnocující funkce***. Tato problematika je široce řešena teoreticky, existuje řada optimalizačních metod (analytické, numerické). My se v dalším textu budeme zabývat vybranými speciálními metodami řešení uvedeného problému, které využívají postupů, uvedených v předchozích kapitolách.



**Obrázek 8.1 Řešení kombinatorických úloh**

## 8.1 TŘÍDA ŘEŠENÝCH PROBLÉMŮ

Pro naše potřeby provedeme následující zjednodušení:

1. **Omezení počtu parametrů**. Budeme pracovat jen s ohodnocujícími funkcemi, které mají známý počet parametrů. Dále budeme předpokládat, že hodnoty těchto parametrů jsou vzájemně nezávislé nebo se sice ovlivňují, ale nejsme schopni definovat funkční závislosti parametrů.
2. **Omezení oboru hodnot parametrů**. Budeme pracovat jen s parametry, jejichž hodnota je omezena povoleným rozsahem (oborem) hodnot. Dalším zjednodušením bude omezení oboru hodnot parametru na množinu hodnot. Budeme tedy pracovat s diskrétními parametry.

Na základě zavedených omezení se dostáváme do situace, kdy vlastně řešíme problém, jehož jednotlivá řešení jsou dána kombinací hodnot jednotlivých parametrů. Proto se tyto úlohy často nazývají ***kombinatorické***. V dalším textu rozebereme některé algoritmy řešení těchto problémů. Nejprve si však uvědomíme, kolik existuje různých řešení dané úlohy. Počet je dán následujícím vztahem:

$$r = \prod_{i=1}^{n} m_i \,, \tag{8.2}$$

kde je $r$ - počet možných řešení,

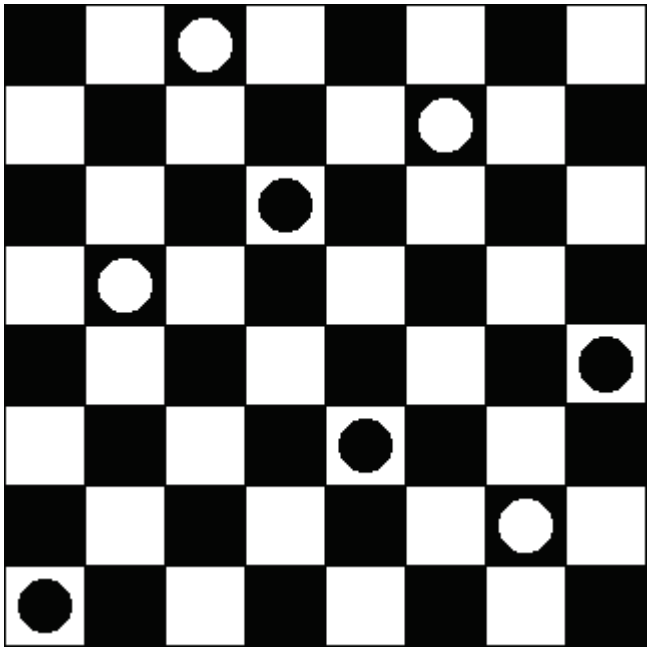
$m_i$ - počet možných hodnot parametru $p_i$,

$n$ - počet parametrů.

Počet různých řešení tedy s rostoucím počtem parametrů roste geometricky. Zde je základní potíž. Protože řešení jsou vzájemně nezávislá, měli bychom vyzkoušet všechna, abychom s plnou jistotou našli optimální řešení. V řadě algoritmů tomu tak naštěstí není. Abychom mohli ukázat zvolené metody řešení, vybereme nyní příklad, který budeme řešit.

*Příklad:*

Jedná se o úlohu rozestavení 8 dam na šachovnici tak, aby se vzájemně neohrožovaly. Při popisu úlohy vyjdeme ze skutečnosti, že v každém sloupci (1 ÷ 8) musí ležet právě jedna dáma.



**Obrázek 8.2 Řešení úlohy rozložení osmi dam na šachovnici**

Tato dáma musí ležet na jednom z řádků (1 ÷ 8). Úloha je tedy popsána pomocí osmi parametrů, které nabývají hodnot z množiny {1, 2, 3, ....., 8}. Ohodnocující funkci tedy můžeme definovat jako počet dam, které neohrožují jinou dámu a řešení nalezneme pro:

$$f(p_1, p_2, ....., p_8) = 8, \tag{8.3}$$

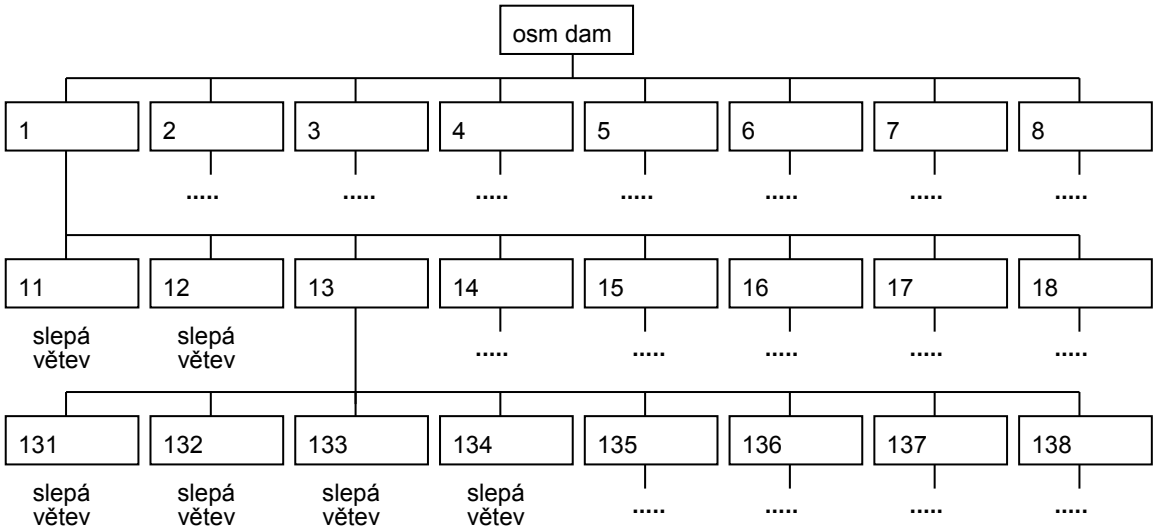
kde je $f$ - ohodnocující funkce,

$p_i$ - i-tý parametr (pozice dámy na i-tém sloupci).

Vyhodnocení ohodnocující funkce poněkud zjednodušíme tím, že budeme zkoumat dámy od sloupce 1 do 8 a jako správně umístěnou dámu považujeme tu, která neohrožuje žádnou dámu ležící na sloupci nižšího čísla. Počet všech možných rozložení dam určíme dle vzorce (8.4):

$$r = \prod_{i=1}^{8} 8 = 8^8 = 16777216 \doteq 17.10^6 \tag{8.4}$$

Jak vidíme, existuje skoro 17 miliónů různých rozložení. Z nich však jen některá jsou správná. Znázorněme si strom jednotlivých rozložení. Na každé nižší úrovni vždy přidáme jeden parametr. Je zřejmé, že tím rozvineme vždy 8 větví stromu. Hodnoty parametrů budeme pro jednoduchost zapisovat za sebou.




**Obrázek 8.3 Strom všech možných rozložení dam**

Jak vidíme na obr.8.3, ve stromu existuje řada slepých větví, které nemá smysl procházet, neboť v nich řešení nemůže existovat. Tuto skutečnost se pokusíme také využít.

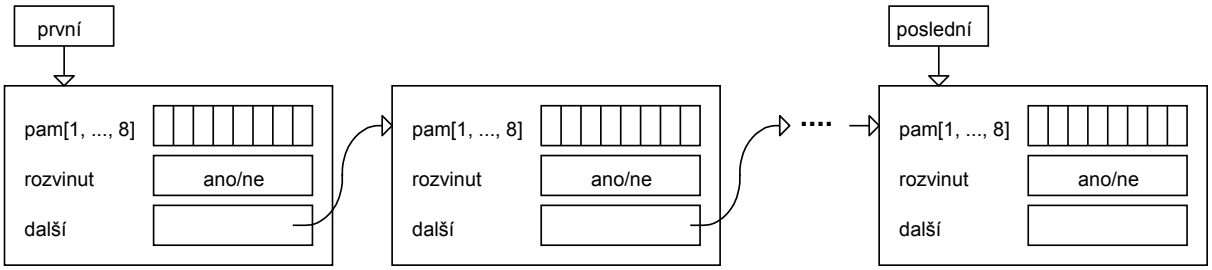
Nyní však již přejděme k jednotlivým algoritmům hledání řešení.

## 8.2 ALGORITMY NEHEURISTICKÉ

Tyto algoritmy vycházejí z předpokladu, že ohodnocující funkce (heuristika) jednoho řešení je zcela nezávislá na jeho poloze ve stromu řešení. To znamená, že řešení může ležet na jakékoliv úrovni vnoření do stromu a ve kterékoliv větvi. Jak víme, v námi řešeném příkladu to není pravda. Proto se budou dále uvedené metody jevit jako neefektivní. Uvádíme je však, neboť v jiných úlohách mohou mít své místo.

### 8.2.1 Prohledávání do šířky

Algoritmus ***prohledávání do šířky*** se snaží odstranit nebezpečí velmi hlubokého vnoření do stromu rozložení, pokud bychom se vydali po slepé větvi. Skutečnost, že je větev slepá, totiž zjistíme až vyzkoušením všech řešení, která na ní leží. Současně se snaží dosáhnout řešení co nejrychleji. Postupuje tak, že vždy rozvine všechny prvky na dané úrovni vnoření. Pokud se mezi nimi nenachází řešení, rozvine prvky další úrovně. Je zřejmé, že algoritmus má velké nároky na paměť, protože musíme pro každý prvek stromu rozložení, který použijeme, uložit informaci o parametrech řešení a pro rychlejší zpracování nejlépe také hodnotu ohodnocující funkce:




**Obrázek 8.4 Datová struktura pro uložení vytvořených rozložení**

V paměti budeme vytvářet ***lineární seznam***, ukazovátka ***první*** a ***poslední*** k němu umožňují přístup. Tím obcházíme nutnost zjistit předem řád stromu a vytvořit příslušnou stromovou strukturu. Je zřejmé, že pokud budeme chtít nalézt všechna řešení, dojde v konečném stavu k rozvinutí celého stromu rozložení. To obvykle není možné zajistit. Požadavky na paměť by byly příliš velké. Proto se tento algoritmus používá především tam, kde hledáme jedno řešení nebo úloha sama o sobě má jedno optimální řešení (maximum ohodnocující funkce), a to má nepříliš velkou hloubku vnoření do stromu. Postup práce pak můžeme zapsat:

```
počáteční nastavení
while není nalezeno řešení
  rozviň všechny nerozvinuté prvky a označ je jako
rozvinuté
  spočti ohodnocující funkci všech nových prvků
end while
vyhlas nalezené řešení
```

V paměti sice vytvoříme lineární seznam, ale budeme jím reprezentovat rozvíjení stromu rozložení. Pokud bychom chtěli vytvořit datovou strukturu odpovídající skutečně našemu stromu, pak by každý prvek musel obsahovat osm ukazovátek na další prvek, řada z nich by přitom nebyla využita (pokud nebyl prvek rozvinut) a hledání vhodného prvku k rozvinutí by bylo komplikovanější.

Z ukázky vidíme **výhody** algoritmu:

dobrá rychlost,

nehrozí nebezpečí nadměrného vnoření do stromu.

Musíme však upozornit také na **nevýhody**:

velké nároky na paměť,

složitější manipulace s datovou strukturou.

*Poznámka:*

Námi vytvořená datová struktura má zvláštní vlastnost, každý prvek nese informaci o tom, jak se k němu ve stromu rozložení dostat. Proto můžeme prvky, které již byly rozvinuty, z paměti odstranit. Manipulace se stromem bude složitější, ale ušetříme paměť.

### 8.2.2 Prohledávání do hloubky

Algoritmus ***prohledávání do hloubky*** se snaží odstranit velké paměťové nároky algoritmu prohledávání do šířky. V každém kroku je zvolen jeden prvek, ten je rozvinut a řešení je hledáno na nižší úrovni. Pokud není řešení nalezeno, je opět rozvinut některý z prvků s dosud největší hloubkou vnoření. Teprve pokud dorazíme k nejvyššímu možnému vnoření a řešení nenalezneme, vracíme se zpět a zkoušíme jinou větev stromu řešení. Pokud však řešení

neexistuje, dojdeme do stejného stavu jako při použití algoritmu prohledávání do šířky a tedy také ke stejným paměťovým nárokům.

Jako **výhody** algoritmu můžeme uvést:

postupujeme rychle do hloubky,

potřebujeme relativně málo paměti k provedení algoritmu.

Opět však nesmíme opomenout **nevýhody**:

pokud se řešení nachází blízko středu stromu, budeme ho hledat poměrně dlouho,

pokud nevíme, na jaké úrovni vnoření se řešení nachází a zejména pokud úroveň vnoření není omezena, může se stát, že budeme postupovat do velkých hloubek stromu bez úspěchu. Tento problém se někdy řeší ***prohledáváním do hloubky s omezenou hloubkou vnoření*** (s navracením). Jakmile dojdeme do určené hloubky, vracíme se zpět, bez ohledu na kvalitu nalezených řešení. V řešeném příkladu bychom nejspíš pracovali s maximální hloubkou = 8.

### 8.2.3 Vyzkoušení všech kombinací parametrů

Tento postup je na místě tam, kde nevíme nic o vlastnostech úlohy, a v plné míře platí, že neexistuje žádný vztah mezi hodnotami parametrů. Z vlastností stromu rozložení přitom víme, že řešení leží jedině na nejhlubší úrovni vnoření (na koncových prvcích). To znamená, že musíme vzít v úvahu všechny parametry. Přitom se budeme snažit zbavit velkých nároků na dostupnou paměť.

Postupně tedy zkoušíme všechna rozložení dam. V nejhorším případě vyzkoušíme všechna a teprve potom můžeme říci, že úloha nemá řešení. Pro vlastní realizaci využijeme vektor počítadel s hodnotami 1 ÷ 8. Vždy zvětšíme hodnotu posledního, pokud přesáhne povolený rozsah (8) nastavíme ho na 1 a zvětšíme počítadlo předchozí. Generování skončí přetečením rozsahu prvního počítadla.

Celý postup pak můžeme zapsat:

```
nastavení počátečního rozložení dam [1, 1, 1, 1, 1, 1, 1,
1]
while nepřeteklo první počítadlo
  if ohodnocující funkce aktuálního řešení = 8
        vyhlas řešení
  end if
  generuj další rozložení dam
end while
```

Uvedeným postupem se pohybujeme ve stromu možných řešení po koncových prvcích zleva doprava, až projdeme všechna rozložení. Přitom najdeme celkem 92 různých řešení této úlohy. Doba řešení však bude velmi dlouhá.

## 8.3 ALGORITMY HEURISTICKÉ

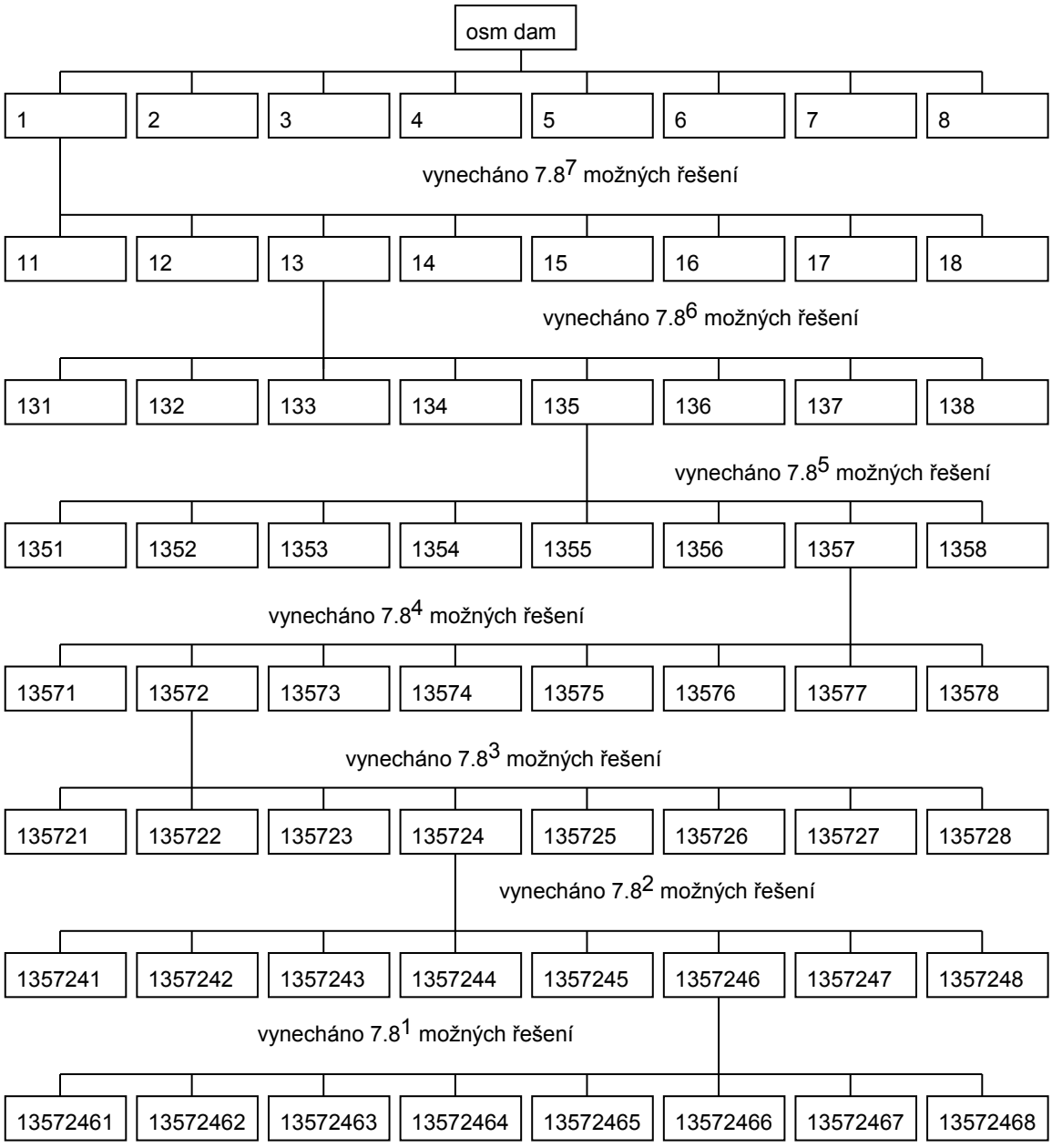
Heuristické algoritmy vycházejí ze zjištění, že nemá smysl postupovat při hledání řešení po stromu rozložení do prvku, který snižuje hodnotu ohodnocující funkce (heuristiky). To je samozřejmě možné jen tehdy, pokud se při cestě k prvku s řešením hodnota ohodnocující funkce nesnižuje. V našem příkladu víme, že musí při přechodu na další prvek vždy vzrůst. Abychom se pokud možno vyvarovali velkých požadavků na paměť, upravíme algoritmus prohledávání do hloubky na ***heuristické prohledávání***.

### 8.3.1 Heuristické prohledávání

Algoritmus heuristického prohledávání můžeme popsat následovně:

```
počáteční nastavení
while není nalezeno řešení
  najdi první dosud nerozvinutý prvek s maximální hodnotou
ohodnocující funkce
  rozviň nalezený prvek a označ ho jako rozvinutý
  spočti ohodnocující funkci všech nových prvků
end while
vyhlas nalezené řešení
```

Postup heuristického prohledávání ilustruje následující obrázek.




**Obrázek 8.5 Postup heuristického prohledávání stromu do hloubky**

Z ukázky vidíme **výhody** algoritmu:

vynecháváme celé oblasti stromu rozložení, ve kterých nemůže řešení existovat,

potřebujeme dosti málo paměti k provedení algoritmu.

### 8.3.2 Vyzkoušení všech slibných kombinací parametrů

Tento algoritmus je na první pohled podobný předchozímu. Také se budeme pohybovat po stromu všech rozložení. Také budeme využívat skutečnosti, že existuje mnoho „slepých větví“, na kterých řešení ležet nemůže. Proto se jim budeme vyhýbat. K jejich nalezení využijeme ohodnocující funkci tak, jak byla definována dříve. Aby další řešení nebylo slepou větví, musí pro dílčí řešení na i-té úrovni stromu platit:

$$f(p_1, p_2, ..., p_i) = i. \tag{8.5}$$

Pak má smysl pokračovat ve stromu hlouběji rozvinutím tohoto prvku. Pro vlastní realizaci využijeme vektor počítadel:

pam[1 - 8 ] = pam[1], pam[2], .....pam[8]

a proměnnou *i*, obsahující číslo úrovně stromu rozložení, na které se nacházíme. Řešení tedy nalezneme tak, že platí podmínka (8.5) a současně $i = 8$. Celý postup pak můžeme zapsat:

```
vynulování pole počítadel
i = 1
while nepřeteklo první počítadlo
  pam[i] = pam[i] + 1
  if pam[i] > 8
        pam[i] = 0
        i = i - 1
  else
        if f(pam[1], ... pam[i]) = i
              if i = 8
                    vyhlas řešení
              else
                    i = i + 1
              end if
        end if
  end if
end while
```

Opět obdržíme jeho aplikací všech 92 řešení, oproti neheuristickému algoritmu vyzkoušení všech možných rozložení však výrazně rychleji.

## 8.4 ALGORITMY PRAVDĚPODOBNOSTNÍ

Uvedené algoritmy vycházejí z poznání, že pro náhodně zvolenou kombinaci hodnot parametrů existuje určitá pravděpodobnost, že tato kombinace určuje optimální řešení.

### 8.4.1 Metoda pokusu a omylu

Algoritmus ***náhodného přístupu*** je velmi neefektivní. Spočívá v náhodném generování rozložení, až dojde k vygenerování správného řešení. Jeho nízkou účinnost si snadno uvědomíme, pokud určíme pravděpodobnost náhodného nalezení řešení při jednom pokusu. Víme, že existuje 92 řešení při 16 777 216 možných rozloženích dam. Pravděpodobnost úspěchu jednoho pokusu tedy je:

$$P(1) = \frac{92}{16777216} \doteq 5{,}48.10^{-6} \tag{8.6}$$

Pravděpodobnost neúspěchu při jednom pokusu určíme snadno jako

$$\overline{P}(1) = 1 - P(1). \tag{8.7}$$

Samozřejmě budeme pokusy opakovat, a tak nás zajímá, jaká bude pravděpodobnost, že při ***n*** pokusech najdeme alespoň jedno řešení (nebo 2, 3,...,n). Nejprve určeme, jaká je pravděpodobnost, že řešení nenalezneme žádné. Jedná se o sjednocení ***n*** vzájemně nezávislých jevů (nenalezení při prvním pokusu, při druhém pokusu atd.):

$$\overline{P}(n) = \prod_{i=1}^{n} \left(1 - P(1)\right) = \left[1 - P(1)\right]^n. \tag{8.8}$$

Pravděpodobnost, že alespoň jedno řešení najdeme, je pak:

$$P(n) = 1 - \left[1 - P(1)\right]^n \tag{8.9}$$

Pro nás bude jistě zajímavé, kolik pokusů je třeba vykonat, abychom se zvolenou pravděpodobností našli alespoň jedno řešení. Využijeme vzorce (8.9) a zjistíme:

$$n = \frac{log\left[1 - P(n)\right]}{log\left[1 - P(1)\right]}. \tag{8.10}$$

Např. pro pravděpodobnost nalezení alespoň jednoho řešení 10 % je třeba vykonat 19 214 pokusů.

Algoritmus náhodného přístupu nevypadá nijak povzbudivě. Přesto může být v některých úlohách úspěšně používán. Určitě bychom ho doplnili o paměť nejlepšího dosaženého řešení, takže při ***n*** pokusech dosáhneme poměrně dobrého řešení.

Mimochodem, ze vzorců je zřejmé, že pro dosažení 100 % jistoty nalezení nejlepšího řešení musíme vykonat nekonečně mnoho pokusů.

### 8.4.2 Genetický algoritmus

***Genetický algoritmus*** využívá pravděpodobnostního přístupu a velmi vhodně jej rozvíjí. Vychází z pozorování přírody, která dokáže úspěšně řešit i velmi složité problémy. Z analogie s živými organismy popišme naší úlohu následovně.

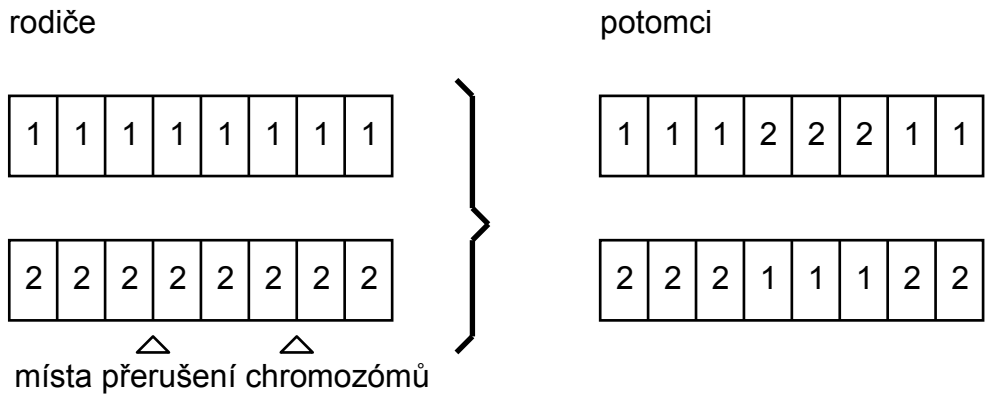
Vlastnosti řešení (rozložení dam) rozdělíme na samostatné prvky, v našem případě to bude umístění jednotlivých dam, a nazveme je ***geny***. Skupina genů, která plně popisuje úlohu, pak bude ***chromozóm***. Ten bude současně vyjadřovat vlastnosti jednoho konkrétního rozložení dam, konkrétního řešení, tedy jednoho ***jedince***. V přírodě se však současně vyskytuje více jedinců, celá ***populace*** jedinců různých vlastností. Tito jedinci spolu vytvářejí ***potomky***, tedy jedince následující ***generace***. Potomci přebírají vlastnosti svých ***rodičů*** do značné míry náhodně, ale pravidlo přirozeného výběru říká, že nejvíce se množí nejlepší jedinci, zatímco nejhorší hynou.

Uvedené principy využijeme v naší úloze. Začneme tím, že náhodně vytvoříme počáteční populaci. Jedince seřadíme podle jejich kvalit. K tomu využijeme již známé ohodnocující funkce. Nejhorší jedince zlikvidujeme a nahradíme potomky nejlepších jedinců. Přitom je vhodné, aby nejlepší jedinci přešli do následující generace beze změny, jinak bychom mohli ztratit nejlepší dosažené řešení.

Jak vznikají potomci. K jejich vytvoření se používají operace, které mají opět analogii v přírodě. Jsou to ***reprodukce*** a ***mutace***.

### 8.4.2.1 *Reprodukce*

Při reprodukci vzniká ze dvou jedinců (***rodičů***) nový jedinec (***potomek***), který přebírá části chromozómů svých rodičů. Kombinace vlastností probíhá tak, že chromozómy rodičů rozdělíme na několik úseků, které přebírají potomci. Místa rozdělení chromozómů jsou volena náhodně. Obvykle je jich 1 ÷ 5 v závislosti na počtu genů.




**Obrázek 8.6 Průběh reprodukcí jedinců**

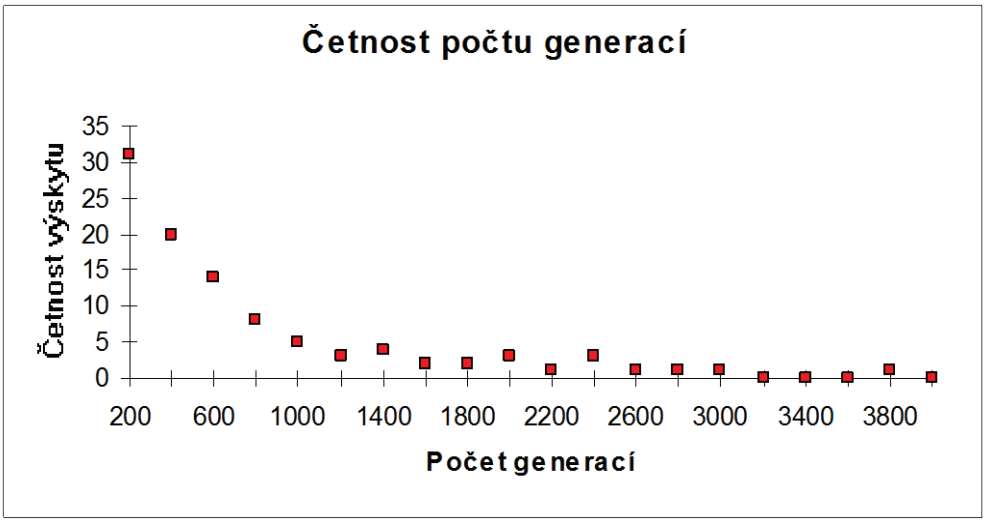
### 8.4.2.2 *Mutace*

Mutace jsou náhodné změny hodnot genů, které dovolí prohledávat okolí nového chromozómu. V algoritmu je nutné stanovit maximální počet změn (obvykle je také náhodný) a maximální změnu hodnoty genu.

Po naplnění nové populace je opět provedeno seřazení jedinců a vytvoření další generace. Algoritmus končí nalezením řešení. Protože konvergence algoritmu může být pomalá, bývá často omezen maximální počet populací. Po jejich provedení sice není zaručeno, že najdeme optimální řešení, ale při vhodném nastavení parametrů algoritmu obvykle najdeme velmi dobré řešení. Genetický algoritmus pracuje s náhodným přístupem, ale díky jeho paměti a variaci s nejlepšími dosaženými výsledky dosahuje výrazně lepší výsledky než metoda pokusu a omylu.

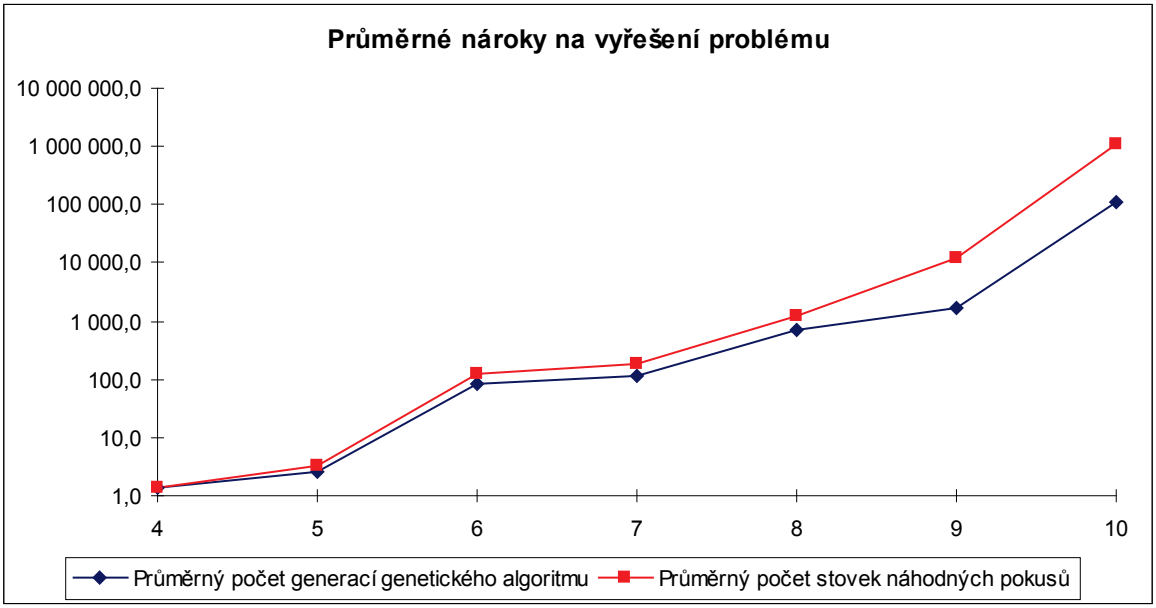
Celý postup je velice jednoduchý. Z pohledu algoritmu není nutná znalost zákonitosti systému (vztahy při umísťování jednotlivých dam), jen popis vlastností systému (ohodnocení kvality jedinců), tedy již několikrát použitá ***ohodnocující funkce*** (***heuristika***). Tato malá znalost je nahrazena velkým výpočetním výkonem. Bohužel nikde není zaručeno, že najdeme skutečně optimální řešení, pokud nevznikne nekonečně velký počet generací.

Testováním algoritmu na problému 8 dam (chromozóm má 8 genů s hodnotami 1, 2, ..., 8) s velikostí populace 100 jedinců, 2 místa přerušení při reprodukci, maximálně 3 mutace v rozsahu změny hodnoty genu maximálně -3 ÷ 3 byl ze 100 průběhů hledání řešení určen střední počet potřebných generací ve výši 685 a jeho rozptyl ve výši 742. Na průběhu četnosti vidíme, že počet generací je náhodná veličina s exponenciálním rozdělením.




**Obrázek 8.7 Průběh četnosti počtu generací nutných k nalezení řešení**

Z hlediska hodnocení složitosti genetického algoritmu je zajímavé srovnání průměrného počtu generací nutných k nalezení řešení. Na obr.8.8 vidíme v logaritmických souřadnicích srovnání průměrného počtu generací s průměrným počtem náhodných pokusů (po 100 pokusech). Z průběhu lze usuzovat, že genetický algoritmus má exponenciální složitost, nároky na dosažení řešení však rostou výrazně pomaleji než u náhodných pokusů.




**Obrázek 8.8 Průměrné nároky na vyřešení problému různými metodami v závislosti na jeho složitosti**

# POUŽITÁ LITERATURA


[1] Arlow, J. & Neustadt, I. *UML a unifikovaný proces vývoje aplikací*. 1. Vyd. Brno, Computer Press, 2003, 388 s. ISBN 80-7226-947-X.

[2] Barton, D. P. & Pears, A. N. Application of Evolutionary Computation. In *Proceedings of First International Conference on Genetic Algorithms "Mendel '95"*. Red. Ošměra, P. Brno, VUT 1995, s. 15 - 21.

[3] Bayer, R. & McCreight, E. M. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1, 1972.

[4] Březina, T. *Informatika pro strojní inženýry I*. 1. vyd. Praha ČVUT 1991, 187 s.

[5] Brodský, J. & Skočovský, L. *Operační systém UNIX a jazyk C*. 1. vyd. Praha, SNTL 1989, 368 s.

[6] Cockburn, A. *Use Case – Jak efektivně modelovat aplikace*. 1. vyd. Brno, CP Books a.s., 2005, 262 s. ISBN 80-251-0721-3.

[7] Častová, N. & Šarmanová, J. *Počítače a algoritmizace*. 3. vyd. Ostrava, skriptum VŠB 1983, 190 s.

[8] Donghui Zhang. *B Trees*. Northeastern University, 22 pp. Dostupný z webu: http://zgking.com:8080/home/donghui/publications/books/dshandbook_BTree.pdf

[9] Drózd, J. & Kryl, R. *Začínáme s programováním*. Praha, Grada 1992, 312 s.

[10] Drozdová, V. & Záda, V. *Umělá inteligence a expertní systémy*. 1. vyd. Liberec, skriptum VŠST 1991, 212 s.

[11] Farana, R. *Zaokrouhlovací chyby a my*. Bajt 1994, č. 9, s 243 – 244.

[12] Flaming, B. *Practical data structures in C++*. New York, USA, Wiley, 1993.

[13] Hodinár, K. *Štandardné aplikačné programy osobných počítačov*. 1. vyd. Bratislava, Alfa 1989, 272 s.

[14] Holeček, J. & Kuba, M. *Počítače z hlediska uživatele*. Praha, SPN 1988, 240 s.

[15] Honzík, J. M., Hruška, T. & Máčel, M. *Vybrané kapitoly z programovacích technik*. 3. vyd. Brno, skriptum VUT 1991, 218 s.

[16] Hudec, B. *Programovací techniky*. Praha, ČVUT 1990.

[17] Jackson, M. A. *Principles of Program Design*. New York (USA), Academic Press 1975.

[18] Jandoš, J. *Programování v jazyku GW BASIC*. Praha, NOTO - Kancelářské stroje 1988, 164 s.

[19] Kačmář, D. *Programování v jazyce C++. Objektová a neobjektová rozšíření jazyka*. Ostrava, ES VŠB-TU 1995, 92 s.

[20] Kačmář, D. & Farana, R. *Vybrané algoritmy zpracování informací*. 1. vyd. Ostrava: VŠB-TU Ostrava, 1996. 136 s. ISBN 80-7078-398-2.

[21] Kaluža, J., Kalužová, L., Maňasová, Š. *Základy informatiky v ekonomice*. 1. vyd. Ostrava, skriptum VŠB 1992, 193 s.

[22] Kanisová, H. & Müller, M. *UML srozumitelně*. 1. vyd. Brno, Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

[23] Kapoun, K. & Šmajstrla, V. *Základní fyzikální problémy - programy v jazyce BASIC a FORTRAN*. 1. vyd. Ostrava, skriptum VŠB 1987, 312 s.

[24] Kelemen, J. aj. *Základy umelej inteligencie*. 1. vyd. Bratislava, Alfa 1992, 400 s.

[25] Knuth, D. E. *The Art of Computer Programming*. Volumes 1-4A, 3rd ed. Reading, Massachusetts, Addison-Wesley, 2011, 3168 pp. ISBN 0-321-75104-3.

[26] Kopeček, I. & Kučera, J. *Programátorské poklesky*. Praha, Mladá fronta 1989, s. 150-155.

[27] Krček, B. & Kreml, P. *Praktická cvičení z programování. FORTRAN*. 1. vyd. Ostrava, skripta VŠB 1986, 199 s.

[28] Kučera, L. *Kombinatorické algoritmy*. 2. vyd. Praha, SNTL 1989, 288 s.

[29] Kukal, J. *Myšlením k algoritmům*. 1. vyd. Praha, Grada 1992, 136 s.

[30] Marko, Š. - Štěpánek, M. *Operačné systémy mikropočítačov SMEP*. 2. vyd. Bratislava/Praha, Alfa/SNTL 1988, 264 s.

[31] Medek, V. & Zámožík, J. *Osobný počítač a geometria*. 1. vyd. Bratislava, Alfa 1991, 256 s.

[32] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 1. vyd. Heidelberg, Springer-Verlag Berlin 1992, 250 s.

[33] *Microsoft Developer Network*. Development Library January 1995. Microsoft Corporation 1995, CD - ROM.

[34] Molnár, Ľ. & Návrat, P. *Programovanie v jazyku LISP*. 1. vyd. Bratislava, Alfa 1988, 264 s.

[35] Molnár, Ľ. *Programovanie v jazyku Pascal*. Bratislava/Praha, Alfa/SNTL 1987, 160 s.

[36] Molnár, Z. *Moderní metody řízení informačních systémů*. Praha, Grada 1992, s. 211 - 221.

[37] Moos, P. *Informační technologie*, 1. vyd. Praha, ČVUT 1993, 200 s.

[38] Nešvera, Š., Richta, K. & Zemánek, P. *Úvod do operačního systému UNIX*. 1. vyd. Praha, ČVUT 1991, 185 s.

[39] Olehla, J. & Olehla, M. aj. *BASIC u mikropočítačů*. 1. vyd. Praha, NADAS 1988, 386 s.

[40] Ošmera, P. Použití genetických algoritmů v neuronových modelech. In *Sborník konference "EPVE 93"*. Brno VUT 1993, s. 88 - 95.

[41] Paleta, P. *Co programátory ve škole neučí aneb Softwarové inženýrství v reálné praxi*. 1. vyd. Brno, Computer Press, 2003, 337 s. ISBN 80-251-0073-1.

[42] Petroš, L. *Turbo Pascal 5.5* – Uživatelská příručka. 1. vydání. Zlín, MTZ 1990, s. 20 – 21.

[43] Plávka, J. *Algoritmy a zložitosť*. Košice, TU Košice, 1998. ISBN 80-7166-026-4.

[44] Podlubný, I. *Počítať na počítači nie je jednoduché*. PC World, 1994, č. 2, s. 112 – 115.

[45] Rawlins, G. J. E. *Compared to what – an introduction to the analysis of algorithms*. Computer Science Press, New York, 1992.

[46] Reverchon, A. & Ducamp, M. *Mathematical Software Tools in C++*. West Sussex (England) John Wiley & Sons Ltd. 1993, 507 s.

[47] Rychlík, J. *Programovací techniky*. České Budějovice, KOPP 1992, 188 s.

[48] Sedgewick, R. *Algorithms*. 1st ed. Addison-Wesley. ISBN 0-201-06672-6.

[49] Sirotová, V. *Programovacie jazyky*. 1. vyd. Bratislava, skriptum SVTŠ 1985, 138 s.

[50] Soukup, B. SGP verze 2.30. *Referenční a uživatelská příručka systému*. Uherské hradiště, SGP Systems 1991, s. 25-55.

[51] Synovcová, M. *Martina si hraje s počítačem*. 1. vyd. Praha, Albatros 1989, 144 s.

[52] Šarmanová, J. *Teorie zpracování dat*. Ostrava, FEI VŠB-TU Ostrava, 2003, 160 s.

[53] Šmiřák, R. *Unified Modeling Language*. Softwarové noviny, 2004, č. 12, s. 76 – 77.

[54] Tichý, V. *Algoritmy I.* Praha, FIS VŠE v Praze, 2006, 190 s. ISBN 80-245-1113-4.

[55] Vejmola, S. *Hry s počítačem*. 1. vyd. Praha, SPN 1988, 256 s.

[56] Virius, M. *Základy algoritmizace*. Praha, ČVUT 2008, 265 s. ISBN 978-80-01-04003-4.

[57] Víteček, A. aj. *Využití osobních počítačů ve výuce*. 1. vyd. Ostrava, ČSVTS FS VŠB Ostrava 1986, 202 s.

[58] Vítečková, M., Smutný, L., Farana, R. & Němec, M. *Příkazy jazyka BASIC*. Ostrava, katedra ASŘ VŠB Ostrava 1989, 44 s.

[59] Vlček, J. *Inženýrská informatika*. 1. vyd. Praha, ČVUT 1994, 281 s.

[60] Wirth, N. *Algoritmy a štruktúry udajov*. 2. vydání. Bratislava, Alfa 1989, s. 19 – 89.

[61] *Základy algoritmizace a programové vybavení*. 2. vyd. Praha, Tesla Eltos 1986, 168 s.

[62] Zelinka, I., Oplatková, Z., Šeda, M., Ošmera, P. & Včelař, F. *Evoluční výpočetní techniky. Principy a aplikace*. 1. vyd. Praha, BEN – Technická literatura, 2009. ISBN 978-80-7300-218-3.