GENETICKÉ ALGORITMY A PROBLÉM N DAM

(GENETIC ALGORITHMS AND N-QUEENS PROBLEM)

Josef HYNEK
Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu Univerzity Hradec Králové
Nejedlého 573, 500 03 Hradec Králové, tel. +420 49 5061 233, E-mail: Josef.Hynek@uhk.cz

SUMMARY

The n-queens problem is a classical combinatorial problem that has been tackled by various problem-solving strategies. This paper reviews prior attempts to solve the problem with the particular stress on utilisation of genetic algorithms. We present a new approach based on a hybrid algorithm that exploits the power of specially designed pre-processing operator and a mutation operator based on a local search. The initial pre-processing allows us to obtain very quickly a partial solution with the relatively low number of conflicts and then the special mutation operator does the soft part of the work as it smoothes away the remaining conflicts. Using this kind of a local space search we are able to speed up the evolution process. This efficient algorithm is capable of finding a solution for large size n-queens problems on a common PC. We believe that our approach may be useful for understanding other constrain-based search problems.

Keywords: *N-queens problem, genetic algorithm, genetic operators, heuristic algorithm*

1. ÚVOD

Problém N dam je klasický kombinatorický problém, který je často studován různými metodami z oblasti umělé inteligence. Je to typický problém typu splňování podmínek (constraint satisfaction problem), který je jednoduše definovatelný a zřejmě i proto je opakovaně používán jako vhodná testovací úloha pro různé strategie řešení problémů.

Problém *N* dam je inspirován hrou šachy, kde dáma je figurkou, která má po šachovnici možnost pohybu v rámci řádku, sloupce a diagonálně o libovolný počet políček přímým směrem. Najít řešení problému *N* dam vyžaduje rozmístit *N* dam na šachovnici o rozměrech *N*x*N* tak, aby se dámy vzájemně neohrožovaly, to jest aby žádná z dam neatakovala kteroukoliv jinou dámu.

I přes velice jednoduché zadání a přímou vazbu na hru šachy má problém *N* dam celou řadu praktických aplikací v různých oborech lidské činnosti. Jako příklad lze uvést aplikace v oblastech testování VLSI, komunikačních systémů, rozvrhování úloh a zdrojů, datové komprese a mnohých dalších (viz. [6], [12]).

2. TRADIČNÍ ALGORITMY

Velké množství prací bylo v této oblasti věnováno nejrůznějším strategiím řešení problému *N* dam na bázi backtrackingu (viz. například přehled uvedený v [12]). V tomto případě umístíme *i*-tou dámu (počínaje $i = 1$) na přípustné políčko šachovnice tak, aby žádná z dosud umístěných dam (předpokládáme, že *i*-1 dam je již rozmístěno na šachovnici) neohrožovala nově přidanou *i*-tou dámu. Tímto způsobem pokračujeme do té doby, až se nám podaří umístit všech *N* dam. Pokud nastane kdykoliv v průběhu tohoto algoritmu situace, že další dámu není možné na šachovnici umístit, vrátíme se o krok zpět a změníme pozici dámy, kterou jsme umístili v tomto kroku. Backtracking sice umožňuje najít všechna řešení problému pro zadané *N* (řešení existuje pokud $N \geq 4$), ale vzhledem ke kombinatorické explozi tímto způsobem můžeme řešit pouze problémy s velmi malým počtem dam *N*. V ostatních případech se potom musíme spolehnout na nejrůznější algoritmy heuristické, jejichž základní přehled je uveden například v [3].

Sosič a Gu [12, 14] navrhli několik stochastických algoritmů, které jsou založeny na myšlence lokálního hledání možností odstranění konfliktů mezi jednotlivými dámami. Tyto algoritmy generují náhodné rozmístění *N* dam na šachovnici a protože takovéto rozmístění obvykle zahrnuje velké množství konfliktů, jsou použity různé heuristiky na bázi vzájemné výměny pozic dam tak, aby počet konfliktů byl redukován. Procedura vyměňující dvojice dam (respektive jejich umístění) se opakuje tak dlouho, dokud není nalezeno řešení problému. Pokud není možné provést žádnou další výměnu, která by redukovala aktuální nenulový počet konfliktů, vygeneruje se náhodně nové počáteční rozmístění a celý proces se opakuje.

Bylo navrženo a testováno několik variant výše popsaného algoritmu, které se vzájemně liší heuristikou výběru dvojic dam pro vzájemnou výměnu pozic [13, 14]. Bez ohledu na tyto jednotlivosti příslušný algoritmus pracuje v polynomiálním čase a nepoužívá žádnou formu backtrackingu. Testováním bylo ověřeno, že tyto stochastické algoritmy jsou schopny řešit i problémy *N* dam, kde *N* je v řádu miliónů [13].

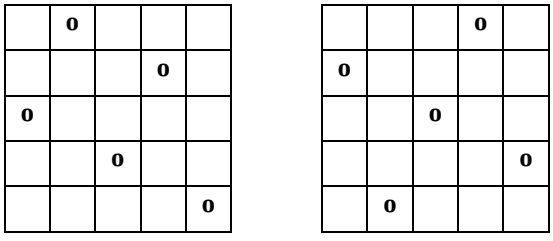
3. GENETICKÉ ALGORITMY

Než popíšeme některé dokonalejší způsoby aplikace genetického algoritmu pro řešení problému

N dam, rádi bychom využili této příležitosti a alespoň krátce na tomto problému ukázali, že snadná a zdánlivě přímočará řešení nemusí ani zdaleka být optimální, ale jsou spíše zdrojem velkých problémů. A tyto problémy můžeme snadno způsobit již nevhodnou volbou reprezentace individuí.

Je zřejmé, že každá z N dam může být umístěna na libovolné políčko šachovnice. Pokud si políčka například očíslujeme postupně po řádcích čísly 1 až N^2, potom můžeme potenciální řešení problému zakódovat jako chromosomy o N genech, přičemž každý z těchto genů reprezentuje pozici jedné konkrétní dámy na šachovnici a může tedy nabývat N^2 různých hodnot. Příklad této poněkud naivní reprezentace individuí můžeme vidět na obrázku číslo 1.

Nyní bychom mohli začít přemýšlet o návrhu vhodných genetických operátorů, ale bude mnohem rozumnější ještě předtím kriticky posoudit, zda námi navržená reprezentace individuí je smysluplná a efektivní. Začneme-li přemýšlet o redundanci, která je s tímto konkrétním kódováním spjata, mělo by nám okamžitě přijít dosti podezřelé, že náš způsob reprezentace umožňuje zaobírat se i takovými individui, která jsou na první pohled nepřípustná. Předpokládáme-li pro účely tohoto výkladu $N = 5$, potom jako příklad lze uvést třeba chromosom (3,3,3,3,3), který reprezentuje řešení, kdy všech pět dam bude umístěno na jediném políčku uprostřed prvního řádku šachovnice.



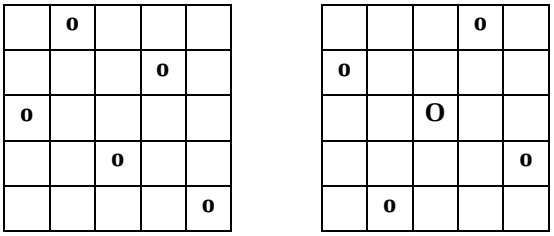
[2,9,11,18,25] [4,6,13,20,22]

Obr. 1 Příklad nevhodné reprezentace problému N dam (pro N=5)
Fig. 1 An example of unsuitable representation of the N-queens problem (N=5)

K odstranění tohoto problému nám postačí předpokládat, že chromosom nesmí obsahovat dvě stejné alely, čímž bude zajištěno, že při dekódování libovolného chromosomu nenastane situace, kdy by bylo požadováno umístit více než jednu dámu na totéž políčko šachovnice.

Při dalším zkoumání zvoleného způsobu kódování nás jistě napadne, že výše uvedeným způsobem jsme vyřešili pouze nejkřiklavější problém, ale mnoho dalších na odpovídající řešení ještě stále čeká. Budeme-li pro jednoduchost uvažovat chromosom (1,2,3,4,5), jasně vidíme, že dámy sice obsadí různá políčka šachovnice, ale ani tímto chromosomem a jemu odpovídajícím potenciálním řešením by se náš algoritmus neměl příliš zdržovat. Všechny dámy jsou totiž umístěny na jednom jediném řádku a totéž se nám bude analogicky opakovat pro všechny sloupce a úhlopříčky. Pokud bychom byli požádání úlohu řešit manuálně bez použití počítače, zcela jistě bychom takovouto kombinaci umístění dam ani nezkoušeli, protože znamená jen nevyhnutelnou ztrátu času. Nechceme-li však zbytečně trávit čas čekáním na dokončení běhu námi navrženého algoritmu, musíme změnit způsob reprezentace individuí a to tak, že využijeme stejných znalostí, které nám pomohly výše popsanou redundaci použitého kódování odhalit.

K reprezentaci individuí použijeme permutační kódování, kde jednotlivá potenciální řešení jsou zakódována jako permutace množiny čísel $\{1,..,N\}$, kde N je velikost problému. Číslo i na j-té pozici v této permutaci potom reprezentuje dámu, která byla umístěna na i-tý řádek a do j-tého sloupce (viz. obrázek č. 2).



[3,1,4,2,5] [2,5,3,1,4]

Obr. 2 Permutační reprezentace problému N dam (pro N=5)
Fig. 2 Permutation based representation of the N-queens problem (N=5)

Protože pozice čísla v permutaci určuje sloupec, ve kterém je příslušná dáma umístěna, jsou automaticky eliminovány jakékoliv konflikty v rámci jednoho sloupce. Navíc, protože jde o permutaci a každé číslo z množiny $\{1,..,N\}$ se v ní opakuje právě jedenkrát, jsou analogicky eliminovány i veškeré potenciální konflikty v jednotlivých řádcích šachovnice. Volbou tohoto způsobu reprezentace nám pro nalezení řešení zbývá odstranit pouze konflikty na úhlopříčkách.

Zdaleka ještě není vyhráno, ale je nutné si uvědomit, že permutační reprezentací jsme oproti výše diskutovanému způsobu kódování podstatně zmenšili velikost prohledávaného prostoru. I přesto však namísto N^{2N} ještě zbývá $N!$ různých možností, které není možné otestovat vyčerpávajícím způsobem již při relativně malém N. Tabulka číslo 1 shrnuje pro názornější představu velikost prohledávaného prostoru pro různá N a odtud je zcela zřejmé, že exhaustivní algoritmus je v tomto případě nepoužitelný.

Permutační kódování potenciálních řešení využívá genetický algoritmus, který popisuje

Crawford v [1]. Ohodnocení individua je zde definováno skrze počet diagonálních konfliktů, přičemž pro libovolné dvě nebo více dam na stejné úhlopříčce je započítáván pouze jediný konflikt. Z toho vyplývá, že toto ohodnocení je nezáporné a individuum s nulovým počtem konfliktů je řešením problému N dam. V této práci autor nejprve experimentoval s operátorem křížení typu PMX, ale protože výsledky nebyly příliš povzbuzující, tento operátor byl nahrazen jednoduchým operátorem mutace, který provádí výměnu pozic dvou náhodně zvolených dam. K urychlení kalkulace fitness funkce, která odráží počet konfliktů obsažených v daném řešení, zde namísto tradičního způsobu, který postupně načítá konflikty pro jednotlivé dámy, byla navržena jednoduchá heuristika využívající informaci o počtu obsazených diagonál [1]. Závěrem této práce jsou uvedeny výsledky experimentů, ze kterých je zřejmé, že i na velmi výkonném stroji IBM RS6000 trvalo nalezení řešení pro $N = 500$ dam použitím tohoto algoritmu cca 2 hodiny!

N	Naivní kódování		Permutační kódování	
	N^{2N}	10^k	$N!$	10^k
100	100^{200}	10^{400}	100!	10^{158}
500	500^{1000}	10^{2699}	500!	10^{1134}
1000	1000^{2000}	10^{6000}	1000!	10^{2568}
2000	2000^{4000}	10^{13204}	2000!	10^{5735}
5000	5000^{10000}	10^{36990}	5000!	10^{16325}
10000	10000^{20000}	10^{80000}	10000!	10^{35659}

Tab. 1 Velikost prohledávaného prostoru
Tab. 1 Search space size

4. HYBRIDNÍ GENETICKÉ ALGORITMY

Protože výše uvedené výsledky nejsou nijak oslnivé, je pochopitelné, že se mnoho dalších autorů pokoušelo zdokonalit ve své podstatě běžný genetický algoritmus hybridizací s některou z tradičních technik.

Zajímavý hybridní genetický algoritmus k řešení problému N dam byl popsán v [2]. Autoři zde vycházejí rovněž z permutačního kódování, ale od předchozí práce se liší v tom, že navrhují využít speciální asexuální heuristický operátor založený na myšlence lokálního prohledávání. Bylo navrženo nejprve identifikovat dámu s nejvyšším počtem konfliktů, poté se vyčerpávajícím způsobem projdou veškeré možné výměny s ostatními dámami a následně se provede ta výměna, která odstraní co možná nejvíce konfliktů. Eiben a kol. [2] tvrdí, že jejich metoda produkuje lepší výsledky než které uvádí Crawford [1] zejména s ohledem na časovou náročnost algoritmu, ale jakékoliv porovnání na konkrétní úloze zde bohužel chybí.

Při studiu obou výše uvedených prací nás zaujalo, že jejich autoři se velice rychle zřekli možnosti použití operátoru křížení a obě výše uvedená řešení využívají pouze operátoru mutace. Přitom právě operátor křížení bývá obvykle považován za hlavní nástroj přinášející změnu a inovaci v rámci genetického algoritmu, zatímco mutaci bývá přisuzována druhotná role zabraňující předčasnému zastavení vývoje v populaci (viz. např. [11, s. 173]). Tato skutečnost nás motivovala k uskutečnění poměrně rozsáhlé série experimentů, jejichž cílem bylo posoudit efektivnost různých operátorů křížení pro tuto konkrétní úlohu.

Naše experimenty, které jsou podrobně popsány v [8], ukázaly, že operátory křížení umožňují poměrně rychle vygenerovat částečná řešení s relativně nízkým počtem konfliktů, ale v naprosté většině případů se evoluční proces výrazně zpomalil ve chvíli, kdy počet konfliktů klesl zhruba na $N/3$, kde N je počet dam. Samozřejmě, že následná aplikace operátoru mutace umožnila postupně odstranit zbývající konflikty a řešení problému bylo nalezeno, ale přesto jsme přece jen očekávali ještě výraznější roli křížení a úsporu času při vlastním výpočtu oproti výše zmiňovaným přístupům.

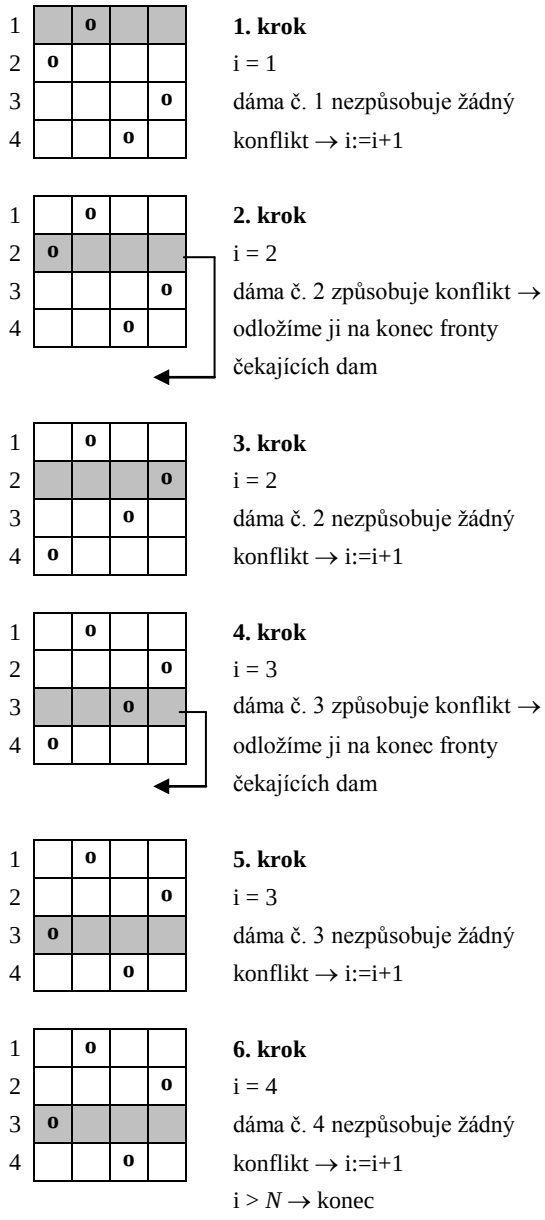
Pro urychlení výpočtu jsme po zhodnocení předchozích experimentů navrhli modifikovaný algoritmus, který vhodně kombinuje výhody operátoru křížení na straně jedné a operátoru mutace založené na lokálním prohledávání na straně druhé. Abychom zkrátili dobu výpočtu, modifikovaný algoritmus používá v prvních k generacích pouze operátor křížení s částečným přiřazením a teprve po nalezení řešení s relativně nízkým počtem konfliktů genetický algoritmus začne využívat i operátoru mutace, který lokálně vyhledává možné změny pozic dam s cílem dosáhnout odstranění všech zbylých konfliktů. V téže práci [8] jsme ukázali, že tímto způsobem lze významným způsobem celý algoritmus urychlit.

5. HEURISTICKÝ ALGORITMUS

Při výše popsaných experimentech jsme testovali nejen tradiční operátory křížení, které byly speciálně navrženy pro práci s permutační reprezentací (viz. např. [5]), ale také celou řadu námi navržených operátorů, jejichž společným cílem bylo co nejrychleji vygenerovat částečná řešení problému s co možná nejmenším počtem konfliktů. Mezi celou řadou dalších nápadů nás velice zaujal velmi jednoduchý postup, který jsme nazvali operátorem předběžného zpracování [7].

Tato jednoduchá procedura je založena na myšlence postupného odstranění dam (respektive odpovídajících řádků), které jsou příčinou některého z konfliktů, z částečného řešení a jejich ponechání ve frontě pro pozdější zapracování. Postupně procházíme permutaci, která reprezentuje potenciální řešení, a zjišťujeme, zda i-tá dáma není

v konfliktu s dámami umístěnými na řádcích 1 až *i*-1. Jestliže žádný konflikt nezaznamenáme, zvýšíme hodnotu počítadla i a provedeme kontrolu pro další dámu v pořadí. Pokud *i*-tá dáma je v konfliktu s některou dam umístěných na předchozích řádcích, tuto dámu prozatím vyloučíme z uvažování a přemístíme ji na *N*-tou pozici v permutaci tak, že dámy na řádcích *i*+1 až *N* se posunou vždy o řádek nahoru. Hodnota počítadla i se nemění a bude tedy znovu posuzována vhodnost umístění *i*-té dámy (původně dáma na řádku *i*+1, která se posunem dostala na *i*-tý řádek). Celý postup je názorně předveden na obrázku číslo 3.



Obr. 3 Použití operátoru předběžného zpracování
Fig. 3 Application of the preprocessing operator

Protože operátor předběžného zpracování je založen na velice jednoduché myšlence, byli jsme velice překvapeni pozoruhodnými výsledky, kterých je možné při jeho použití dosáhnout. Již od prvních pokusů bylo zřejmé, že tento operátor zcela zřetelně předčí operátory křížení, s nimiž jsme experimentovali dříve. Jestliže operátory křížení nám umožňovaly nalézt částečné řešení s počtem konfliktů úměrným zhruba třetině počtu dam (*N/3* - viz. předchozí oddíl), výše popsaný operátor předběžného zpracování produkuje částečná řešení s malým počtem konfliktů bez ohledu na velikost parametru *N*. Abychom toto tvrzení podpořili experimentálně, zkoumali jsme chování operátoru předběžného zpracování na 100 náhodně vygenerovaných permutacích příslušné délky a v tabulkách číslo 2 a 3 jsou zaznamenány průměrné, minimální a maximální počty konfliktů v částečném řešení před a po aplikaci tohoto operátoru.

Počet dam (*N*)	100	250	500	1000
Minimum	45	114	233	506
Maximum	61	150	285	554
Průměr	52,9	132,7	263,8	525,1

Tab. 2 Počet konfliktů v náhodně vygenerovaném částečném řešení *(pro každou hodnotu parametru N bylo provedeno 100 nezávislých experimentů)*
Tab. 2 Number of conflicts in random permutations *(Average of 100 independent experiments)*

Počet dam (*N*)	100	250	500	1000
Minimum	1	1	3	4
Maximum	49	57	26	30
Průměr	10,3	13,7	12,9	13,9

Tab. 3 Počet konfliktů v částečném řešení po předběžném zpracování *(pro každou hodnotu parametru N bylo provedeno 100 nezávislých experimentů)*
Tab. 3 Number of conflicts in random permutations after preprocessing *(Average of 100 independent experiments)*

Z těchto tabulek je zcela zřetelné, že průměrný počet konfliktů po předběžném zpracování individua je velmi malý a nezávisí na velikosti problému. Lze tedy konstatovat, že operátor předběžného zpracování umístí bez vyvolání konfliktu téměř všechny dámy i pro velké hodnoty *N*. Odtud vyplývá, že můžeme-li tímto způsobem velmi snadno získat částečné řešení s malým počtem konfliktů, nemá velký smysl generovat celou populaci takovýchto "téměř nekonfliktních" jedinců. Namísto toho bude zcela přirozeně vhodnější použít dříve navržený a úspěšně otestovaný operátor mutace (viz. [8]) a pokusit se odstranit zbývající konflikty. Operátor mutace funguje tím způsobem, že vybere v permutaci pozici *x*, která odpovídá dámě způsobující některý z konfliktů, a vyčerpávajícím způsobem hledá pozici *y* ($x \neq y$) v téže permutaci takovou, že výměnou příslušných dam dojde k redukci celkového počtu konfliktů.

Finální verze našeho heuristického algoritmu (viz. [7]) vygeneruje náhodně částečné řešení (libovolnou permutaci množiny $\{1,..,N\}$), které je potom transformováno operátorem předběžného zpracování, čímž dojde k výraznému snížení počtu konfliktů. Následně je aplikován operátor mutace, který postupnými výměnami pozic dam eliminuje zbylé konflikty. Jestliže není možné další výměnou dosáhnout snížení počtu konfliktů a řešení ještě nebylo nalezeno (t.j. počet konfliktů je větší než nula), je vygenerována nová náhodná permutace a celý proces se znovu opakuje.

Na první pohled je zřejmé, že předem neznámý počet cyklů, kdy musíme vygenerovat nové počáteční řešení a nemáme jistotu, že se podaří odstranit veškeré konflikty a bude nalezeno přípustné řešení, se může zdát velkou slabinou tohoto algoritmu. Z tabulky číslo 4 však vidíme, že počty počátečních permutací pro nalezení 100 řešení problému N dam pro různé hodnoty N, nepředstavují vážnou překážku pro praktické použití tohoto algoritmu a kromě toho s rostoucí hodnotou parametru N se počet vynucených opakovaných startů snižuje.

Počet dam (N)	100	250	500	1000
Počet počátečních permutací	196	163	107	103

Tab. 4 Počet náhodně vygenerovaných permutací pro nalezení 100 řešení
Tab. 4 Number of initial random permutations needed to find 100 solutions

Algoritmus byl implementován v prostředí LPA WIN-PROLOGu, přičemž jsme využili některých obecných procedur z dříve vyvinutého systému GAP, který je detailně popsán v [9]. Veškeré výpočty byly prováděny na běžném PC (Intel Pentium Pro 150). I přesto, že jde o velmi standardní stroj a PROLOG bývá obvykle vnímán jako nepříliš efektivní programovací jazyk z hlediska provedení vlastního výpočtu, podařilo se nám ve velmi rozumném čase získat řešení problému pro 100, 250, 500 i 1000 dam (viz. tabulka č. 5).

Počet dam (N)	100	250	500	1000
Minimum [s]	0,05	0,49	7,53	33,01
Maximum [s]	0,33	2,58	12,79	57,46
Průměr [s]	0,26	2,01	8,78	48,26

Tab. 5 Čas výpočtu *(pro každou hodnotu parametru N bylo provedeno 100 experimentů)*
Tab. 5 Execution time *(Average of 100 independent experiments)*

6. ZÁVĚR

Z výše uvedeného vyplývá, že naše snaha navrhnout efektivní genetický algoritmus pro řešení problému N dam vedla postupně od poněkud naivního genetického algoritmu přes hybridní genetické algoritmy zpět k návrhu heuristického algoritmu, který namísto celé populace individuí pracuje pouze s jediným potenciálním řešením. Přesto se domníváme, že tato cesta nebyla marná, protože výsledný heuristický algoritmus vznikl díky znalostem, které jsme nabyli během testování různých variant algoritmu genetického. Operátor mutace jsme zde použili v nezměněné podobě a úvahy o návrhu dalších rekombinačních operátorů nás přivedly k myšlence na jednoduchý, ale přesto velice účinný, operátor předběžného zpracování. Výsledky naší práce navíc bezprostředně využili Estivill-Castro a Torres-Velázquez [4] při konstrukci genetického algoritmu pro problém N dam s oceněním, kdy jsou jednotlivým políčkům šachovnice o rozměru $N*N$ přiřazeny určité váhy a cílem je najít takové bezkonfliktní rozmístění N dam, kdy součet vah polí, na kterých tyto dámy jsou rozmístěny, je maximální.

Tento příspěvek bychom mohli ukončit konstatováním, že přestože se nám nepodařilo navrhnout efektivní evoluční algoritmus pro řešení problému N dam, naše práce přinesla výsledek ve formě funkčního a poměrně rychlého heuristického algoritmu. Odtud by se mohlo zdát, že je dosti nepravděpodobné, aby genetický algoritmus dosáhl při řešení problému N dam lepšího výkonu než relativně jednoduchý heuristický algoritmus popsaný výše. Přesto se objevují nové a nové pokusy aplikovat genetické algoritmy na tuto úlohu, které mohou přinejmenším znovu obohatit možnosti jejího řešení.

Jeden z posledních článků na toto téma, které jsme při psaní této práce měli možnost získat a prostudovat, je zajímavý naprosto odlišným způsobem myšlenkové konstrukce navrhovaného řešení.

Kiliç a Kaya [10] si povšimli faktu, že vezmeme-li pro dané konkrétní N všechna přípustná řešení a ohodnotíme jednotlivá políčka na šachovnici číslem, které odpovídá počtu řešení, v nichž na daném políčku byla umístěna dáma, zjistíme značné rozdíly mezi takto získaným ohodnocením jednotlivých políček i celých oblastí šachovnice. Na základě této informace můžeme snadno vytvořit jakousi mapu rozložení přípustných řešení na šachovnici, která nám udává pravděpodobnost s jakou se při nalezení přípustného řešení bude některá z dam vyskytovat na vybraném políčku či v některé části šachovnice. Zřetelně se například ukazuje, že mezi oblastmi, kde je poměrně nízká pravděpodobnost bezkonfliktního umístění některé dámy patří oblasti uprostřed a v rozích šachovnice. S využitím této mapy Kiliç a Kaya navrhli rozdělit plochu šachovnice na pět disjunktních oblastí spolu s metodou, jak při generování počáteční populace postupně využívat těchto oblastí k získání potenciálních řešení s minimálním počtem konfliktů.

Přestože podle provedených experimentů ani toto řešení nemůže konkurovat výše prezentovanému algoritmu heuristickému, jedná se o zajímavou myšlenku, která může být využita při zdokonalování některého z existujících algoritmů.

LITERATURA

[1] Crawford, K.D.: Solving the n-Queens Problem Using Genetic Algorithms. In: Proceedings of ACM/SGAPP Symposium on Applied Computing, Kansas City, March 1-3, 1992, pp. 1039-1047.

[2] Eiben, A.E., Raué,P.E., Ruttkay, Z.: How to Apply Genetic Algorithms to Constrained Problems. In: Chambers, L. (Ed.): Practical Handbook of Genetic Algorithms. Vol. 1, 1995, pp. 307-353.

[3] Erbas, C., Sarkeshik, S., Tanik, M.M.: *Different Perspectives of the N-Queens Problem*. In Proceedings of the ACM 1992 Computer Science Conference, ACM Press, 1992, s. 99-108.

[4] Estivill-Castro, V., Torres-Velázquez, R.: *How Should Feasibility be Handled by Genetic Algorithms on Constraint Combinatorial Optimization Problems?* In Proceedings of The Genetic and Evolutionary Computation Conference GECCO 2001, San Francisco, 2001.

[5] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA, 1989.

[6] Gu, J.: Constraint-Based Search. Cambridge University Press, New York 1992.

[7] Hynek, J.: *The N-Queens Problem Revisited*. In Proceedings of the ICSC Congress on Intelligent Systems and Applications (ISA'2000), University of Wollongong, Australia. ICSC Academic Press, 2000, Vol. II, s. 379-384.

[8] Hynek, J.: *An Improved Genetic Algorithm for the n-Queens Problem*. In Proceedings of the International Conference on Artificial Intelligence IC-AI'2000, Las Vegas, NV. CSREA Press 2000, Vol. I., s. 517-522.

[9] Hynek, J.: A Prolog Implementation of Genetic Algorithms. Ph.D. Thesis, Charles University, Prague 1998.

[10] Kiliç, A., Kaya, M.: *A New Local Search Algorithm Based on Genetic Algorithms for the N-Queens Problem.* In Proceedings of The Genetic and Evolutionary Computation Conference GECCO 2001, San Francisco, 2001.

[11] Mitchell, M.: An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1996.

[12] Sosič, R., Gu, J.: Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem. IEEE Transactions on Knowledge and Data Engineering. Vol. 6, 5, pp. 661-668, Oct 1994.

[13] Sosič, R., Gu, J.: 3,000,000 Queens in Less Than One Minute. SIGART Bulletin, Vol. 2, pp. 22-24, 1991.

[14] Sosič, R., Gu, J.: A Polynomial Time Algorithm for the N-Queens Problem. SIGART Bulletin, Vol. 1, 3, pp. 7-11, Oct 1990.

BIOGRAPHY

Josef Hynek got his Ph.D. in theoretical informatics from The School of Mathematics and Physics, Charles University in Prague in 1998. He has been working for The Faculty of Management and Information Technology, University of Hradec Králové since 1991. Currently he has a position of a vice-dean and is responsible for science, development and international relations. His scientific focus is on evolutionary algorithms and their applications.