МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2009
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2009
Proceedings of the Thirty Eighth Spring Conference of
the Union of Bulgarian Mathematicians
Borovetz, April 1–5, 2009

АЛГОРИТМИ ЗА КЛАСИФИКАЦИЯ НА КОМБИНАТОРНИ ОБЕКТИ С ОТХВЪРЛЯНЕ НА ИЗОМОРФНИТЕ В ПРОЦЕСА НА ГЕНЕРИРАНЕ*

Илия Буюклиев

Има три основни типа алгоритми за класификация на комбинаторни обекти с отхвърляне на изоморфните. Първият се базира на поддържане на списък с неизоморфни представители на класовете от вече обходени обекти. Вторият, известен като "генериране с каноничен представител", използва представител, който е максимален елемент от класа на еквивалентност по отношение на (лексикографска) наредба в множеството от разглеждани обекти. Този подход е въведен независимо от Read и Фараджев. Третият е подходът на McKay, известен като "канонично разширяване". В тази статия са представени основните концепции на тези три подхода.

1. Въведение. Много актуални дискретни задачи са свързани с класификация на комбинаторни обекти. Много малко са случаите, в които могат да бъдат построени без компютър всички различими комбинаторни структури с желаните свойства, даже и броят им да не е голям. Понякога е възможно да се намерят горни и долни граници за този брой или той да се определи точно, но дори и тогава генерирането е възможно само с помощта на подходящ алгоритъм. Задачата за класификация на комбинаторни обекти формално може да се разгледа като съставена от две подзадачи: едната е генериране или конструиране на обектите, които разглеждаме, а другата е тестът за изоморфизъм. От гледна точка на сложността на алгоритми, изчерпващото генериране на много от комбинаторните обекти се числи към изчислително трудните задачи. А и задачата за проверка на изоморфност, в общия случай, засега няма полиномиално решение. Затова решаването на определени класификационни задачи зависи от разработването на ефективни алгоритми, при които се реализира генерирането на обекти едновременно с тестване за изоморфизъм, с цел максимално намаляване на необходимия компютърен ресурс. Този вид алгоритми са известни като алгоритми за класификация с отхвърляне на изоморфните обекти в процеса на генериране. Освен стратегията, при която неизоморфните представители се записват и всеки новогенериран представител се тества за изоморфизъм с вече известните неизоморфни обекти, са възможни и други техники. Те основно се базират на работите на Read [13], Фараджев [5] и MacKay [12].

*2000 Mathematics Subject Classification: 05E20.
Ключови думи: класификация, канонична форма, алгоритми.

Тази статия е посветена на основните принципи на тези методи. В следващата част е дадена по-формална дефиниция на проблема, като са въведени и необходимите понятия. След това са представени различните подходи, като са дадени и съответни примери. Накрая са споменати някои резултати.

2. Изчерпващо търсене и алгоритми за класификация. С класификационните алгоритми се изучават комбинаторни обекти, притежаващи нетривиални свойства, например регулярност (както при комбинаторните дизайни), екстремални структурни свойства (оптимални линейни кодове) и други. Построяването на обект с желаните свойства става с използването на вече построена или известна част от него, която ще наричаме подобект. На практика всеки от алгоритмите за генериране на такива обекти има характерни особености. Общото между тях е, че генерирането става стъпка по стъпка, по всички възможности за генериране на подобекти. Този начин на генериране на практика се реализира чрез търсене върху клас от подобекти. Основните затруднения идват от това, че параметрите на подобектите не са строго определени, като в повечето случаи се интересуваме само от необходими условия дадена структура да е подобект. Затова и търсенето се извършва върху голямо множество от подобекти.

Сега ще дадем по-конкретно описание. Да дефинираме пространството или областта на търсене.

Дефиниция 1. *Под област на търсене ще разбираме крайно множество Ω, което съдържа всички обекти (подобекти), разглеждани в процеса на търсене.*

За по-голяма яснота на изложението в тази работа ще считаме, че Ω се състои от всички подмножества на крайно множество U или $\Omega = 2^U$. Нека в Ω е дефинирана релация на еквивалентност $\sim$, която го разбива на класове на еквивалентност. На практика, ако се интересуваме от подмножество от обекти $\Gamma \subset \Omega$, за които са изпълнени исканите ограничения, задачата за класификация се дефинира като намиране на точно един представител на всеки клас на еквивалентност на Γ. Процесът на търсене се моделира удобно с кореново дърво.

Дефиниция 2. *Дървото на търсене е кореново дърво, чиито възли са обекти от Ω. Два възела са свързани с ребро тогава и само тогава, когато единият може да се получи от другия с една стъпка на търсене.*

Дефиниция 3. *Под **ниво** или **дълбочина** на възел разбираме дължината на пътя от възела до корена.*

Дефиниция 4. *Всички възли, които са на ниво с едно повече от нивото на даден възел X и са свързани с него, се наричат негови преки наследници (деца) и се бележат с $C(X)$. Възелът, който е на ниво с едно по-малко от нивото на даден възел X и е свързан с него, се нарича родителски възел и се бележи с $p(X)$.*

На практика дървото на търсене се дефинира с областта си на търсене Ω, с кореновия си възел $r \in \Omega$ и правилото $X \to C(X)$, по което се получават наследниците на даден възел. Това правило се определя в общия случай от свойство (или условие P), което се онаследява. За удобство се разглежда това дърво, което в много случаи е по-голямо от дървото, обхождано от определен алгоритъм.

Изпълнението на алгоритмите за търсене може да се разглежда като обхождане на дървото на търсене и посещаване на всеки от възлите посредством ребрата.

Модел на такова обхождане е обхождането на дърво в дълбочина. То може да се опише така: Започвайки от корена r, обходи рекурсивно всички наследници на възела, преди да се върнеш в този възел.

Това обхождане може да се реализира с подхода търсене с връщане (backtracking). Да обсъдим отново дървото на търсене. Възлите на първо ниво съответстват на подобекти и могат да се разгледат като решения за това ниво, принадлежащи на крайното множество U. Решението, съответстващо на ниво k, може да се разгледа като k-мерен вектор $(a_1, a_2, \ldots, a_k)$, $a_i \in U, i = 1, 2, \ldots, k$, тъй като пряко зависи от решенията на предните нива. Решенията на някакво предварително фиксирано ниво l отговарят на търсения обект. Тези решения наричаме *пълни*.

Търсенето с връщане представлява рекурсивно разширяване на текущото решение с една стъпка. Нека имаме решението $(a_1, a_2, \ldots, a_k)$ като вход. Процедурата за търсене с връщане определя множество от разширения на текущото решение $A_{k+1} = A_{k+1}(a_1, a_2, \ldots, a_k) \subseteq U$ и за всяко разширение $a_{k+1} \in A_{k+1}$ рекурсивно извиква себе си с вход $(a_1, a_2, \ldots, a_k, a_{k+1})$. Когато всички елементи на A_{k+1} са обходени или е достигнато нивото на пълните решения, се реализира етапът на връщане в извикващата процедура.

```
PROCEDURE BACKTRACK ((a_1, a_2, ..., a_k) : решение )
begin
    if k = l then изведи намереното решение;
    пресметни множеството от решения A_{k+1}(a_1, a_2, ..., a_k);
    for all a_{k+1} ∈ A_{k+1} do
    backtrack(a_1, a_2, ..., a_k, a_{k+1});
end;
```

Основният въпрос, който разглеждаме нататък в статията, е какви са методите за съкращаване на дървото на търсене, като се използва въведената релация на еквивалентност $\sim$ в Ω. Изоморфизмът между комбинаторни обекти в много случаи е удобно да се представи в термините на действие на група върху множество.

Нека разгледаме крайна група G и крайното непразно множество U. Действието на групата G върху множеството U се дефинира с хомоморфизъм $\mu : G \mapsto S(U)$, където $S(U)$ е симетричната група на U. Това действие на G върху U по естествен начин може да се разшири и като действие на G върху фамилията от всички подмножества Ω на U, дефинирано с хомоморфизъм $\gamma : G \mapsto S(\Omega)$. Образът на $X \in \Omega$ под действието на $\gamma_g \in S(\Omega)$ за $g \in G$ ще означаваме с gX. Тогава имаме $(g_1 g_2)X = g_1(g_2 X)$ за всеки $g_1, g_2 \in G$ и $X \in \Omega$.

Групата G разбива елементите на Ω на непресичащи се орбити. Това разбиване дефинира релация на еквивалентност, наречена изоморфизъм. Казваме, че $X, Y \in \Omega$ са изоморфни (или $\sim_G$ еквивалентни), ако съществува $g \in G$ такова, че $gX = Y$. Тогава g наричаме изоморфизъм на X в Y. Изоморфизъм на X в себе си наричаме автоморфизъм. Подгрупата на G, състояща се от всички автоморфизми на X, се нарича група от автоморфизми на X и се бележи с $G_X = Aut(X)$. От друга страна тази група представлява стабилизатор на X по отношение на G. В тази посока може да продължи обсъждането на действието на групата G_X върху X и $U \backslash X$, разгледани като множества от елементи на U, и съответните орбити, породени от това действие.

Казваме, че класификационната задача се дефинира с действие на група върху множество, ако релациите $\sim_G$ и $\sim$ съвпадат върху множеството от разглеждани обекти Г. На практика това разглеждане не съкращава изчисленията при пресмятане на изоморфизъм между комбинаторни обекти, защото мощността на групите е много голяма, но спомага за разработване на ефективни методи за генериране.

Един от начините за установяване дали два обекта са изомофни се базира на концепцията за каноничен представител, която и ние разглеждаме. При нея на всеки елемент от $X \in \Omega$ се съпоставя елемента B от същия клас на еквивалентност (от същата орбита), който има уникални свойства и се нарича канонична форма на X и каноничен представител на класа на еквивалентност на X. По-формално имаме следната дефиниция:

Дефиниция 5. ***Канонично изображение*** *наричаме функция* $\rho: \Omega \mapsto \Omega$, *която има следните свойства:*

1. *за всяко* $X \in \Omega$ *е в сила:* $\rho(X) \sim X$,
2. *за всяко* $X, Y \in \Omega$ *е в сила: от* $X \sim Y$ *следва* $\rho(X) = \rho(Y)$.

Казваме, че обектът X *е в* ***канонична форма****, ако* $\rho(X) = X$.

По този начин задачата за тестване за изоморфизъм се редуцира до по-леката задача за сравняване на съответните канонични форми. Намирането на канонична форма на X обикновено е свързано със следните задачи: намиране на каноничен елемент $c \in G$, такъв че $c(X) = B$; намиране на множество от пораждащи на G_X и орбитите на X по отношение на тази група. На практика тези задачи се решават с помощта на изчерпващи по отношение на подмножество на G алгоритми. Примери за такива алгоритми са представени в [11, 1, 9]. Характерно за тези алгоритми е, че като изход от тях освен каноничен елемент $c \in G$ се получава и съответна на този каноничен елемент наредба *cord* на елементите в X. Тези алгоритми имат добро представяне за много комбинаторни обекти. Въпреки това всяко обръщение към алгоритъм от такъв тип дори и в най-добрия случай изисква значителен изчислителен ресурс и води до забавяне на процеса на генериране. Затова и целта на методите за класификация е от една страна да няма два възела от обхожданото дърво, които да са в един клас на еквивалентност, от друга това да се постига (ако е възможно за съответната задача) с най-малко обръщения към тези алгоритми.

3. Основни подходи

3.1. Запазване на неизоморфните обекти. Този подход е широко използван и единствено възможен в много от случаите. При него се поддържа списък R на представители на класовете на еквивалентност на обектите, обходени до момента. Обектът, който съответства на текущия възел, се тества за изоморфизъм с вече записаните обекти. При отрицателен тест той се добавя в списъка и трябва да бъдат обходени всичките му наследници. Ако се установи, че този обект е изоморфен на друг от R, то не е необходимо обхождане на наследниците му. Този подход е представен в следния алгоритъм:

```
procedure btrF(X:object)
begin
if there exists Z ∈ R such that X ∼ Z then exit
if X is object of interest then output X
```

find $C(X)$;
for all $Y \in C(X)$ do btrF(Y)
$R := R \bigcup \{X\}$
end;
procedure mainF(T:search tree)
begin
$R := \{\}$
btrF($r(T)$)
end.

Този алгоритъм е коректен само ако е изпълнено следното условие: Ако два обекта са изоморфни, то в дървото на търсене за всеки наследник на единия съществува изоморфен наследник на другия и обратно.

Пример 1.1. *Нека е дадено множеството U от всички m-мерни двоични вектор стълбове, които имат k единици. Две подмножества на U наричаме изоморфни, ако елементите на едното могат да се получат от елементите на другото с пермутация на координатите. Да предположим, че $s = kl/m$ е цяло число и да разгледаме следната задача: Да се намерят всички неизоморфни подмножества на U с l елемента, които имат точно s единици във всяка координата. В термините на двоични матрици тази задача може да се дефинира по следния начин: На всеки n елемента на U по естествен начин съпоставяме $m \times n$ двоична матрица A. Казваме, че две матрици са изоморфни, ако стълбовете на едната могат да се получат от стълбовете на другата с пермутация на редовете. Тогава задачата е да се намерят всички неизоморфни $m \times l$ матрици, които имат точно k единици във всеки стълб и s единици във всеки ред. Съгласно терминологията, въведена в предната част, тук можем да разгледаме действието на симетричната група $G = S_m$ върху множествата U и $\Omega = 2^U$. Дървото на търсене T има за корен празното множество. Възлите на първо ниво съответстват на елементите на U и т.н. Нека множеството X от елементи на U е възел на j-то ниво. Тогава негови наследници са множествата $X' = X \cup y$, $y \in U$, такива, че да няма повече от s единици във всяка координата.*

Алгоритъмът ще работи коректно за така построеното дърво. Но той има няколко недостатъка. Първият недостатък е, че ще е необходима много памет при голям брой неизоморфни възли. Освен това отделните клони при обхождането не са независими и алгоритъмът е труден за паралелизация. Друг недостатък е, че ще е необходима канонична форма за всички обходени възли, а техният брой може да е много по-голям от броя на неизоморфните възли. Въпреки че с използването на канонична форма тестването между два обекта се свежда до сравняване, ефективността на алгоритъма драматично намалява, ако броят N на неизоморфните възли е голям. Това може да се преодолее с използване на хеширане с обработка на колизиите. Тогава времето, необходимо за сравняване, е константа.

3.2. Канонично представяне. Друга възможност за постигане на генериране без изоморфни възли е да се обходят само каноничните представители от всеки клас на еквивалентност спрямо подходящо избрано канонично изображение. Т.е. ще се интересуваме само от клоните, които водят от каноничен представител на един клас

(който например има j елемента от U) към каноничен представител на друг клас (който ще има $j+1$ елемента от U). При този подход за канонично изображение се използва функция, която на всеки клас на еквивалентност съпоставя максимален (или минимален) елемент по отношение на някаква наредба (обикновено лексикографска). Реализира се със следния алгоритъм:

procedure btrCR(X:object)
begin
if $X \neq c(X)$ then exit
if X is object of interest then output X
find $C(X)$;
for all $Y \in C(X)$ do btrCR(Y)
end;
procedure mainCR(T:search tree)
begin
btrCR($r(T)$)
end.

За да е коректен алгоритъмът, е необходимо да са изпълнени следните условия: Всеки клас на еквивалентност да има каноничен представител в дървото на обхождане и за всеки възел, който е каноничен представител, родителският му да е също каноничен представител на съответния клас. На практика този алгоритъм не изисква никаква допълнителна памет и тест за изоморфизъм изобщо не се налага. Изпълнението на алгоритъма върху различни клони от дървото е напълно независимо и не използва общ ресурс, така че е лесен за паралелизация и използване на много процесори и/или компютри при решаване на една задача. Недостатъците на този алгоритъм са трудната проверка дали даден обект е каноничен представител спрямо канонично изображение, което е необходимо за случая. От друга страна броят на тези проверки трябва да е поне колкото броя на класовете на еквивалентност.

Пример 1.2. *Продължаваме разглеждането на задачата от Пример* 1.1. *Нека X да е елемент от Ω, а A да е съответната $m \times n$ матрица. Тогава на A съпоставяме вектор $W_A = (a_{1,1}, a_{2,1}, \ldots, a_{m,1}, a_{1,2}, a_{2,2}, \ldots, a_{m,2}, \ldots, a_{1,n}, a_{2,n}, \ldots, a_{m,n})$. За канонично изображение в този случай дефинираме функцията, която на всеки клас на еквивалентност съпоставя елемент B, чийто вектор $W_B = (b_{1,1}, b_{2,1}, \ldots, b_{m,1}, b_{1,2}, b_{2,2}, \ldots, b_{m,2}, \ldots, b_{1,n}, b_{2,n}, \ldots, b_{m,n})$ лексикографски предхожда всички останали (B се нарича минимален лексикографски елемент). Да отбележим, че първите m елемента на вектора W съвпадат с елементите от първия стълб (а не ред) на B и т.н. Това е така, защото на практика разширяването се извършва по стълбове. Алгоритми за намиране и за тестване на минимален елемент са представени в* [8, 10].

3.3. Канонично разширяване. Най-широко използваната в последно време техника за генериране с отхвърляне на изоморфните случаи се свързва с подхода на McKay, представен в [12]. Поради това, че подходът е прекалено общ, ние се спираме и обсъждаме една по-конкретна реализация, която е съобразена с използваната терминология. Нека X е елемент на Ω, $G_X = Aut(X)$ е групата му от автоморфиз-

ми, $c \in G$ е каноничен елемент на групата, а *cord* е съответна на този каноничен елемент наредба на елементите в X. Групата от автоморфизми G_X разбива както множеството X, така и $U \setminus X$ на орбити. Наличието на наредба на елементите в X, съответстваща на $c \in G$, ни позволява да дефинираме една от орбитите на X за *специална*. Това например може да е най-малката по мощност измежду орбитите на X (ако има единствена най-малка). Да дефинираме като *специална* орбитата, в която се намира последният елемент на X при наредбата *cord*. Тази орбита и елементите ѝ могат да се определят за всеки разглеждан обект и всеки обект от класа на еквивалентност на X, като му се намери канонична форма, а оттам и $c \in G$ и *cord*. При нашите разглеждания нов обект се получава от родителския X с добавяне на подходящ елемент $x' \in U$. За канонично разширяване използваме *родителски тест* или *parent test*. Ще казваме, че $X \cup x'$ удовлетворява родителския тест, ако $x' \in U$ е в специална орбита по отношение на $G_{X \cup x'}$.

```
procedure btrCE(X:object)
begin
from each orbit of G_X on U \ X choose one child { if exists }
for every Y = X ∪ x' of these children do
if Y pass the parent test then
if Y is object of interest then output Y else
btrCE(Y);
end;
procedure mainCE(T:search tree)
begin
btrCE(r(T))
end.
```

Понякога е трудно да се намерят представители на всяка орбита на $U \setminus X$ при действието на G_X. За това може да се използва идентичният алгоритъм:

```
procedure btrCE(X:object)
begin
if X is object of interest then output X; exit;
find in C(X) all passing parent test and save them in M(X);
find all inequivalent S(X) from M(X);
for all Y ∈ S(X) do btrCE(Y)
end;
procedure mainCE(T:search tree)
begin
btrCE(r(T))
end.
```

Коректността на тези алгоритми следва от общата теория на McKay, а доказателства на реализации, идентични на дадените, са представени в [2] и [14]. Голямото предимство на този тип алгоритми е, че в много случаи е възможно предефиниране на специалната орбита чрез използване на инварианти, пресмятането на които е

много по-лесно от пресмятането на канонична форма. Под инварианта на елемент на X относно G_X разбираме функция $Inv : U \to \mathbb{N}$ такава, че ако X_i и X_j са в една орбита по отношение на G_X, то $Inv(X_i) = Inv(X_j)$. Всяка инвариантa разделя елементите на X на подмножества, които наричаме псевдоорбити, състоящи се от една или няколко орбити на X при действието на G_X. С използване на инварианта можем да предефинираме специална орбита, като за това има две стратегии. При първата, за *специална* дефинираме орбита O, която се намира в най-малката псевдоорбита, чийто последен елемент, определен от *cord*, принадлежи на O. При втората стратегия се избира орбита от псевдоорбитата, за която стойността на Inv е максимална.

Пример 1.3. *Да дефинираме инварианта на стълб от матрицата A, която съответства на $X \in \Omega$ от Пример* 1.2. *За j-тия стълб инварианта представлява общият брой на единиците във всички редове на матрицата A, в които този стълб има единица. Други примери за инварианта са представени в* [1].

4. Примери за резултати, получени с описаните подходи. Обсъдените подходи имат много широко приложение и ние ще се спрем само на някои примери. Не винаги е възможно директно приложение на само един от методите. Понякога е необходимо и съчетаване на няколко от тези методи с други комбинаторни техники. Задълбочен и прецизен обзор на много широк кръг задачи е направен в монографията на Kaski и Östergård [8]. Там могат да се намерят формални доказателства и на алгоритмите от 3.1 и 3.2. Класификация на различни типове графи е представена в [12], а някои приложения за обекти, свързани с крайни геометрии, са дадени в [14]. Много атрактивно решение на задача от този тип е представено в [7]. Генериране на орбитни матрици с каноничен представител е направено в [6], а ортогонални резолюции на 2-дизайни в [15].

Сега ще се спрем на две по-конкретни задачи. Едната се отнася до класификация на двоични линейни кодове с параметри $[56, 7, 26]$ като част от задачата за класификация на оптималните кодове с размерност 7. Тази задача беше атакувана в съвместен проект с Jaffe [4] с алгоритъма, описан в [3] през 2000 година, който представлява вариант на метода от 3.1. Тогава беше установено, че броят на неекивалентните кодове с такива параметри е над 19549 на 500 MHz Alpha 21264 процесор. Беше оценено, че е необходима повече от година процесорно време за цялостно решаване на задачата. С модификация на същия алгоритъм с добавяне на хеширане, при което се използват 100000 клетки с динамична дължина, тази задача беше решена за около 60 часа на процесор Atlon 1800 MHz. Беше установено, че точният брой на тези кодове е 20197.

Другата задача, на която ще се спрем по-подробно, е класификацията на всички двоични проективни кодове с размерност 6. Нека множеството U се състои от точките на $PG(5, 2)$. Да разгледаме действието на $G = PGL(6, 2)$ върху това множество и неговите подмножества. Тогава задачата е идентична с намирането на всички неизоморфни множества от точки от $PG(5, 2)$. Тази задача беше решена с използване на алгоритъм за канонично разширяване. Алгоритъмът беше изпълнен до дълбочина 32 за около 80 часа на 1800 MHz PC. Останалите неизоморфни множества с мощност до 63 се получават като допълнение на получените. Конструирани са 284625281 неизоморфни множества с мощност до 32. Броят на всички разгледани множества,

получени като наследници, е 8252302118. За тези множества е пресметната подходяща инварианта f. Броят на множествата, за които е пресмятана канонична пермутация, е 331742121. Допълнителна информация е представена в [2]. Този резултат в последствие се оказа много полезен за намиране на всички точни стойности на функцията $t_2(n, 6)$.

REFERENCES

[1] I. Bouyukliev. About the code equivalence. In: Advances in coding theory and cryptography (Eds T. Shaska et al.) Selected papers based on the presentations at the Vlora conference in coding theory and cryptography, Vlora, Albania, May 26–27, 2007 and a special session on coding theory as part of the applications of computer algebra conference, Rochester, MI, USA, July 19–22, 2007. Hackensack, NJ: World Scientific. Series on Coding Theory and Cryptology, **3**, 2007, 126–151.

[2] I. Bouyukliev. On the binary projective codes with dimensions 6. *Discrete Applied Mathematics*, **154** (2006), 1693–1708.

[3] I. Boukliev. 'Q-EXTENSION' – strategy in algorithms. Proceedings of the International Workshop ACCT, Bansko, Bulgaria, 2000, 84–89.

[4] I. Bouyukliev, D. Jaffe. Optimal binary linear codes of dimension at most seven. *Discrete Mathematics*, **226**, No 1–3, (2001), 51–70.

[5] I. A. Faradzev. Constructive enumeration of combinatorial objects. Problemes Combinatoires et Theorie des Graphes, University d'Orsay, July 9–13, 1977, CNRS, Paris, 1978, 131—135.

[6] Ст. Капралов. Алгоритми за генериране и изследване на орбитни матрици. 100 години от рождението на академик Любомир Чакалов, Трудове на юбилейна научна сесия, Самоков, 14–15.02.1986, 114–121.

[7] K. Coolsaet, H. Sticker. A full classification of the complete k-arcs of PG(2, 23) and PG(2, 25). *Journal of Combinatorial Designs* (to appear).

[8] P. Kaski, P. R. Östergård. Classification Algorithms for Codes and Designs. Springer, 2006.

[9] W. Kocay. On writing isomorphism programs. In: Computational and Constructive Design Theory (ed. W. D. Wallis). Kluwer, 1996, 135–175.

[10] Zl. Mateva. Constructing canonical form of a matrix in several problems about combinatorial designs. *Serdica Journal of Computing* (to appear).

[11] B. D. McKay. Practical graph isomorphism. *Congr. Numer.*, **30** (1981), 45–87.

[12] B. D. McKay. Isomorph-free exhaustive generation. *J. Algorithms*, **26** (1998), 306–324.

[13] R. C. Read. Every one a winner; or, How to avoid isomorphism search when cataloguing combinatorial configurations. *Ann. Discrete Math.*, **2** (1978), 107—120.

[14] G. F. Royle. An orderly algorithm and some applications in finite geometry. *Discrete Math.*, **185**, No 1–3 (1998), 105–115.

[15] S. Topalova, S. Zhelezova. Doubly resolvable designs with small parameters, preprint.

Илия Буюклиев
Институт по математика и информатика
МОИ – В. Търново – БАН
5000 В. Търново, Р.К. 323

ABOUT ALGORITHMS FOR ISOMORPHISM-FREE GENERATIONS OF COMBINATORIAL OBJECTS

Ilya Bouyukliev

There are three types of approaches for isomorph-free generation. The first technique for isomorphism rejection is the approach of keeping a record of the isomorphism classes of objects seen so far during traversal of the search tree. The next one, known as "generation by canonical representative", uses canonical representatives which are extremal elements of equivalence classes relative to a (lexicographic) order on Ω. Generation by canonical representatives was introduced independently by Faradjev and Read. The other method, known as "canonical augmentation" was introduced by McKay. In this paper, we present basis concepts of these three approaches.