Liet. matem. rink., T. 40, spec. nr., 320–328

Computing permutation groups of error-correcting codes

Gintaras SKERSYS (VU)
e-mail: gintaras.skersys@maf.vu.lt

1. Presentation of the problem

1.1. *Error-correcting codes*

If we transmit data, e.g., a string of bits, over a channel, there is some probability that the received message will not be identical to the transmitted message. In order to reduce the probability of errors, the original message is *encoded* before the transmission, adding *redundancy* to it in some way, and *decoded* using this redundancy after the transmission. We can do this in the following way.

Let $\mathbf{F}_q$ be the finite field with q elements. An $[n, k]$ *linear code* C *over* $\mathbf{F}_q$ is a k-dimensional linear subspace of $\mathbf{F}_q^n$. The parameters n and k are called the *length* and the *dimension* of C, respectively. The elements of C are called *codewords*.

Let's take a basis of the linear subspace C and let's form a matrix G where the rows of G are the basis vectors of C. The matrix G is called a *generator matrix* of C.

We suppose that the original data is a stream of elements of $\mathbf{F}_q$. We divide this stream into words $\mathbf{v}$ of length k (row vectors of $\mathbf{F}_q^k$), and we encode each word separately. Encoding a word $\mathbf{v}$ consists of multiplying it by the generator matrix G, i.e., we compute $\mathbf{x} = \mathbf{v}G \in C$, and we transmit it. The channel adds noise to the transmitted codeword, yielding a received vector $\mathbf{y} = \mathbf{x}+\mathbf{e}$, where $\mathbf{e} \in \mathbf{F}_q^n$ is an error vector. The received vector $\mathbf{y}$ is decoded by searching the closest codeword $\mathbf{x}'$ of C, where the distance between two vectors of $\mathbf{F}_q^n$ is the *Hamming distance* (the number of positions in which the vectors differ), and by computing the word $\mathbf{v}' \in \mathbf{F}_q^k$ from the equation $\mathbf{x}' = \mathbf{v}'G$.

This procedure can correct at least $\lfloor \frac{d-1}{2} \rfloor$ errors, where d is the *minimum distance* of C, i.e., the minimum Hamming distance between any two distinct codewords of C.

For future reference we give here some more definitions related to linear codes.

Let C be an $[n, k]$ linear code. The *Hamming weight* of a vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbf{F}_q^n$ is the number of nonzero components v_i. The *minimum weight* of C is the minimum Hamming weight of the nonzero codewords of C. Let A_i be the number of codewords of C having Hamming weight equal to i. The *weight enumerator* of C is the polynomial $\mathcal{W}(C) = \sum_{i=0}^{n} A_i X^i$.

If $\mathbf{u} = (u_1, \ldots, u_n)$, $\mathbf{v} = (v_1, \ldots, v_n)$ are vectors of $\mathbf{F}_q^n$, their *scalar product* is $\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + \cdots + u_n v_n$. The *dual code* $C^\perp$ of C is defined to be $C^\perp = \{\mathbf{u} \in \mathbf{F}_q^n \mid$

$\mathbf{u} \cdot \mathbf{v} = 0$ for all $\mathbf{v} \in C\}$. The *hull* $\mathcal{H}(C)$ of C is its intersection with its dual code, i.e., $\mathcal{H}(C) = C \cap C^{\perp}$.

Let J be a subset of $\{1, \ldots, n\}$. The code C *punctured* in J, denoted by C_J, consists of all elements of C where the coordinates indexed by J are replaced by zeros.

1.2. *Equivalence and permutation group*

For the purpose of studying the error-correcting properties of linear codes for any channel with independent errors, two linear codes that differ only in the arrangement of symbols have the same probability of error. Two such linear codes are called equivalent. More precisely, let Ω be a set of size n used to index the coordinates of the vectors of $\mathbf{F}_q^n$, and let $Sym(\Omega)$ be the symmetric group on Ω. We denote by i^g the image of $i \in \Omega$ under the action of the permutation $g \in Sym(\Omega)$. A permutation $g \in Sym(\Omega)$ acts on a vector $\mathbf{v} = (v_i)_{i \in \Omega} \in \mathbf{F}_q^n$ as follows: $\mathbf{v}^g = \left(v_{i^{g^{-1}}}\right)_{i \in \Omega}$, where g^{-1} is the inverse permutation of g. Two codes C and C' of length n are *equivalent* if there exists such a permutation $g \in Sym(\Omega)$ that $C' = C^g$, where $C^g = \{\mathbf{c}^g \mid \mathbf{c} \in C\}$. The *permutation group* $Perm(C)$ of a linear code C of length n is the subgroup of all the elements g of $Sym(\Omega)$ such that $C^g = C$.

The study of permutation groups and equivalence of linear codes is an interesting problem of the theory of error-correcting codes. The knowledge of the permutation groups helps to classify the codes, to compute their weight enumerators. Moreover, some decoding algorithms use the permutations. The permutation groups of most classic codes are rather particular (the Mathieu group, the linear group, the affine group, the projective special linear group, etc.). The equivalence of linear codes is used in some cryptosystems based on the theory of error-correcting codes (see [4, 6]).

This article outlines an efficient algorithm for computing permutation groups and determining equivalence of linear codes. This algorithm combines two existing algorithms: an algorithm of Jeffrey S. Leon [1, 2, 3] and the support splitting algorithm (SSA) of Nicolas Sendrier [8].

1.3. *Leon's algorithm*

Leon's algorithm (LA) is practically the only algorithm for computing the permutation group of a linear code and determining equivalence of two linear codes. It is used in such computer algebra systems like Magma and Gap. Let $\mathcal{P}$ be a *property* (i.e., a Boolean-valued function) on $Sym(\Omega)$ that is readily computable. LA computes the set $Sym(\Omega)_{\mathcal{P}} \stackrel{\text{def}}{=} \{g \in Sym(\Omega) \mid \mathcal{P}(g)\}$. For instance, if $\mathcal{P}_C$ is the property such that $\mathcal{P}_C(g)$ holds exactly when $g \in Perm(C)$, then LA computes $Sym(\Omega)_{\mathcal{P}_C} = Perm(C)$.

LA employs *backtrack search*. It makes use of the concepts of *base and strong generating set*, developed by Charles C. Sims [10], to facilitate pruning of the *search tree* associated with backtracking. In addition, it employs successive *refinement of ordered partitions*, introduced by Brendan D. McKay [5], to facilitate choice of a smaller and

more nearly optimal base and to allow more extensive pruning of the backtrack search tree.

The method of refining the ordered partitions depends on a particular property $\mathcal{P}$. Apart from the other input parameters, LA accepts a property $\mathcal{P}$ and a set of mappings, related to $\mathcal{P}$, called $\mathcal{P}$-refinements, that specify how to refine the ordered partitions. In his papers [2, 3], Leon proposes sets of $\mathcal{P}$-refinements for several properties $\mathcal{P}$, among them for the properties $\mathcal{P}_C$ and $\mathcal{P}_{C,C'}$ ($\mathcal{P}_{C,C'}(g)$ holds exactly when $C^g = C'$). But his $\mathcal{P}_C$- and $\mathcal{P}_{C,C'}$-refinements are not peculiar to linear codes. They are destined to compute the permutation group of a matrix and determine the equivalence of two matrices, respectively (in p. 323 we describe how a permutation acts on a matrix). In order to compute the permutation group of a linear code C, we must provide LA with a matrix W such that $Perm(C) \subset Perm(W)$. LA finds the permutations of $Perm(W)$ and verifies if they lie in $Perm(C)$. In that way LA obtains $Perm(C)$.

To construct such a matrix W, we may consider, for instance, the set of all codewords of C, $C^\perp$ or $\mathcal{H}(C)$, the set of minimum weight (or constant weight) codewords of C, $C^\perp$ or $\mathcal{H}(C)$, the union of some of these sets, etc., and form a matrix W whose rows are the vectors of such a set. But W must be "reasonably small" (small enough to permit computation in a permutation group of degree $n + m$, where m is the number of rows of W), and $|Perm(W) : Perm(C)|$ must be very small. Given a large linear code C it is difficult to compute such a matrix W. Therefore LA is limited to relatively small linear codes for which one can "easily" find such a matrix, for example, in binary case (i.e., when the finite field is $\mathbf{F}_2 = \{0, 1\}$) to the linear codes of length up to 100 or dimension up to 50 – with a few exceptions.

1.4. *The support splitting algorithm*

On the other hand, N. Sendrier created an algorithm (the SSA) for determining if two linear codes are equivalent, and finding the permutation between them if they are (see [8]). The SSA only works if the permutation groups of the considered codes are *trivial*, that is, reduced to the identity permutation. Note that in this case the permutation between two equivalent codes is unique. Moreover, the hulls of the considered codes must be small, for the SSA computes the weight enumerators of the hulls (for example, in the binary case the dimension of the hull must be less than 20–30). This condition holds for most linear codes since the average dimension of the hull of linear codes is a small positive constant [7], but there exist some important families of linear codes in which many codes have a big hull [11]. When both abovementioned conditions are satisfied, the SSA is very efficient, for instance, it can determine the equivalence of two binary linear codes of length up to several thousands.

By generalizing the SSA, we were able to construct sets of $\mathcal{P}_C$- and $\mathcal{P}_{C,C'}$-refinements for LA. This highly extends the domain of application of LA for the properties $\mathcal{P}_C$ and $\mathcal{P}_{C,C'}$, for example, now we can compute $Perm(C)$ and find a permutation g such that $C^g = C'$ for two binary linear codes C and C' of length up to several thousands, provided that the hulls of C and C' are small enough.

2. Leon's algorithm

In this section we will show by an example how LA works.

We begin by some definitions. Let Ω be a set of size n. A *partition* Π of Ω is a collection of disjoint non-empty subsets of Ω whose union is Ω. The elements of Π are called its *cells*. An *ordered partition* of Ω is a sequence $(\Pi_1, \Pi_2, \ldots, \Pi_l)$ for which $\{\Pi_1, \Pi_2, \ldots, \Pi_l\}$ is a partition. The set of all ordered partitions of Ω will be denoted by $PartOrd(\Omega)$. If Π is a partition (ordered or not), the number of cells of Π is denoted by $|\Pi|$. Π is called *discrete* if $|\Pi| = n$. If $|\Pi| + 1 \leqslant i \leqslant n$, Π_i will denote the empty set.

We will need to compare the ordered partitions. For this we define an ordering. If $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_l)$ and $\Sigma = (\Sigma_1, \Sigma_2, \ldots, \Sigma_m)$ are ordered partitions, we define $\Pi \leqslant \Sigma$ to mean (1) $|\Pi| \geqslant |\Sigma|$, (2) $\Pi_i \subseteq \Sigma_i$ for $i \leqslant |\Sigma|$, and (3) Π_i, $i > |\Sigma|$, is contained in some cell of Σ. If $\Pi \leqslant \Sigma$ and $\Pi \neq \Sigma$, we write $\Pi < \Sigma$ and say that Π is *finer* than Σ.

We give the formal definition of a $\mathcal{P}$-refinement.

DEFINITION 1. [2, Def. 8 and 3, Def. 9] If $\mathcal{P}$ is a property, a *$\mathcal{P}$-refinement* E is a pair $(E_{\mathsf{L}}, E_{\mathsf{R}})$ of mappings of $PartOrd(\Omega)$ into $PartOrd(\Omega)$ such that, for all $\Pi, \Sigma \in PartOrd(\Omega)$ and for all $g \in Sym(\Omega)$, the following conditions hold:

(a) $E_{\mathsf{L}}(\Pi) \leqslant \Pi$ and $E_{\mathsf{R}}(\Pi) \leqslant \Pi$.

(b) $|E_{\mathsf{L}}(\Pi)| \leqslant |\Pi| + 1$ and $|E_{\mathsf{R}}(\Pi)| \leqslant |\Pi| + 1$.

(c) If $\mathcal{P}(g)$ and if $\Pi^g = \Sigma$, then $E_{\mathsf{L}}(\Pi)^g = E_{\mathsf{R}}(\Sigma)$.

Note [2, p. 542]. The subscripts "L" and "R" are formal symbols chosen to suggest left and right, corresponding to the appearance of E_{L} and E_{R} on the left side and the right side, respectively, of the equation in (c) above. Note that (a) and (b) imply that either $E_{\mathsf{z}}(\Pi) = \Pi$ or $|E_{\mathsf{z}}(\Pi)| = |\Pi| + 1$ for $\mathsf{z} \in \{\mathsf{L}, \mathsf{R}\}$; that is, each component of a $\mathcal{P}$-refinement either leaves Π unchanged or splits exactly one of its cells.

Thus, we can describe the action of a $\mathcal{P}$-refinement by precising this cell and a part of it which splits off. More precisely, if $\Pi = (\Pi_1, \ldots, \Pi_l) \in PartOrd(\Omega)$, $1 \leqslant i \leqslant n$, $\Gamma \subset \Omega$, then we define

$$\mathcal{F}_{i,\Gamma}(\Pi) = \begin{cases} (\Pi_1, \ldots, \Pi_{i-1}, \Pi_i \setminus \Gamma, \Pi_{i+1}, \ldots, \Pi_l, \Pi_i \cap \Gamma) & \text{if } 1 \leqslant i \leqslant l \text{ and} \\ & \emptyset \subsetneq \Pi_i \cap \Gamma \subsetneq \Pi_i, \\ \Pi & \text{otherwise.} \end{cases}$$

To simplify, we will only show how LA computes $Perm\,(C)$ for a binary linear code C. Let $A = (a_{ij})$ be $m \times l$ binary matrix whose columns and rows are indexed by $\Omega_{\mathsf{C}} \stackrel{\text{def}}{=} \{1, \ldots, l\}$ and $\Omega_{\mathsf{R}} \stackrel{\text{def}}{=} \{l+1, \ldots, l+m\}$, respectively, upon which each permutation g of $\Omega \stackrel{\text{def}}{=} \Omega_{\mathsf{C}} \cup \Omega_{\mathsf{R}}$ stabilizing $(\Omega_{\mathsf{C}}, \Omega_{\mathsf{R}})$ acts by moving column i to column position i^g and row j to row position j^g. Let n be the size of Ω. Let $\mathcal{P}_A$ be the property such that $\mathcal{P}_A(g)$

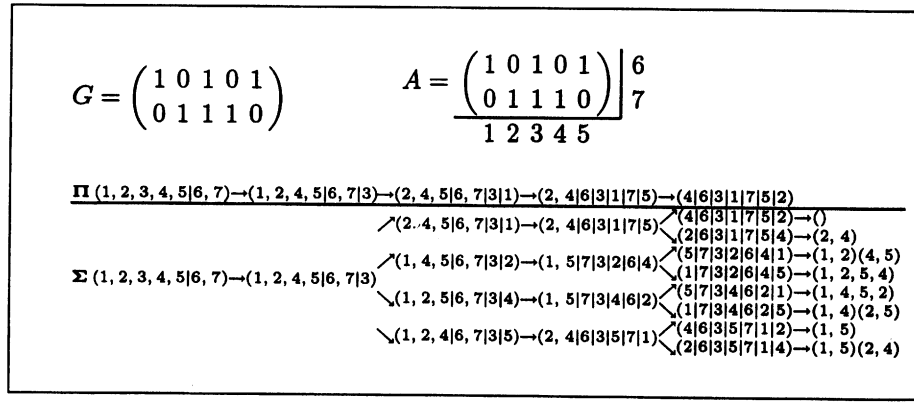

Fig. 1. An example of backtrack search tree with Leon's refinements.

holds exactly when $A^g = A$. Leon proposes (see [2, §9(f)] and [3, Fig. 2] the set $\mathbf{I}_A$ of $\mathcal{P}_A$-refinements, where $\mathbf{I}_A = \{(I_{A,i,j,c}, I_{A,i,j,c}) \mid 1 \leqslant i, j \leqslant n, 0 \leqslant c \leqslant n\}$ and

$$I_{A,i,j,c}(\Pi) = \begin{cases} \mathcal{F}_{i,\Xi}(\Pi) & \text{where } \Xi = \{\xi \in \Omega_C \mid \sum_{\mu\in\Pi_j} a_{\mu\xi} = c\}, \\ & \text{if } \Pi_i \subseteq \Omega_C \text{ and } \Pi_j \subseteq \Omega_R, \\ \mathcal{F}_{i,\Lambda}(\Pi) & \text{where } \Lambda = \{\lambda \in \Omega_R \mid \sum_{\mu\in\Pi_j} a_{\lambda\mu} = c\}, \\ & \text{if } \Pi_i \subseteq \Omega_R \text{ and } \Pi_j \subseteq \Omega_C, \\ \Pi & \text{otherwise.} \end{cases}$$

Let C be a $[5, 2]$ binary linear code generated by the matrix G (see Fig. 1). We form the matrix A from the minimum weight codewords of C (see Fig. 1). We index the columns of A by $1, \ldots, 5$, and the rows by 6 and 7. Ordered partition constants will be written using simplified notation, illustrated as follows: the ordered partition $(\{1, 5\}, \{2\}, \{3, 4, 7\}, \{6\})$ will be written as $(1, 5|2|3, 4, 7|6)$.

We want to compute $Perm\,(C)$. We compute $Sym\,(\Omega)_{\mathcal{P}_A}$, and when we find a permutation $g \in Sym\,(\Omega)_{\mathcal{P}_A}$, we verify if the projection $g|_{\Omega_C}$ of g to Ω_C lies in $Perm\,(C)$.

We will walk through the search tree of Fig. 1. As we progress in the tree the partitions become finer and finer, until we reach the leaves labeled by discrete partitions. Note that if ordered partitions Π and Σ are discrete we can easily recover $g \in Sym\,(\Omega)$ such that $\Pi^g = \Sigma$.

We begin with $\Pi = \Sigma = (\Omega_C, \Omega_R) = (1, 2, 3, 4, 5|6, 7)$. Evidently, $\Pi^g = \Sigma$ for all $g \in Sym\,(\Omega)_{\mathcal{P}_A}$. We try to refine Π and Σ in such a way that this condition remains satisfied. For this we apply $\mathcal{P}_A$-refinements of $\mathbf{I}_A$ to Π and Σ (that is, we apply the first component of a $\mathcal{P}_A$-refinement to Π, and the second to Σ) trying to refine them.

We notice that $I_{A,1,2,2}(\Pi) = \mathcal{F}_{1,\{3\}}(\Pi) = (1, 2, 4, 5|6, 7|3) < \Pi$. We apply $I_{A,1,2,2}$ to both Π and Σ and we get new partitions $\Pi = \Sigma = (1, 2, 4, 5|6, 7|3)$ (the second column of the search tree in Fig. 1). Note that $\Pi^g = \Sigma$ for all $g \in Sym\,(\Omega)_{\mathcal{P}_A}$ because of Def. 1(c). Then we try to apply other $\mathcal{P}_A$-refinements of $\mathbf{I}_A$, but we cannot refine Π and Σ any more.

At this point we divide the computation of $Sym(\Omega)_{\mathcal{P}_A}$ into four cases: we fix one point of Ω, e.g., we choose $1 \in \Omega$, and we begin by computing all $g \in Sym(\Omega)_{\mathcal{P}_A}$ such that $1^g = 1$, then all $g \in Sym(\Omega)_{\mathcal{P}_A}$ such that $1^g = 2$, $1^g = 4$ and $1^g = 5$ (for any $g \in Sym(\Omega)_{\mathcal{P}_A}$ we have not $1^g = 3$, for 1 and 3 lie in different cells of Π, and $\Pi^g = \Sigma = \Pi$ for all $g \in Sym(\Omega)_{\mathcal{P}_A}$). But, for instance, $\Pi^g = \Sigma$ and $1^g = 4$ hold exactly when $\mathcal{F}_{1,\{1\}}(\Pi)^g = \mathcal{F}_{1,\{4\}}(\Sigma)$, that is, $(2,4,5|6,7|3|1)^g = (1,2,5|6,7|3|4)$. We get four branches in the search tree (the third column in Fig. 1). We advance in the first branch: Π and Σ become equal to $\mathcal{F}_{1,\{1\}}(\Pi) = \mathcal{F}_{1,\{1\}}(\Sigma) = (2,4,5|6,7|3|1)$. We try again the $\mathcal{P}_A$-refinements of $\mathbf{I}_A$. We find that $I_{A,2,4,0}(\Pi) = \mathcal{F}_{2,\{7\}}(\Pi) = (2,4,5|6|3|1|7) < \Pi$. We apply $I_{A,2,4,0}$ to both Π and Σ and we get now $\Pi = \Sigma = (2,4,5|6|3|1|7)$. Then we notice that $I_{A,1,5,0}(\Pi) = \mathcal{F}_{1,\{1,5\}}(\Pi) = (2,4|6|3|1|7|5) < \Pi$, we apply $I_{A,1,5,0}$ to Π and Σ and we get new $\Pi = \Sigma = (2,4|6|3|1|7|5)$. We try to apply some more $\mathcal{P}_A$-refinements of $\mathbf{I}_A$, but being unable to refine Π and Σ any more we stop trying. Note that we need not to try all $\mathcal{P}_A$-refinements of $\mathbf{I}_A$. Usually we stop after a given number of tries, because the set $\mathbf{I}_A$ is often too large.

Then we divide the search into two cases as above. At last we get two discrete partitions Π and Σ (the fifth column in Fig. 1), from which we recover the identity permutation () that surely lies in $Perm(C)$. Then we return to the fourth column in Fig. 1 and we choose another branch, we get another permutation $(2,4) \in Perm(C)$, we return to second column in Fig. 1, we choose another branch, we apply the same mappings $I_{A,2,4,0}$ and $I_{A,1,5,0}$ to $\Sigma = (1,4,5|6,7|3|2)$, we get new $\Sigma = (1,5|7|3|2|6|4)$, and so on. We obtain all eight permutations of $Perm(C)$.

Usually we want to find only a set of generators of $Perm(C)$. There exist some results, due to Sims [10] and generalized by Leon [2, Prop. 8], which allow us to prune the search tree. After that we get a tree which gives only a particular set of generators, called *strong generating set.*

3. Support splitting algorithm of Sendrier

In [8] Sendrier introduces the notion of *signature.* Let Ω be a set of size n used to index the coordinates of the vectors of $\mathbf{F}_q^n$, let C be an $[n,k]$ linear code. A *signature* S over a set F maps a code C and an element i of Ω into an element of F and is such that for all permutations g of Ω, $S(C^g, i^g) = S(C,i)$. It is said to be *discriminant* for C if for some i and j in Ω, we have $S(C,i) \neq S(C,j)$. It is said to be *fully discriminant* for C if for all i and j distinct in Ω, $S(C,i) \neq S(C,j)$.

An *invariant* is a mapping $\mathcal{V}$ such that any two equivalent codes take the same value. The mapping $(C,i) \mapsto \mathcal{V}(C_{\{i\}})$ is a signature.

Let g be a permutation of Ω, let $C' = C^g$ be a linear code equivalent to C, let S be a signature fully discriminant for C. Then S is also fully discriminant for C' and we can recover g from the following remark: $i^g = j$ if and only if $S(C,i) = S(C',j)$.

If S is not fully discriminant for C, we try to get more discriminant signature as follows: if T is a signature over H and L is a subset of H, then the mapping $(C,i) \mapsto$

$(S(C,i), T(C_{K_{S,L}(C)}, i))$, where $K_{S,L}(C) = \{j \in \Omega \mid S(C,j) \in L\}$, is a signature, which is at least as discriminant as S for C. Hopefully, it is more discriminant than S, and after a few such operations we obtain a signature fully discriminant for C, if it exists.

Note that there exists a fully discriminant signature for C if and only if $Perm(C)$ is trivial.

The weight enumerator of the hull of a linear code is an invariant, which is, for most linear codes, easy to compute and discriminant [7]. Sendrier proposes to use the signature S constructed from this invariant:

$$\mathbf{S}(C,i) = \left(\mathcal{W}(\mathcal{H}(C_{\{i\}})), \mathcal{W}\left(\mathcal{H}\left((C^{\perp})_{\{i\}}\right)\right)\right). \tag{1}$$

4. SSA-based refinements

We generalize the notion of signature in the following way:

DEFINITION 2. Let C be an $[n,k]$ linear code, let Π be an ordered partition of Ω, let F be a set, and let $j \in \Omega$. A mapping R which associates Π, C and j to an element of F is called *generalized signature* over F if $\mathsf{R}(\Pi^g, C^g, j^g) = \mathsf{R}(\Pi, C, j)$ for all $g \in Sym(\Omega)$.

Theorem 1 [11, §3.4]. *Let $e \in F$, let $1 \leqslant i \leqslant n$. Let $Q_{C,\mathsf{R},i,e}$ be the mapping of $PartOrd(\Omega)$ into $PartOrd(\Omega)$ defined by $Q_{C,\mathsf{R},i,e}(\Pi) = \mathcal{F}_{i,\Phi}(\Pi)$, where $\Phi = \{j \in \Omega \mid \mathsf{R}(\Pi, C, j) = e\}$. The pairs $(Q_{C,\mathsf{R},i,e}, Q_{C,\mathsf{R},i,e})$ and $(Q_{C,\mathsf{R},i,e}, Q_{C',\mathsf{R},i,e})$ are $\mathcal{P}_C$- and $\mathcal{P}_{C,C'}$-refinements, respectively.*

Let P and Q be generalized signatures. Then $\mathsf{P}^{\perp}$, defined by $\mathsf{P}^{\perp}(\Pi, C, i) = \mathsf{P}(\Pi, C^{\perp}, i)$, and $\mathsf{P} \times \mathsf{Q}$, defined by $\mathsf{P} \times \mathsf{Q}(\Pi, C, i) = (\mathsf{P}(\Pi, C, i), \mathsf{Q}(\Pi, C, i))$, are generalized signatures. Thus, $\overline{\mathsf{R}} \stackrel{\text{def}}{=} \mathsf{R} \times \mathsf{R}^{\perp}$ is also a generalized signature.

Let S be a signature over a set F, let $\Pi \in PartOrd(\Omega)$, let L be a subset of $\{1,\ldots,n\}$ and let $j \in \Omega$. We introduce the generalized signature $\mathsf{T}_{S,L}$ defined by $\mathsf{T}_{S,L}(\Pi, C, j) = S\left(C_{\bigcup_{i \in L} \Pi_i}, j\right)$. For LA we propose the sets $\mathbf{Q}_C$ and $\mathbf{Q}_{C,C'}$ of $\mathcal{P}_C$- and $\mathcal{P}_{C,C'}$-refinements, respectively, where

$$\mathbf{Q}_C = \left\{\left(Q_{C,\overline{\mathsf{T}}_{\mathbf{S},L},i,e}, Q_{C,\overline{\mathsf{T}}_{\mathbf{S},L},i,e}\right) \mid L \subset \{1,\ldots,n\}, 1 \leqslant i \leqslant n, e \in \mathbb{Z}[X]^4\right\},$$
$$\mathbf{Q}_{C,C'} = \left\{\left(Q_{C,\overline{\mathsf{T}}_{\mathbf{S},L},i,e}, Q_{C',\overline{\mathsf{T}}_{\mathbf{S},L},i,e}\right) \mid L \subset \{1,\ldots,n\}, 1 \leqslant i \leqslant n, e \in \mathbb{Z}[X]^4\right\},$$

and S is defined in Eq. (1) (we take e from $\mathbb{Z}[X]^4$ since $\overline{\mathsf{T}}_{\mathbf{S},L}(\Pi, C, j)$ is a sequence of four weight enumerators).

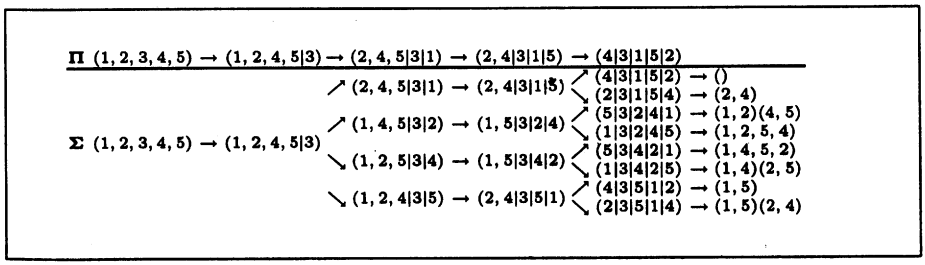

Fig. 2. An example of backtrack search tree with SSA-based refinements.

In Fig. 2 we give an example of backtrack search tree with SSA-based $\mathcal{P}_C$-refinements, where C is the same linear code as in Fig. 1. To simplify, we use $Q_{C,\mathsf{T}_{S,L},i,e}$ with the signature $S : (C, j) \mapsto \mathcal{W}(\mathcal{H}(C_{\{j\}}))$. Since

$$\mathcal{W}(\mathcal{H}(C_{\{j\}})) = \begin{cases} 1 & \text{if } j = 1, 2, 4, 5, \\ 1 + 2X^2 + X^4 & \text{if } j = 3, \end{cases}$$

we pass from the first column of Fig. 2 to the second by means of the mapping $Q_{C,\mathsf{T}_{S,\emptyset},1,1+2X^2+X^4}$. Likewise, since

$$\mathcal{W}(\mathcal{H}(C_{\{1,j\}})) = \begin{cases} 1 & \text{if } j = 1, 2, 4, \\ 1 + X^2 & \text{if } j = 3, 5, \end{cases}$$

we use $Q_{C,\mathsf{T}_{S,\{3\}},1,1+X^2}$ to pass from the third column to the fourth. The rest is performed as in Fig. 1.

References

[1] J. Leon, Computing automorphism groups of error-correcting codes, *IEEE Transactions on Information Theory*, **IT-28**(3), 496–511 (1982).

[2] J. Leon, Permutation group algorithms based on partitions, I: Theory and algorithms, *J. Symbolic Computation*, **12**, 533–583 (1991).

[3] J. Leon, Partitions, refinements, and permutation group computation, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **28**, 123–158 (1997).

[4] R. J. McEliece, A public-key cryptosystem based on algebraic coding theory, *DSN Progress Report*, Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pp. 114–116 (1978).

[5] B. McKay, Computing automorphisms and canonical labellings of graphs, *Lecture Notes in Math.*, **686**, Springer-Verlag, Berlin and New York, pp. 223–232 (1978).

[6] H. Niederreiter, Knapsack-type cryptosystems and algebraic coding theory, *Problems of Control and Information Theory*, **15**(2), 157–166 (1986).

[7] N. Sendrier, On the dimension of the hull, *SIAM Journal on Applied Mathematics*, **10**(2), 282–293 (1997).

[8] N. Sendrier, Finding the permutation between equivalent binary codes, In: *IEEE Conference, ISIT'97*, Ulm, Germany (1997).

[9] N. Sendrier, and G. Skersys, Permutation groups of error-correcting codes, In: *Proceedings of Workshop on Coding and Cryptography*, INRIA, Paris, pp. 33–41 (1999).

[10] C. Sims, Determining the conjugacy classes of a permutation group, In: *Proc. of the Symposium on Computers in Algebra and Number Theory*, Amer. Math. Soc., New York, pp. 191–195 (1971).

[11] G. Skersys, Calcul du groupe d'automorphismes des codes. Détermination de l'équivalence des codes, *Ph.D. Thesis*, Limoges University (1999).

Klaidas taisančių kodų keitinių grupių skaičiavimas

G. Skersys

Darbe pristatomas algoritmas klaidas taisančių kodų keitinių grupei skaičiuoti ir dviejų kodų ekvivalentumui nustatyti. Šis algoritmas remiasi J. Leono algoritmu ir N. Sendrier pagrindo dalijimo algoritmu. Jis labai efektyvus, kai kodų sankirtos su jų dualiais kodais yra mažos.