UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tecniche efficienti per la soluzione del problema di equivalenza tra codici

Efficient techniques for the solution of the code equivalence problem

Candidato:
Daniele Spalazzi

Relatore:
Prof. Marco Baldi

Correlatore:
Prof. Paolo Santini

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tecniche efficienti per la soluzione del problema di equivalenza tra codici

Efficient techniques for the solution of the code equivalence problem

Candidato:
Daniele Spalazzi

Relatore:
Prof. Marco Baldi

Correlatore:
Prof. Paolo Santini

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ringraziamenti

Questo lavoro e il percorso che ho intrapreso, come tutte le attività che una persona svolge, rappresentano una manifestazione fenomenologica dell'essere. Tuttavia, ciò che ci definisce è in gran parte il risultato delle persone che incontriamo nella vita. Le persone a cui voglio bene hanno avuto un impatto fondamentale nello sviluppo di ciò che sono; per questo motivo desidero far sapere a tutti coloro che amo che questa tesi e questa laurea sono merito loro. Tutti quelli che mi hanno permesso di essere e mi rendono ciò che sono hanno contribuito a questo lavoro, mettendoci qualcosa di sé.

In modo particolare, dedico questo lavoro a due persone che mi hanno amato da sempre e che hanno plasmato e cambiato la mia vita: i miei nonni. Molto di ciò che sono oggi e di ciò che sarò in futuro lo devo a loro e al loro modo di essere.

Fin da piccolo mi sono sentito profondamente amato e coccolato, grazie a chi mi ha insegnato tanto e trasmesso valori preziosi. La parte più importante di questa tesi è proprio questa: un'opportunità per riflettere su ciò che mi ha permesso di portare a termine questo lavoro, ovvero la mia essenza, il mio essere. Condividerò solo una parte di questo percorso personale, perché ci sono molte altre persone a cui voglio esprimere il mio affetto direttamente.

Sono quel bambino che amava aiutare nonno Giorgio nell'orticello, anche se, a dire il vero, la maggior parte delle volte finivo per aumentare il lavoro, ma anche il divertimento. Mi piaceva lavorare il legno con lui, sperimentare nuove ricette divertendoci in cucina e sporcando tutto. Sono anche quel bambino che trascorreva le estati facendo lunghe e impegnative passeggiate con nonno Benedetto, alternandole a splendide corse in bicicletta. Spesso venivo accusato ingiustamente dai passanti di attentare alla vita di nonno, ma in realtà stavo semplicemente facendo la cosa più bella: vivere pienamente, condividendo la mia vitalità con chi amo.

Questi momenti, apparentemente semplici, sono stati per me fonte di ispirazione e hanno plasmato il mio modo di affrontare la vita e le sfide.

Avrebbero voluto tantissimo leggere questa tesi, magari senza comprenderne appieno i concetti tecnici, ma afferrando con amore ciò che conta di più: che questo lavoro è una parte di me. Avrei amato ricevere le correzioni impeccabili e i consigli sulla lingua di nonno Benedetto, che avrebbero reso questa tesi ancora più bella, e vedere l'orgoglio negli occhi di entrambi. Sono sicuro di averli resi fieri e spero di continuare a farlo.

Grazie nonni.

Ancona,

Daniele Spalazzi

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La certezza di poter condividere informazioni tra due nodi senza che queste vengano modificate o lette da persone non autorizzate è stata una caratteristica ricercata ben prima dell'invenzione dei calcolatori. Per un mondo digitalmente interconnesso, come quello che conosciamo, però questa non è solo una caratteristica ricercata ma anche fondamentale.

Nell'ambito delle reti di calcolatori, le informazioni devono poter essere condivise rispettando alcune caratteristiche. Per prima cosa per molte comunicazioni è necessario che le informazioni siano comprensibili solo per le persone autorizzate, si parla in questo caso di confidenzialità dell'informazione. Inoltre può essere necessario certificare che le informazioni provengano da un determinato mittente senza essere state modificate, in questo caso si parla di autenticità e di integrità. Infine, le informazioni devono essere trasmesse senza errori. Questo significa che in presenza di errori, il sistema di trasmissione deve essere in grado di rilevarli e correggerli, in questo caso si parla di affidabilità e resilienza.

Tutte queste proprietà possono essere ottenute tramite l'utilizzo di opportuni algritmi e sistemi crittografici (o crittosistemi). Una classe fondamentale di algoritmi crittografici è quella degli schemi *asimmetrici*, basati su una coppia di chiavi diverse: chiave pubblica e chiave privata, con la chiave pubblica ottenuta tramite una funzione della chiave privata, difficile da invertire. La sicurezza di sistemi di questo tipo è proprio basata sulla complessità di calcolare la funzione inversa. Negli anni vari problemi matematici sono stati utilizzati per realizzare sistemi crittografici asimmetrici. Ad esempio uno dei crittosistemi più famosi, RSA, utilizza il problema della fattorizzazione degli interi, ovvero, il problema di determinare i fattori primi di un numero intero. Un altro problema molto comune è il calcolo del logaritmo discreto utilizzato, ad esempio, in sistemi come Diffie-Hellman e El Gamal.

L'avvento dei computer quantistici rischia di rendere la quasi totalità degli schemi di crittografia asimmetrica insicuri perchè i problemi matematici su cui si basano sono facili da risolvere per computer quantistici. Peter Shor infatti ha mostrato un algoritmo quantistico capace di fattorizzare gli interi in un tempo polinomiale e che, adeguatamente modificato, può essere usato per risolvere anche il problema del logaritmo discreto (anche per curve ellittiche) [1].

L'impellenza di trovare sistemi capaci di sopravvivere in un mondo di computer quantistici ha portato il NIST (National Institute of Standards and Technology) nel 2017 ad indire una competizione internazionale per determinare protocolli di crittografia asimmetrica sia per la confidenzialità sia per la firma digitale [2] [3]. Gli algoritmi vincitori di questa competizione sono stati annunciati nel 2022 [4] [5]. Il problema è che tutti i vincitori tranne uno sono basati su reticoli strutturati: se questo approccio dovesse rivelarsi matematicamente semplice da risolvere, tutti questi schemi risulterebbero attaccabili. Per questo motivo, nel 2022 [6] il NIST ha indetto un altro round della competizione per determinare altri vincitori basati su problemi matematici diversi. Tra gli algoritmi presentati per questa ulteriore fase della competizione è presente LESS (Linear Equivalence Signature Scheme), uno dei due algoritmi presentati da alcuni ricercatori dell'Università Politecnica delle Marche in collaborazione con un team di ricercatori provenienti da università e centri di ricerca nazionali ed internazionali [7] [8] [9] [10] [11].

LESS è un protocollo Zero-Knowledge e schema di firma digitale dove la sicurezza è basata sulla difficoltà di risoluzione del problema dell'*equivalenza dei codici* o *code equivalence*; per questo problema non sembrano esserci degli algoritmi quantistici efficienti.

Il problema dell'equivalenza dei codici richiede di determinare l'equivalenza di due codici. Due codici si dicono equivalenti se hanno delle proprietà comuni tra cui la distanza minima, la distribuzione dei pesi o la capità di correggere degli errori. Codici equivalenti sono legati da una trasformazione lineare che permette di passare da un codice ad un altro. Una trasformazione molto utilizzata è la permutazione delle colonne. Dati due codici risulta però difficile da determinare se questi sono equivalenti e da quale trasformazione sono legati. LESS si basa sulla difficoltà di risolvere questo problema: nello schema, infatti, la chiave pubblica è costituita da una coppia di codici equivalenti, mentre la chiave privata è proprio la trasformazione che lega i due codici. In questo modo, solo chi conosce la trasformazione può provare che i codici sono equivalenti.

Come per tutti i problemi l'algortimo più intuitivo e semplice che si può pensare è quello a forza bruta che cerca tra tutte le possibili permutazioni per determinare l'uguaglianza dei codici. Questo approccio richiede un tempo esponenziale con la dimensione del codice perchè il numero di permutazioni di k elementi è pari a $k!$ che per l'*approssimazione di Stirling* sappiamo avere un andamento più che esponenziale. Algortitmi più efficienti di questo sono stati studiati e implementati ma la loro efficacia dipende fortemente dalla dimensione dell'intersezione del codice con il suo duale, detto hull del codice (per una definizione di hull vedere il Capitolo 2). In particolare la riduzione al problema dell'isomorfismo dei grafi (GIP) è possibile solo quando l'hull ha dimensione nulla; in questi casi il problema è risolvibile in tempo quasi polinomiale, perchè come mostrato da László Babai [12] GIP è risolvibile in tempo quasi polinomiale e il problema dell'equivalenza dei codici è trasformabile nel problema dell'isomorfismo dei grafi in tempo polinomiale.

Esistono però delle istanze del problema difficili da risolvere come ad esempio i codici debolmente auto-duali che hanno dimensione dell'hull massima. In questi casi il tempo risulta esponenziale.

Contributo

Questo lavoro cerca di migliorare gli attacchi per istanze del problema con codici debolmente auto-duali. Il miglioramento viene ottenuto tramite il calcolo dello *square code*: partendo da una coppia di codici auto-duali, tramite lo square code si ottiene una nuova coppia di codici equivalenti che, però, sono differenti rispetto a quelli di partenza ed avranno quindi un hull diverso. In particolare si ha che, con elevata probabilità, l'hull dello square code di codici debolmente auto-duali è più piccolo dell'hull dei codici di partenza. Quando l'hull dello square code è triviale, si può addirittura applicare la trasformazione a GIP per risolvere il problema, ottenendo un risolutore con tempo quasi polinomiale nel worst case. Quando la probabilità che l'hull sia triviale è alta, molte istanze prima considerate difficili sono in realtà risolvibili in tempo quasi polinomiale.

Prima di tutto, sono stati realizzati degli esperimenti per verificare la distribuzione della dimensione dell'hull di codici casuali da confrontare con la distribuzione della dimensione dell'hull dello square code di codici debolmente auto-duali. I risultati ottenuti confermano il comportamento che si attendeva: la dimensione dell'hull viene ridotta significativamente e, con alta probabilità, la dimensione va a zero (l'hull diventa triviale).

Successivamente, sono stati effettuati degli espermenti sull'algoritmo risolutivo decisionale per codici debolmente auto-duali per vedere la percentuale di successo dell'algoritmo. Gli esperimenti mostrano tassi di successo molto alti e che, al crescere della lunghezza del codice, tendono al 100%. Questi risultati confermano che l'approccio studiato in questa tesi costituisce un risolutore efficiente (con tempo quasi-polinomiale nel worst case) ed efficiace per il problema del code equivalence; in particolare, istanze ritenute difficili in precedenza possono essere risolti efficientemente con questo algoritmo.

Come ultimo contributo, infine, è stato approcciato parzialmente il problema di ricerca, effettuando delle prove però solo nel caso di codici già con hull triviale. Anche in questo caso, i risulti ottenuti mostrano percentualli di successo molto alte.

Organizzazione della tesi

La tesi è stata organizzata in modo che nel Capitolo 2 vengono presentati i concetti teorici fondamentali per comprendere al meglio il lavoro svolto sul nuovo tipo di attacco presentato. Nel Capitolo 3 vengono spiegati gli algoritmi decisionali e di ricerca implementati e l'insieme degli esperimenti eseguiti. I risultati degli esperimenti e l'analisi di questi ultimi vengono mostrati nel Capitolo 4. Infine, nel Capitolo 5 vengono riassunte le conclusioni che si possono trarre da questo lavoro, i limiti di questo approccio e i possibili sviluppi futuri della ricerca.

Capitolo 2

Fondamenti teorici

Questo capitolo introduce alcuni concetti fondamentali della teoria dei codici utili per analizzare il problema dell'equivalenza di codici. Vengono anche spiegate le notazioni utilizzate nel resto del lavoro nella Tabella 2.1.

2.1 Campo finito

La struttura algebrica utilizzata per definire dei codici correttori è un *campo finito* (vedi Definizione 2.1.3) perchè è necessario poter effettuare addizioni, sottrazioni, moltiplicazioni e divisioni all'interno del campo.

Definizione 2.1.1. Un *campo* [13] F è un insieme di elementi con due operazioni: $+$ detta addizione e $\cdot$ detta moltiplicazione, che soddisfano le seguenti proprietà:

(i) Chiusura: F è chiuso rispetto a $+$ e $\cdot$, cioè $a + b$ e $a \cdot b \in F$ ogni volta che $a, b \in F$.

Per ogni a, b e c appartenenti a F, valgono le seguenti proprietà:

(ii) Proprietà commutativa: $a + b = b + a$, $a \cdot b = b \cdot a$.

(iii) Proprietà associativa: $(a + b) + c = a + (b + c)$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(iv) Proprietà distributiva: $a \cdot (b + c) = a \cdot b + a \cdot c$.

Inoltre, gli elementi identità 0 e 1 devono esistere in F soddisfacendo:

(v) $a + 0 = a$ per ogni a appartenente ad F.

(vi) $a \cdot 1 = a$ per ogni a appartenente ad F.

(vii) Per ogni a appartenente ad F esiste un elemento opposto $(-a)$ appartenente ad F tale che $a + (-a) = 0$.

a	Uno scalare è rappresentato con una lettera minuscola
$\mathbf{a}$	Un vettore è rappresentato con una lettera minuscola in grassetto
$\mathbf{A}$	Una matrice è rappresentata con una lettera maiuscola in grassetto
A	Un insieme è rappresentato con una lettera maiuscola
q	Rappresenta una potenza di un numero primo, cioè $q = p^m$ dove p è un numero primo e $m \in \mathbb{N}$
$\mathbb{F}_q$	Campo finito o di Galois di ordine q
$\mathbb{F}_q^n$	Spazio vettoriale di dimensione n e ordine q
$\langle \mathbf{u}, \mathbf{v} \rangle$	Prodotto scalare tra due vettori
$\langle \mathbf{u}, \mathbf{v} \rangle = 0$	$\mathbf{u}$ e $\mathbf{v}$ sono ortogonali

Tabella 2.1: Notazione usata nel presente lavoro

(viii) Per ogni $a \neq 0$ appartenente ad F esiste un inverso a^{-1} appartenente ad F tale che $a \cdot a^{-1} = 1$.

Definizione 2.1.2. Un insieme di elementi dotato delle operazioni $+$ e $\cdot$ che soddisfa le proprietà dalla (i) alla (vii) ma non necessariamente la (viii) è detto *anello* [13].

Definizione 2.1.3. Un *campo finito* [13] è un campo con un numero finito di elementi, questo numero è detto *ordine* del campo.

Gli studi di Evariste Galois sono molto importanti nell'ambito dei campi finiti, come i risultati di seguito riportati [13].

Teorema 2.1.1. *Esiste un campo finito di ordine q se e solo se q è una potenza di un numero primo, cioè $q = p^m$ dove p è un numero primo e $m \in \mathbb{N}$. Inoltre se q è la potenza di un numero primo, allora esiste un solo campo di quell'ordine a meno di una rietichettatura.*

Comunemente ci si riferisce ad un campo finito di ordine q come ad un campo di Galois di ordine q. Notazioni comuni sono $GF(q)$ e $\mathbb{F}_q$; in questo lavoro, si utilizzerà la seconda scelta.

Se q è un numero primo allora il campo può essere definito in modo molto semplice con una chiusura modulo q, dove le operazioni aritmetiche vengono effettuate semplicemente riducendone modulo q il risultato.

Definizione 2.1.4. Sia m un numero intero positivo. Due interi a e b vengono detti congruenti modulo q [13], simbolicamente

$$a \equiv b \pmod{q}$$

se $a - b$ è divisibile per q, cioè se $a = kq + b$ per qualche intero k.

É possibile dimostrare che la chiusura modulo un numero primo produce un campo finito, mentre se il numero non è primo produce un anello. Gli elementi sono rappresentati come $\{0, 1, \ldots, q-1\}$.

Se q non è un numero primo ma una potenza di un numero primo allora risulta un po' più complesso definire il campo di Galois. Se si vuole costruire un campo finito di ordine q^m, con q primo, i suoi elementi si possono rappresentare come polinomi con coefficienti in $\mathbb{F}_q$ e grado massimo $m-1$. Per prima cosa va però scelto un polinomio irriducibile $g(x)$ di grado m con coefficienti sempre in $\mathbb{F}_q$. Le operazioni tra i polinomi all'interno del campo saranno definite modulo $g(x)$.

2.2 Codici lineari

Un codice permette di rappresentare un insieme di informazioni aggiungendo ridondanza alle sequenze originali e mappandone biunivocamente ciascuna in una sequenza più lunga, detta parola di codice, allo scopo di consentire il rilevamento o la correzione degli errori nelle telecomunicazioni.

Formalmente, un codice può essere definito come segue [14].

Definizione 2.2.1. Detto *alfabeto* un insieme finito di elementi chiamati *simboli*, allora una *parola* è una sequenza finita di simboli dell'alfabeto e un *codice* è un insieme di parole definite sull'alfabeto considerato.

Verranno considerati codici che hanno alfabeto $\mathbb{F}_q$.

Definizione 2.2.2. $C \subset \mathbb{F}_q^n$ è un *codice lineare* [13] se:

(1) $\mathbf{u} + \mathbf{v} \in C$, $\forall$ $\mathbf{u}$ e $\mathbf{v} \in C$

(2) $a\mathbf{u} \in C$, $\forall$ $\mathbf{u} \in C$ e $a \in \mathbb{F}_q$

Quindi un codice lineare $C \in \mathbb{F}_q^n$ di dimensione k e lunghezza n è un sottospazio vettoriale di $\mathbb{F}_q^n$ di dimensione k. Un vettore $\mathbf{c} \in C$ è una *parola* del codice.

Il codice può essere definito tramite una *matrice generatrice* $\mathbf{G}$ che è una matrice a rango pieno in $\mathbb{F}_q^{k\times n}$ tale per cui $C = \{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_q^k\}$. Quindi le righe di $\mathbf{G}$ rappresentano una base del sottospazio vettoriale associato a C.
Le parole del codice sono delle combinazioni lineari delle righe di $\mathbf{G}$, formando queste una base per il codice. Se il codice ha dimensione k allora lo spazio associato ha dimensione k e il codice contiene q^k parole.
La matrice generatrice in *forma sistematica* è del tipo $(\mathbf{I_k}, \mathbf{V})$ dove $\mathbf{V} \in \mathbb{F}_q^{k\times(n-k)}$.

Un altro modo per definire un codice lineare è tramite una *matrice di parità* che è una matrice a rango pieno $\in F_q^{(n-k)\times n}$ tale per cui $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{xH}^T = \mathbf{0}\}$.
Se la matrice generatrice è in forma sistematica è possibile trovare facilmente la matrice di parità nel seguente modo:

$$\mathbf{G} = (\mathbf{I}_k\mathbf{V}) \iff \mathbf{H} = \left(-\mathbf{V}^T, \mathbf{I}_{n-k}\right) \tag{2.1}$$

2.3 Duale di un codice

Definizione 2.3.1. Dato un codice C di dimensione k e lunghezza n il *codice duale* di C rappresentato da $C^\perp$ è definito come l'insieme di vettori in $\mathbb{F}_q^n$ che sono ortogonali a tutte le parole di C [13], ovvero:

$$C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \langle \mathbf{u}, \mathbf{v} \rangle = 0 \ \forall \mathbf{u} \in C\} \tag{2.2}$$

É possibile dimostrare che il duale di un codice è anch'esso un codice. Inoltre possiamo notare che la matrice generatrice del duale è la matrice di parità del codice.

2.4 Hull di un codice

La dimensione dell'hull di un codice è legata alla facilità di risoluzione del problema di equivalenza, per questo motivo è importante in questo lavoro.

Definizione 2.4.1. L'*Hull* di un codice lineare C è definito come l'intersezione tra il codice e il suo duale [15], ovvero:

$$\mathcal{H}(C) = C \cap C^\perp \tag{2.3}$$

Quando $C \cap C^\perp = \{\mathbf{0}_k\}$ allora l'hull ha dimensione nulla e diciamo che l'hull è triviale. Invece se $C = C^\perp$ allora il codice viene detto auto-duale e l'hull ha dimensione k con $k = \frac{n}{2}$ perchè C e $C^\perp$ hanno la stessa dimensione e quindi $k = n - k$. Infine se $C \subset C^\perp$ allora il codice viene detto *debolmente auto-duale* e l'hull ha dimensione massima pari a k.

Nota 2.4.1. Ogni vettore appartenente all'hull banalmente appartiene sia al codice che al suo duale, ciò implica che è ortogonale ad ogni altro vettore di C e di $C^\perp$, inoltre siccome sarà una parola sia di C che di $C^\perp$ allora la dimensione dell'hull sarà nell'intervallo $[1; \min(k, n-k)]$.

Proposizione 2.4.1. L'hull è un sottospazio lineare di $\mathbb{F}_q^n$.

Dimostrazione. Siano $\mathbf{c}, \mathbf{c}' \in \mathcal{H}(C)$ allora $\exists \mathbf{u}, \mathbf{u}' \in \mathbb{F}_q^k$ e $\tilde{\mathbf{u}}, \tilde{\mathbf{u}}' \in \mathbb{F}_q^{n-k} : \mathbf{c} = \mathbf{u}\mathbf{G}, \mathbf{c}' = \mathbf{u}'\mathbf{G}, \mathbf{c} = \tilde{\mathbf{u}}\mathbf{H}, \mathbf{c}' = \tilde{\mathbf{u}}'\mathbf{H}$.
Consideriamo una combinazione lineare di $\mathbf{c}$ e di $\mathbf{c}'$:

$$\mathbf{c}'' = \alpha\mathbf{c} + \beta\mathbf{c}'$$

con $\alpha, \beta \in \mathbb{F}_q$. Allora:

$$\mathbf{c}'' = \alpha(\mathbf{u}\mathbf{G}) + \beta(\mathbf{u}'\mathbf{G}) = (\alpha\mathbf{u} + \beta\mathbf{u}')\mathbf{G} = \mathbf{u}''\mathbf{G}, \tag{2.4}$$

$$\mathbf{c}'' = \alpha(\tilde{\mathbf{u}}\mathbf{H}) + \beta(\tilde{\mathbf{u}}'\mathbf{H}) = (\alpha\tilde{\mathbf{u}} + \beta\tilde{\mathbf{u}}')\mathbf{H} = \tilde{\mathbf{u}}''\mathbf{H}. \tag{2.5}$$

Per l'Equazione 2.4 $\mathbf{c}'' \in C$ e per l'Equazione 2.5 $\mathbf{c}'' \in C^{\perp}$ e quindi $\mathbf{c}'' \in \mathcal{H}(C)$. $\square$

2.5 Cambio di base

Indichiamo con GL_k il gruppo lineare delle matrici non singolari cioè invertibili appartenenti a $\mathbb{F}_q^{k \times k}$. Le matrici appartenenti a GL_k vengono chiamate *matrici di cambio di base*. Considerando che le matrici generatrici associate ad un codice sono formate dai vettori che formano una base del sottospazio vettoriale, è possibile ottenere un'altra matrice generatrice cambiando base. Quindi dato un codice C con una matrice generatrice $\mathbf{G}$ allora $\forall \mathbf{S} \in GL_k$, la matrice $\mathbf{SG}$ è generatrice dello stesso codice C.

É possibile calcolare la matrice generatrice in forma sistematica di un codice C se si conosce una qualsiasi matrice generatrice $\mathbf{G}$ nella forma $\mathbf{G} = (\mathbf{A}, \mathbf{B})$ con $\mathbf{A} \in \mathbb{F}_q^{k \times k}$ e $\mathbf{B} \in \mathbb{F}_q^{k \times (n-k)}$, considerando $\mathbf{V} = \mathbf{A}^{-1}\mathbf{B}$. Infatti applicando il cambio di base $\mathbf{S} = \mathbf{A}^{-1}$ si ottiene $\mathbf{SG} = (\mathbf{SA}, \mathbf{SB}) = (\mathbf{A}^{-1}\mathbf{A}, \mathbf{SB}) = (\mathbf{I}_k, \mathbf{V})$

2.6 Permutazione

Una *permutazione* [16] può essere rappresentata con la seguente notazione:

$$\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix} \tag{2.6}$$

Si può anche rappresentare con una notazione abbreviata su una singola linea $\pi = (i_1, i_2, \dots, i_n)$ tale per cui $\pi(j) = i_j$.
Dato un vettore $\mathbf{v}$ di dimensione n la permutazione π sposta l'elemento in posizione j nella posizione i_j, cioè descriviamo l'azione della permutazione sul vettore $\mathbf{v}$ come:

$$\pi(\mathbf{v}) = \left(a_{\pi^{-1}(1)}, a_{\pi^{-1}(2)}, \dots, a_{\pi^{-1}(n)}\right).$$

La permutazione può essere applicata anche ad una matrice con n colonne $\mathbf{A}$; $\pi(\mathbf{A})$ scambia le colonne della matrice.

Inoltre, è sempre possibile associare ad una permutazione una matrice di permutazione. Una *matrice di permutazione* è una matrice quadrata che ha esattamente uno ed un solo 1 in ogni riga e in ogni colonna con tutti gli altri elementi pari a 0.
Data una matrice di permutazione $\mathbf{P} \in \mathbb{F}_q^{n \times n}$ e una matrice $\mathbf{A} \in \mathbb{F}_q^{n \times n}$ allora pre-moltiplicando $\mathbf{P}$ si ottiene la matrice $\mathbf{PA}$ che ha le righe permutate mentre post-moltiplicando $\mathbf{P}$ si ottiene la matrice $\mathbf{AP}$ che ha le colonne permutate.

É possibile dimostrare che ogni matrice di permutazione è ortogonale, cioè $\mathbf{P}^{-1} = \mathbf{P}^T$.

$$\begin{array}{ccc}
\pi : \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix} & \leftrightarrow & \mathbf{P} : \left[p_{i,j}\right] \text{ con } p_{i,j} = 1 \text{ se } j = \pi(i) \\
\updownarrow & & \updownarrow \\
\mathbf{P}^T : \left[p^T_{i,j}\right] \text{ con } p^T_{i,j} = p_{j,i} & \leftrightarrow & \pi^{-1} : \begin{pmatrix} 1 & 2 & \dots & n \\ \pi^{-1}(1) & \pi^{-1}(2) & \dots & \pi^{-1}(n) \end{pmatrix}
\end{array} \tag{2.7}$$

Esempio 2.6.1.

$$\begin{array}{ccc}
\pi : \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} & \leftrightarrow & \mathbf{P} : \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \\
\updownarrow & & \updownarrow \\
\mathbf{P}^T : \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} & \leftrightarrow & \pi^{-1} : \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}
\end{array}$$

Moltiplicando a destra o a sinistra una matrice $\mathbf{A}$ per $\mathbf{P}$ o $\mathbf{P}^T$ si scambiano le righe o le colonne secondo la permutazione π o π^{-1}.

1. Pre moltiplicando $\mathbf{A}$ per $\mathbf{P}^T$ si ottiene:
   $\left(\mathbf{P}^T\mathbf{A}\right)_{i,j} = \sum_{i=1}^{n} p_{k,i} a_{k,j}$ dove $p_{k,i} = 1$ quando $i = \pi(k)$ cioè quando $k = \pi^{-1}(i)$
   quindi la somma si riduce al termine $a_{\pi^{-1}(i),j}$
   Quindi gli indici delle righe sono permutati di π.

2. Post moltiplicando $\mathbf{A}$ per $\mathbf{P}$ con ragionamenti simili al caso 1 gli indici delle colonne sono permutati di π.

3. Pre moltiplicando $\mathbf{A}$ per $\mathbf{P}$ con ragionamenti similial caso 1 gli indici delle righe sono permutati di π^{-1}.

4. Post moltiplicando $\mathbf{A}$ per $\mathbf{P}^T$ con ragionamenti simili al caso 1 gli indici delle colonne sono permutati di π^{-1}.

Esempio 2.6.2. Riprendendo le permutazioni definite nell'Esempio 2.6.1.
Dati $\mathbf{a} = (a_1, a_2, a_3)$ e $\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$ allora:

$$\pi(\mathbf{a}) = (a_2, a_3, a_1)$$
$$\pi(\mathbf{A}) = [\mathbf{a_2}, \mathbf{a_3}, \mathbf{a_1}] = \begin{bmatrix} a_{1,2} & a_{1,3} & a_{1,1} \\ a_{2,2} & a_{2,3} & a_{2,1} \end{bmatrix}$$

2.7 Equivalenza tra codici lineari

Indichiamo con S_n l'insieme delle permutazioni di n elementi detto *gruppo simmetrico*, con $\mathbb{F}_q^*$ il *gruppo moltiplicativo* del campo finito $\mathbb{F}_q$, cioè l'insieme degli elementi non nulli rispetto alla moltiplicazione nel campo e infine con $\mathbb{F}_q^{*n}$ l'insieme dei vettori di lunghezza n con elementi in $\mathbb{F}_q^*$.
Inoltre un *automorfismo dei campi* è una mappa biettiva $\sigma : F \to F$ che rappresenta un isomorfismo che preserva le caratteristiche del campo rispetto alle operazioni definite.
Una *isometria lineare* è una trasformazione lineare che preserva la distanza rispetto a una determinata metrica. Siamo interessati a isometrie lineari del tipo $\tau = (\mathbf{v}, \pi) \in \mathbb{F}_q^{*n} \ltimes S_n$, dove $\mathbf{v} \in \mathbb{F}_q^{*n}$ e $\pi \in S_n$ [7]. L'azione sul vettore $\mathbf{a}$ sarà descritta analogamente alle permutazioni come:

$$\tau(\mathbf{x}) = \left(v_1 x_{\pi(1)}, \ldots, v_n x_{\pi(n)}\right).$$

Questa trasformazione può essere rappresentata in forma matriciale come il prodotto $\mathbf{Q} = \mathbf{DP}$, dove $\mathbf{P}$ rappresenta una matrice di permutazione e $\mathbf{D}$ una matrice diagonale quadrata con elementi in $\mathbb{F}_q^*$. Denotiamo con M_n l'insieme delle matrici $\mathbf{Q}$, spesso definite *matrici monomiali.*

Considereremo codici per i quali la metrica definita è la distanza di Hamming.

Definizione 2.7.1. La *distanza di Hamming* [17] tra due vettori $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ e $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, rappresentata come $d_H(\mathbf{x}, \mathbf{y})$, è il numero di posizioni dove i simboli x_i e y_i differiscono.

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i) \tag{2.8}$$

dove

$$\delta(x_i, y_i) = \begin{cases} 0 & x_i = y_i \\ 1 & x_i \neq y_i \end{cases} \tag{2.9}$$

Siccome la trasformazione τ sposta e scala semplicemente gli elementi allora possiamo confermare che essa è isometrica. Notiamo che anche le permutazioni sono isometriche, perché sono un caso particolare di questa trasformazione dove però il fattore di scala è pari a 1, cioè $\mathbf{v} = (1, \ldots, 1)$.

Definizione 2.7.2. Diciamo che due codici lineari C e C' sono equivalenti [7] e scriviamo $C \sim C'$ se esistono un automorfismo $\alpha \in \mathbf{Aut}(\mathbb{F}_q)$ e un'isometria lineare $\tau = (\mathbf{v}; \pi) \in \mathbb{F}_q^{*n} \ltimes S_n$ che mappa C in C', cioè tale per cui $C' = \tau(\alpha(C)) = \{\mathbf{y} \in \mathbb{F}_q^n : \mathbf{y} = \tau(\alpha(\mathbf{x})), \mathbf{x} \in C\}$

Due codici equivalenti avranno delle proprietà in comune come la distanza minima o la distribuzione dei pesi.

Se i codici sono rappresentati dalle matrici generatrici $\mathbf{G}$ e $\mathbf{G}'$ allora:

$$C \sim C' \Leftrightarrow \exists\left(\mathbf{S},(\alpha,\mathbf{Q})\right) \in GL_k \ltimes \left(Aut\left(\mathbb{F}_q\right) \times M_n\right) : \mathbf{G}' = \mathbf{S}\alpha\left(\mathbf{GQ}\right)$$

L'equivalenza con questa notazione è conosciuta come *equivalenza semi lineare.*
Inoltre se l'automorfismo è triviale, cioè è l'identità allora si parla di *equivalenza lineare.*
Infine se la matrice monomiale è una permutazione cioè se $\mathbf{D} = I_n$ allora si parla di *permutation equivalence.*
Il lavoro qui svolto tratta solo della permutation equivalence.

2.7.1 Problema di equivalenza

Il problema di equivalenza può essere espresso in modo differente a seconda che si parli di equivalenza semi lineare, lineare o permutation equivalence. [7].

Problema 2.7.1 (Problema semi lineare)**.** Siano $\mathbf{G}, \mathbf{G}'$ due matrici generatrici per i codici equivalenti C e C'. Trovare un automorfismo $\alpha \in Aut\left(\mathbb{F}_q\right)$ e due matrici $\mathbf{S} \in GL_k$ e $\mathbf{Q} \in M_n$ tale per cui $\mathbf{G}' = \mathbf{S}\alpha\left(\mathbf{GQ}\right)$.

Problema 2.7.2 (Problema lineare)**.** Siano $\mathbf{G}, \mathbf{G}'$ due matrici generatrici per i codici equivalenti linearmente C e C'. Trovare due matrici $\mathbf{S} \in GL_k$ e $\mathbf{Q} \in M_n$ tale per cui $\mathbf{G}' = \mathbf{SGQ}$.

Problema 2.7.3 (Permutation equivalence problem (PEP))**.** Siano $\mathbf{G}, \mathbf{G}'$ due matrici generatrici per i codici equivalenti per una permutazione C e C'. Trovare due matrici $\mathbf{S} \in GL_k$ e $\mathbf{P} \in S_n$ tale per cui $\mathbf{G}' = \mathbf{SGP}$.

Questo lavoro si concentra solamente sulla risoluzione del problema di permutazione o PEP.

Molti problemi matematici si possono presentare in due forme: una decisionale e una di ricerca. Nella forma decisionale la soluzione risiede semplicemente nel rispondere si o no, mentre nella forma di ricerca è necessario determinare una soluzione vera e propria. Anche nel nostro caso il problema si presenta sotto queste due forme. Nel *problema decisionale* dati due codici il compito è determinare se essi sono equivalenti oppure no. In altre parole si deve rispondere alla domanda: i due codici sono equivalenti? Mentre nel *problema di ricerca* una volta determinato che i codici sono equivalenti, allora è necessario trovare la matrice di permutazione $\mathbf{P}$ e la matrice di cambio di base $\mathbf{S}$ associate all'uguaglianza dei codici. Praticamente in questo caso oltre a stabilire se i codici sono equivalenti si deve fornire una rappresentazione concreta dell'equivalenza attraverso le matrici $\mathbf{P}$ e $\mathbf{S}$.

2.8 Isomorfismo dei grafi

Il *problema dell'isomorfismo dei grafi* è un problema classico che è stato studiato fin dagli albori della teoria dell'informazione. Siamo interessati a questo problema perchè è collegato al problema dell'equivalenza dei codici e sotto certe condizioni è possibile trasformare un problema nell'altro.

Un grafo inanzitutto è definito da un insieme di nodi o vertici e da un insieme di archi o spigoli.

Definizione 2.8.1. Un *grafo* [18] è rappresentato come una coppia $G = (V, E)$ dove V rappresenta l'insieme di vertici e $E \subset V \times V$ è l'insieme di archi.

L'isomorfismo dei grafi è condizionato al mantenimento della stessa struttura dati a delle stesse connessioni.

Definizione 2.8.2. Due grafi $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ si dicono *isomorfi* [18] se esiste una biezione tra l'insieme dei vertici V_1 e V_2

$$f : V_1 \to V_2$$

tale per cui per ogni coppia di vertici $u, v \in V_1$ sono connessi in G_1 se e solo se $f(u), f(v)$ sono connessi in G_2, cioè $(u, v) \in E_1$ se e solo se $(f(u), f(v)) \in E_2$.

Nel caso del problema del code equivalence i grafi importanti sono i grafi pesati non orientati dove l'isomorfismo è condizionato anche al peso del grafo.

Definizione 2.8.3. Un *grafo non orientato* è un grafo $G = (V, E)$ dove $(u, v) \in E$ se e solo se $(v, u) \in E$.

La trasformazione del problema è possibile solo introducendo un altro concetto cioè quello di *matrice di adiacenza* di un grafo, che è la rappresentazione matriciale del grafo.

Definizione 2.8.4. Dato un grafo pesato $G = (V, E)$ con $V = \{v_1, \dots v_n\}$ allora la *matrice di adiacenza* $\mathbf{A}$ è una matrice quadrata $n \times n$ tale per cui $a_{i,j} = w(v_i, v_j)$ se e solo se $(v_i, v_j) \in E$ con peso $w(v_i, v_j)$ e $a_{i,j} = 0$ altrimenti.

Nota 2.8.1. La diagonale della matrice di adiacenza avrà ovviamente tutti 1, cioè $a_{i,j} = 1 \forall i, j \in 1, \dots n$.

Nota 2.8.2. Nel caso di un grafo non orientato allora la matrice di adiacenza associata sarà simmetrica, cioè $a_{i,j} = a_{j,i}$ $\forall i, j \in 1, \dots n$ perchè in un grafo non orientato $(u, v) \in E$ se e solo se $(v, u) \in E$.

2.8.1 Problema dell'isomorfismo dei grafi

Il problema dell'isomorfismo dei grafi può essere presentato in modo semplice tramite le matrici di adiacenza. Infatti dati due grafi G e G' con matrici di adiacenza associate $\mathbf{A}$ e $\mathbf{A'}$ allora i grafi sono isomorfi se e solo se esiste una permutazione P tale per cui $\mathbf{A'} = \mathbf{P}^T\mathbf{AP}$ [19].

Problema 2.8.1. Dati due grafi G e G' con matrici di adiacenza associate $\mathbf{A}$ e $\mathbf{A'}$, determinare se esiste una matrice di permutazione $\mathbf{P}$ tale per cui $\mathbf{A'} = \mathbf{P}^T\mathbf{AP}$.

Il problema è spesso abbreviato con GIP che sta per *graph isomorphism problem.*

2.8.2 Trasformazione PEP a GIP

Una prima cosa che possiamo notare è che le formulazioni dei due problemi sono simili, solo che nel caso di PEP è necessario determinare anche un cambio di base, mentre nel caso di GIP abbiamo la trasposta della permutazione. La cosa che ci interessa di più è che per alcune istanze di PEP è possibile trasformare il problema in un'istanza di GIP.

Inanzitutto definiamo una funzione $\mathcal{G} : \mathbb{F}_q^{k\times n} \to \mathbb{F}_q^{n\times n}$ che trasforma una matrice generatrice $\mathbf{G}$ in una matrice di adiacenza [20].

$$\mathcal{G}(\mathbf{G}) = \mathbf{G}^T \left(\mathbf{G}\mathbf{G}^T\right)^{-1} \mathbf{G} = \mathbf{A} \tag{2.10}$$

Nota 2.8.3. E' banale osservare che la matrice quadrata $\mathbf{A}$ che si ottiene è una matrice simmetrica.

Il dominio di questa funzione non è tutto $\mathbb{F}_q^{k\times n}$, ma è limitato alle matrici $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ tali per cui $\mathbf{G}\mathbf{G}^T$ è invertibile: questo è vero quando l'hull è triviale.

Proposizione 2.8.1. Dato un codice lineare C con matrice generatrice $\mathbf{G}$, $\mathbf{G}\mathbf{G}^T$ è invertibile se e soltanto se l'hull del codice è triviale [21].

Dimostrazione. Dimostriamo inanzitutto che se l'hull non è triviale, cioè se $C \cap C^\perp$ allora $\mathbf{G}\mathbf{G}^T$ non è invertibile, che corrisponde al dimostrare che se $\mathbf{G}\mathbf{G}^T$ è invertibile allora l'hull è triviale.
Sia $\mathbf{G} = \begin{bmatrix}\mathbf{I}_k & | & \mathbf{V}\end{bmatrix}$ e $\mathbf{H} = \begin{bmatrix}-\mathbf{V}^T & | & \mathbf{I}_{n-k}\end{bmatrix}$, allora $\forall \mathbf{u} \in \mathbb{F}_q^k$ e $\forall \mathbf{u'} \in \mathbb{F}_q^{n-k}$

$$\mathbf{u}\begin{bmatrix}\mathbf{I}_k & | & \mathbf{V}\end{bmatrix} = \mathbf{u'}\begin{bmatrix}-\mathbf{V}^T & | & \mathbf{I}_{n-k}\end{bmatrix}$$
$$\begin{bmatrix}\mathbf{u}\mathbf{I}_k & | & \mathbf{u}\mathbf{V}\end{bmatrix} = \begin{bmatrix}-\mathbf{u'}\mathbf{V}^T & | & \mathbf{u'}\mathbf{I}_{n-k}\end{bmatrix}$$

$$\begin{cases}\mathbf{u} & = -\mathbf{u'}\mathbf{V}^T \\ \mathbf{u}\mathbf{V} & = \mathbf{u'}\end{cases}$$

Quindi $-\mathbf{u'}\mathbf{V}\mathbf{V}^T = \mathbf{u'}$ ciò implica che $\mathbf{V^T} = -\mathbf{V}^{-1}$ Allora

$$\mathbf{G}\mathbf{G}^T = \begin{bmatrix}\mathbf{I_k} & | & \mathbf{V}\end{bmatrix}\begin{bmatrix}\mathbf{I_k} \\ -\mathbf{V}^T\end{bmatrix} = \mathbf{I}_k + \mathbf{P}\mathbf{P}^T = \mathbf{I}_k - \mathbf{I}_k = \mathbf{0}_k$$

Quindi $\mathbf{G}\mathbf{G}^T$ non è invertibile.
Adesso dimostriamo che se l'hull è triviale allora $\mathbf{G}\mathbf{G}^T$ è invertibile, dimostrando che se $\mathbf{G}\mathbf{G}^T$ non è invertibile allora l'hull non è triviale. Se $\mathbf{G}\mathbf{G}^T$ non è invertibile allora esiste un vettore non nullo $\mathbf{u} \in \mathbb{F}_q^k$ tale per cui $\mathbf{u}\mathbf{G}\mathbf{G}^T = \mathbf{0}_k$ e $\mathbf{u}\mathbf{G}$ sarà non nullo in C. Dato un vettore arbitrario in C questo può essere rappresentato come $\mathbf{v} = \mathbf{u'}\mathbf{G}$ per qualche $\mathbf{u'} \in \mathbb{F}_q^k$. Quindi:

$$(\mathbf{u}\mathbf{G})\mathbf{v}^T = (\mathbf{u}\mathbf{G})(\mathbf{u'}\mathbf{G})^T = \mathbf{u}\mathbf{G}\mathbf{G}^T\mathbf{u'}^T = \mathbf{0}_k\mathbf{u'}^T = \mathbf{0}_k$$

Quindi $\mathbf{uG}$ è un vettore anche in $C^{\perp}$ e quindi $C \cap C^{\perp} \neq \{\mathbf{0}_k\}$. □

Considerando ora un'istanza di PEP, cioè dati due codici C e C' equivalenti con $\mathbf{G}' = \mathbf{P}^T\mathbf{GP}$, ed applicando la Funzione 2.10 a sinistra dell'equazione si ottiene:

$$\begin{aligned} \mathcal{G}(\mathbf{G}') &= \mathbf{G}'^T \left(\mathbf{G}'\mathbf{G}'^T\right)^{-1} \mathbf{G}' \\ &= \mathbf{A}' \end{aligned} \tag{2.11}$$

Applicandolo al secondo termine si ottiene:

$$\begin{aligned} \mathcal{G}(\mathbf{SGP}) &= \mathbf{SGP}^T \left(\mathbf{SGP}(\mathbf{SGP}^T)\right)^{-1} \mathbf{SGP} \\ &= \mathbf{P}^T\mathbf{G}^T\mathbf{S}^T \left(\mathbf{SGPP}^T\mathbf{G}^T\mathbf{S}^T\right)^{-1} \mathbf{SGP} \\ &= \mathbf{P}^T\mathbf{G}^T\mathbf{S}^T(\mathbf{S}^T)^{-1} \left(\mathbf{GG}^T\right)^{-1} \mathbf{S}^{-1}\mathbf{SGP} \\ &= \mathbf{P}^T\mathbf{G}^T \left(\mathbf{GG}^T\right)^{-1} \mathbf{GP} \\ &= \mathbf{P}^T\mathbf{AP} \end{aligned} \tag{2.12}$$

Combinando l'Equazione 2.11 e 2.12 si ottiene la nuova equazione:

$$\mathbf{A}' = \mathbf{P}^T\mathbf{AP} \tag{2.13}$$

É possibile concludere che due codici con hull triviale sono equivalenti se e soltanto se i corrispondenti grafi sono isomorfi.
Ogni qual volta che l'hull dei codici è triviale, è possibile trasformare il problema di equivalenza nel problema dell'isomorfismo dei grafi e risolvere quindi quest'ultimo con un risolutore.
Si nota inoltre che una volta determinato $\mathbf{P}$ con il solutore per GIP diventa banale determinare anche il cambio di base $\mathbf{S}$.

2.9 Complessità computazionale

La teoria della complessità computazionale permette di studiare e stimare le quantità minime di risorse richieste per la risoluzione di un problema, quindi confronta le risorse richieste dai vari algoritmi risolutivi di un problema.

Gli schemi della crittografia asimmetrica sono condizionati a dei problemi difficili, cioè non risolvibili in tempi polinomiali da una macchina deterministica.

Lo studio delle risorse temporali richieste viene approssimato con la notazione del *O grande* che permette di classificare gli algoritmi in delle classi di complessità confrontabili. La stima dei tempi è basata sul calcolo del numero di operazioni elementari che vanno eseguite nel caso peggiore, cioè nell'istanza peggiore per un determinato algoritmo. Le prime classi di complessità definite considerano problemi decisionali cioè per i quali una soluzione corrisponde ad indicare se l'istanza rispetta una proprietà.

Definizione 2.9.1 (Classe $\mathcal{P}$)**.** La classe $\mathcal{P}$ è la classe dei problemi decisionali risolvibili in tempo polinomiale.

Un problema decisionale viene considerato facilmente risolvibile se si trova nella classe di complessità $\mathcal{P}$.

Definizione 2.9.2 (Classe $\mathcal{NP}$)**.** La classe $\mathcal{NP}$ è la classe dei problemi verificabili in tempo polinomiale.

Un problema viene detto verificabile se per ogni istanza x di dimensione n del problema esiste un certificato y di lunghezza polinomiale n ed esiste una funzione di verifica $A(x, y)$ polinomiale.

Nota 2.9.1. Ogni problema di classe $\mathcal{P}$ è anche di classe $\mathcal{NP}$, perchè banalmente ogni istanza può essere verificata in tempo polinomiale applicando l'algoritmo risolutivo per verificare l'esattezza della soluzione iniziale.

Quindi possiamo dire con certezza che $\mathcal{P} \subset \mathcal{NP}$. Esistono dei problemi che sono però verificabili in tempo polinomiale ma non risolvibili in tempo polinomiale, questi sono facili da verificare ma difficili da risolvere. L'esistenza di questi problemi porta alla congettura $\mathcal{P} \neq \mathcal{NP}$, definita come uno dei problemi del millennio.

Inoltre un problema viene detto in classe $\mathcal{NP}$ completo se appartiene ad $\mathcal{NP}$ ma è almeno difficile quanto ognuno dei problemi in $\mathcal{NP}$. L'insieme dei problemi $\mathcal{NP}$ completi è riconducibile l'uno all'altro in tempo polinomiale e sono quindi con complessità equivalente. Se si dimostrasse che un problema in $\mathcal{NP}$ completo è risolvibile in tempo polinomiale allora la congettura sarebbe falsa, mentre se si riuscisse a dimostrare che un problema $\mathcal{NP}$ completo non è risolvibile in tempo polinomiale si verificherebbe la congettura.

Il problema dell'equivalenza dei codici è un problema notoriamente difficile ma, come dimostrato in [22], se si mostra che PEP appartiene ad $\mathcal{NP}$ completo allora $\mathcal{P} = \mathcal{NP}$. Siccome si pensa che la congettura $\mathcal{P} \neq \mathcal{NP}$ sia corretta allora si pone PEP in $\mathcal{NP}$ ma al di fuori di $\mathcal{P}$ o di $\mathcal{NP}$ completo. Questo problema è stato studiato notevolmente e si conoscono numerose istanze che permettono una risoluzione polinomiale, come ad esempio il caso di codici con hull triviale, esistono però allo stesso modo delle istanza che sono difficili da risolvere e che richiedono un tempo polinomiale, come ad esempio il caso dei codici debolmente auto-duali.

Allo stesso modo il problema dell'isomorfismo dei grafi si ritiene sia in $\mathcal{NP}$ ma non completo, perché altrimenti la congettura sarebbe violata. Per questo problema però è stato trovato un algoritmo in tempo quasi polinomiale [12], cioè con un limite superiore pari a $2^{O((\log n)^c)}$. Quindi per le istanze per le quali PEP è riconducibile a GIP il problema diventa almeno quasi polinomiale.

2.10 Square code

Il calcolo dello square code di un codice permette di ottenere un codice che ha un hull con una dimensione con buona probabilità minore del codice di partenza.

Dati due codici C e C' equivalenti, calcolando lo square code di questi due codici si ottengono dei codici equivalenti legati dalla stessa matrice di permutazione, perché il calcolo dello square code non comporta una combinazione lineare delle colonne e la permutazione avviene solo sulle colonne.

Definizione 2.10.1. Dato un codice lineare C con matrice generatrice $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ definiamo lo square code di C come il codice lineare che ha per matrice generatrice la matrice $square(\mathbf{G}) \in \mathbb{F}_q^{k+k*k\frac{k-1}{2} \times n}$ |

$$square(\mathbf{G})_i = \left[\sum_{j=i}^{k} g_{j,1} \sum_{j=i}^{k} g_{j,2} ... \sum_{j=i}^{k} g_{j,n} \right] \forall i = 1...k$$

Dati due codici lineari per poter risolvere il problema basandosi sugli square code dei codici è necessario che anche gli square code siano dei codici e quindi con dimensione inferiore alla lunghezza cioè con: $k + k\frac{k-1}{2} \leq n \wedge k > 0$ cioè:

$$0 < k \leq \frac{-1 + \sqrt{1+8n}}{2} \tag{2.14}$$

che per n grande diventa:

$$0 < k \leq \sqrt{2n} \tag{2.15}$$

Proposizione 2.10.1. Dati C e C' due codici lineari equivalenti con matrici generatrici $\mathbf{G}$ e $\mathbf{G}' \in \mathbb{F}_q^{k \times n}$, $\mathbf{P} \in \mathbb{F}_q^{n \times n}$ matrice di permutazione e $\mathbf{S} \in \mathbb{F}_q^{k \times k}$ matrice di cambio di base tale per cui

$$\mathbf{G}' = \mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}$$

allora

$$square(\mathbf{G}') = \mathbf{S_{sc}} \cdot square(\mathbf{G}) \cdot \mathbf{P}$$

dove indichiamo con $k^* = k + k * k\frac{k-1}{2}$ e con $\mathbf{S}_{sc}$ una matrice invertibile $k^* \times k^*$.

Dimostrazione. Basta far vedere che:

$$\begin{aligned} square(\mathbf{G}') &= square(\mathbf{S} \cdot \mathbf{G} \cdot \mathbf{P}) \\ &= square\,(\mathbf{S} \cdot \mathbf{G}) \cdot \mathbf{P} \end{aligned} \tag{2.16}$$

L'uguaglianza 2.16 vale perchè $\mathbf{P}$ agisce solo sulle colonne, mentre lo square code non le modifica.

$$
\begin{aligned}
square\,(\mathbf{S}\cdot\mathbf{G}) &= square\left(\begin{bmatrix}\mathbf{S}_1\mathbf{G}^1 & \dots & \mathbf{S}_1\mathbf{G}^n\\ \vdots & & \vdots\\ \mathbf{S}_k\mathbf{G}^1 & \dots & \mathbf{S}_k\mathbf{G}^n\end{bmatrix}\right)\\
&= \begin{bmatrix}(\mathbf{S}_1\mathbf{G}^1)^2 & \dots & (\mathbf{S}_1\mathbf{G}^n)^2\\ \vdots & & \vdots\\ (\mathbf{S}_1\mathbf{G}^1\mathbf{S}_k\mathbf{G}^1) & \dots & (\mathbf{S}_1\mathbf{G}^n\mathbf{S}_k\mathbf{G}^n)\\ (\mathbf{S}_2\mathbf{G}^1)^2 & \dots & (\mathbf{S}_2\mathbf{G}^n)^2\\ \vdots & & \vdots\\ (\mathbf{S}_2\mathbf{G}^1\mathbf{S}_k\mathbf{G}^1) & \dots & (\mathbf{S}_2\mathbf{G}^n\mathbf{S}_k\mathbf{G}^n)\\ \vdots & & \vdots\\ (\mathbf{S}_k\mathbf{G}^1)^2 & \dots & (\mathbf{S}_k\mathbf{G}^n)^2\end{bmatrix}\\
&= \begin{bmatrix}(s_{1,1}g_{1,1}+\dots+s_{1,k}g_{k,1})^2 & \dots & (s_{1,1}g_{1,n}+\dots+s_{1,k}g_{k,n})^2\\ \vdots & & \vdots\\ (s_{k,1}g_{1,1}+\dots+s_{k,k}g_{k,1})^2 & \dots & (s_{1,1}g_{1,n}+\dots+s_{k,k}g_{k,n})^2\end{bmatrix}
\end{aligned}
\tag{2.17}
$$

$$
\begin{aligned}
\mathbf{S}_{sc}\cdot square(\mathbf{G}) &= \mathbf{S}_{sc}\cdot\begin{bmatrix}g_{1,1}^2 & \cdots & g_{1,n}^2\\ \vdots & & \vdots\\ g_{1,1}g_{k,1} & \cdots & g_{1,n}g_{k,n}\\ g_{2,1}^2 & \cdots & g_{2,n}^2\\ \vdots & & \vdots\\ g_{2,1}g_{k,1} & \cdots & g_{2,n}g_{k,n}\\ \vdots & & \vdots\\ g_{k,1}^2 & \cdots & g_{k,n}^2\end{bmatrix}\\
&= \begin{bmatrix}\left(s_{sc1,1}g_{1,1}^2+\dots+s_{sc1,k^*}g_{k,1}^2\right) & \dots & \left(s_{sc1,1}g_{1,n}^2+\dots+s_{sc1,k^*}g_{k,n}^2\right)\\ \vdots & & \vdots\\ \left(s_{sck^*,1}g_{1,1}^2+\dots+s_{sck^*,k^*}g_{k,1}^2\right) & \dots & \left(s_{sck^*,1}g_{1,n}^2+\dots+s_{sck^*,k^*}g_{k,n}^2\right)\end{bmatrix}
\end{aligned}
\tag{2.18}
$$

Date le Equazioni 2.17 e 2.18 possiamo concludere che $square\,(\mathbf{S}\cdot\mathbf{G}) = \mathbf{S}_{sc}\cdot square(\mathbf{G})$ per una $\mathbf{S}_{sc}$ e che quindi $square(\mathbf{G}') = \mathbf{S_{sc}}\cdot square(\mathbf{G})\cdot\mathbf{P}$. $\square$

Applicando lo square code, la dimensione dell'hull del problema tende a diminuire e può anche diventare 0. Questo risultato è particolarmente interessante per codici debolmente auto-duali e, infatti, in questo modo istanze considerate sicure possono essere attaccate in modo più efficiente tramite il calcolo dello square code.

Capitolo 3

Hull dello square code di codici auto-duali

Questo capitolo serve a spiegare la metodologia utilizzata per la ricerca, descrivendo per prima cosa gli esperimenti effettuati e successivamente gli algoritmi utilizzati e testati.

3.1 Metodologia

Il lavoro prova a risolvere il problema dell'equivalenza dei codici riconducendolo al problema dell'isomorfismo dei grafi. Questa trasformazione non è sempre applicabile direttamente ma è condizionata alla dimensione dell'hull dei codici. La trasformazione può essere fatta direttamente quando l'hull dei codici ha dimensione nulla.

Come descritto nella Sezione 2.7.1, il lavoro è stato svolto sia per il problema decisionale sia per quello di ricerca.

Nel caso dell'algoritmo di ricerca l'approccio utilizzato si concentra esclusivamente sulla risoluzione del problema per codici che hanno hull con dimensione nulla, quindi dove la trasformazione può essere effettuata direttamente.

Nel caso dell'algoritmo decisionale, invece, il processo è stato ampliato permettendo di risolvere il problema per qualsiasi tipo di codice. Questo è stato possibile tramite il calcolo degli square code che permette di trasformare il problema di equivalenza dei codici di partenza nel problema di equivalenza degli square code. Il vantaggio sta nel fatto che gli square code hanno tendenzialmente hull con dimensione bassa, se l'hull è nullo è possibile trasformare il problema direttamente nel problema dell'isomorfismo dei grafi, altrimenti si applica una riduzione che permette di ottenere un problema di equivalenza dei codici trasformabile.

Gli esperimenti effettuati partono da quelli preliminari sulla dimensione dell'hull per proseguire poi con l'esperimento sul problema decisionale e su quello di ricerca. Riassumendo quindi gli esperimenti effettuati sono:

- il primo esperimento per stimare la distribuzione di probabilità della dimensione dell'hull per codici casuali;
- il secondo esperimento per stimare la probabilità di avere hull triviale nel caso di codici casuali;

- il terzo esperimento per stimare la distribuzione di probabilità della dimensione dell'hull dello square code di codici debolmente auto-duali;

- il quarto esperimento per stimare la probabilità di avere hull triviale nel caso dello square code di codici debolmente auto-duali;

- il quinto esperimento per stimare la probabilità di successo dell'algoritmo decisionale nel caso di codici debolmente auto-duali;

- il sesto esperimento per stimare la probabilità di successo dell'algoritmo di ricerca nel caso di codici con hull triviale.

Le prove sono state effettuate per valori differenti di n, q e k. In particolare q è stato fatto variare per tutti gli esperimenti tranne che per l'ultimo in:

$$q \in \{2, 3, 5, 7, 11\} \tag{3.1}$$

La coppia (n, k) è stata fatta variare in:

$$\begin{aligned}(n, k) \in \{&(10, 3), (20, 4), (30, 5), (40, 6), (50, 7), \\ &(60, 7), (70, 8), (80, 8), (90, 9), (100, 10), \\ &(110, 10), (120, 10), (130, 11), (140, 11), (150, 12), \\ &(160, 12), (170, 13), (180, 13), (190, 13), (200, 14), \\ &(210, 14), (220, 14), (230, 15), (240, 15), (250, 15), \\ &(260, 16), (270, 16), (280, 16), (290, 17), (300, 17), \\ &(310, 17), (320, 17), (330, 18), (340, 18), (350, 18), \\ &(360, 18), (370, 19), (380, 19), (390, 19), (400, 20), (410, 20)\}\end{aligned} \tag{3.2}$$

oppure in:

$$(n, k) \in \{(10, 3), (110, 10), (210, 14), (310, 17), (410, 20)\} \tag{3.3}$$

In questi casi k è stato scelto pari a:

$$k = \lfloor \sqrt{n} \rfloor \tag{3.4}$$

Questo serve per permettere di calcolare lo square code ottenendo sempre una matrice generatrice di codici come descritto dall'Equazione 2.15.

Mentre per l'ultimo esperimento (n, k) è stata fatta variare in:

$$\begin{aligned}(n,k) \in \{&(10,4),\ (20,6),\ (30,7),\ (40,8),\ (50,10),\\ &(60,10),\ (70,11),\ (80,12),\ (90,13),\ (100,14),\\ &(110,14),\ (120,15),\ (130,16),\ (140,16),\ (150,17),\\ &(160,17),\ (170,18),\ (180,18),\ (190,19),\ (200,20),\\ &(210,20),\ (220,20),\ (230,21),\ (240,21),\ (250,22),\\ &(260,22),\ (270,23),\ (280,23),\ (290,24),\ (300,24),\\ &(310,24),\ (320,25),\ (330,25),\ (340,26),\ (350,26),\\ &(360,26),\ (370,27),\ (380,27),\ (390,27),\ (400,28),\ (410,28)\}.\end{aligned} \tag{3.5}$$

Tenendo conto che k viene calcolato come:

$$k = \lfloor \sqrt{2 \times n} \rfloor. \tag{3.6}$$

3.1.1 Esperimenti preliminari

Data l'importanza delle dimensioni dell'hull, inizialmente sono stati condotti degli esperimenti sulla distribuzione delle dimensioni dell'hull per codici casuali. Questo per osservare quanto può aumentare la dimensione dell'hull all'aumentare di n e per vedere come varia la probabilità di avere hull triviale.

Il primo esperimento consiste nel generare 100 codici casuali con valori di q specificati in 3.1 e per valori di (n, k) definiti in 3.3. Per ogni iterazione viene determinato l'hull del codice e la sua dimensione.

Sia

$\alpha(n, q, dim(hull)) = \#$iterazioni con dimensione dell'hull pari a $dim(hull)$ e per n, q

e

$$\beta(n, q) = \#\text{iterazioni per } n, q$$

Allora la probabilità empirica di una certa dimensione di hull per una coppia di q e n viene calcolcata come segue:

$$P(n, q, dim(hull)) = \frac{\alpha(n, q, dim(hull))}{\beta(n, q)} \tag{3.7}$$

La distribuzione teorica dell'hull per codici casuali ha un comportamento asintotico per n che dipende solamente da q e varia nel seguente modo:

$$P(q, dim(hull)) = (1 - \frac{1}{q}) \prod_{1}^{dim(hull)} \frac{1}{q^i - 1} \tag{3.8}$$

Inoltre è stato realizzato un secondo esperimento per calcolare la probabilità che l'hull sia triviale nel caso di codici casuali. L'esperimento è stato condotto per un numero maggiore di n per vedere meglio come varia. La prova si basa sulla generazione di 100 codici casuali al variare di q come definito in 3.1 e di (n, k) come descritto da 3.2. Per ogni codice viene determinato l'hull e la sua dimensione e si verifica se è triviale. Il calcolo della probabilità che l'hull sia triviale è stato effettuato come segue:

$$P(n, q) = \frac{\#\text{iterazioni con hull triviale per } n, q}{\#\text{iterazioni per } n, q} \tag{3.9}$$

Siccome, nel caso dell'algoritmo decisionale, vengono calcolati gli square code per i codici con hull non triviale e viene poi risolto il problema sugli square code, sono stati condotti degli esperimenti per calcolare la distribuzione empirica di probabilità della dimensione dell'hull degli square code. Il caso più complesso è quello di codici debolmente auto-duali, cioè dove $C \subseteq C^{\perp}$, per i quali la dimensione dell'hull è massima e pari a k, per questo motivo le analisi sulla dimensione dell'hull degli square code sono state realizzate su codici debolmente auto-duali.

Il terzo esperimento è simile al primo, perciò permette di calcolare la distribuzione di probabilità della dimensione dell'hull al variare di n, q e k. Questo esperimento prevede di generare 100 codici debolmente auto-duali per q definito da 3.1 e per (n, k) descritti da 3.3. Durante ogni iterazione si determina prima lo square code del codice, poi l'hull dello square code e infine la dimensione dell'hull. La procedura utilizzata per calcolare la probabilità empirica di una certa dimensione di hull per una coppia di q e n è definita dall'Equazione 3.7.

Il quarto esperimento è simile al secondo esperimento, però per codici debolmente auto-duali. Permette quindi di calcolare la probabilità empirica di avere hull triviale degli square code di codici debolmente auto-duali per un numero maggiore di n. Questo esperimento consiste nel generare 100 codici debolmente auto-duali per q definito da 3.1 e per (n, k) descritto da 3.2. L'iterazione è uguale al caso precedente, quindi viene calcolato lo square code del codice, l'hull dello square code e la dimensione dell'hull, si verifica infine se l'hull è triviale. La probabilità che l'hull sia triviale viene calcolata tramite l'Equazione 3.9.

3.1.2 Esperimento sull'algoritmo decisionale

Infine sono stati effettuati degli esperimenti per verificare la distribuzione di probabilità di successo dell'algoritmo decisionale nel caso di codici debolmente auto-duali al variare di n, q e k. Nel caso dell'algoritmo di ricerca invece sono state calcolate le probabilità di successo nel caso di codici con hull triviale al variare di n, q e k.

Il quinto esperimento che viene eseguito è sull'algoritmo decisionale. Anche in questo caso le prove vengono effettuate con codici con q definito da 3.1 e con (n, k) descritto da 3.2. L'esperimento per ogni coppia n e q genera 100 coppie di codici debolmente auto-duali che sono aleatoriamente equivalenti oppure casuali. Una coppia di codici casuali raramente saranno equivalenti, soprattutto per n e q grandi. Una volta che una coppia viene generata, viene calcolato lo square code e se l'hull è triviale viene chiamato direttamente l'algoritmo che trasforma il problema del code equivalence nel problema dell'isomorfismo dei grafi. Verrà quindi generata una risposta se i codici sono equivalenti o meno, nel caso in cui l'hull non sia triviale vengono applicate altre trasformazioni, successivamente spiegate in pseudo-codice, per ottenere un problema trasformabile nel problema dell'isomorfismo dei grafi, sul quale viene richiamato lo stesso algoritmo di prima e viene generata una risposta. Poi viene controllato che la risposta sia corretta. Una volta che questo è stato eseguito per tutte e 100 le coppie di codici viene calcolata la probabilità di successo per n e q come:

$$P(n, q) = \frac{\#\text{successi}n, q}{\#\text{iterazioni per } n, q} \tag{3.10}$$

Il sesto esperimento viene eseguito sull'algoritmo di ricerca. Queste prove sono state realizzate con codici con hull triviale e con q descritto da 3.1 e con (n, k) definito da 3.2, quindi sempre con k definito da 3.4. Inoltre per vedere meglio se cambia la probabilità al variare del rapporto tra n e k sono stati realizzati dei test ponendo k come descritto da 3.6 quindi con (n, k) definiti da 3.5. Avendo hull triviale la trasformazione del problema è sempre possibile. L'esperimento quindi per ogni coppia n e q genera 100 coppie di codici con hull triviale equivalenti. Questo perché il problema di ricerca si appoggia a quello decisionale per determinare se i codici sono equivalenti, risulta quindi necessario solamente testare quante volte nel caso di codici equivalenti vengono trovate le giuste matrici di permutazione $\mathbf{P}$ e la matrice di cambio di base $\mathbf{S}$. A questo punto il problema di equivalenza dei codici viene trasformato nel problema dell'isomorfismo dei grafi; l'algoritmo che risolve il problema di ricerca permette nel caso in cui i codici siano equivalenti di ottenere delle coppie di matrici $\mathbf{P}$ ed $\mathbf{S}$; se una di queste coppie permette di andare da un codice all'altro allora l'algoritmo ha trovato una soluzione corretta. La probabilità di successo di questo algoritmo sarà quindi calcolata con l'Equazione 3.10.

3.2 Algoritmo decisionale

La risoluzione del problema decisionale nel caso generale di due codici è descritto dalla Figura 3.1. Anche da qui si può notare come la dimensione dell'hull è importantissima per determinare la velocità con cui verrà prodotta la risposta. Le parti più importanti sono state descritte in pseudo-codice.

Il cuore dell'algoritmo decisionale è stato descritto in pseudo-codice dalla Funzione 5. Questa funzione riceve un campo finito $\mathbb{F}_q$ e due matrici generatrici $\mathbf{G}$ e $\mathbf{G}'$ e ritorna come valore un Booleano che rappresenta se i codici sono equivalenti o meno. Per prima cosa vengono determinati gli hull dei codici tramite la Funzione 1 e si confrontano le dimensioni degli hull perchè se sono diverse i codici saranno sicuramente

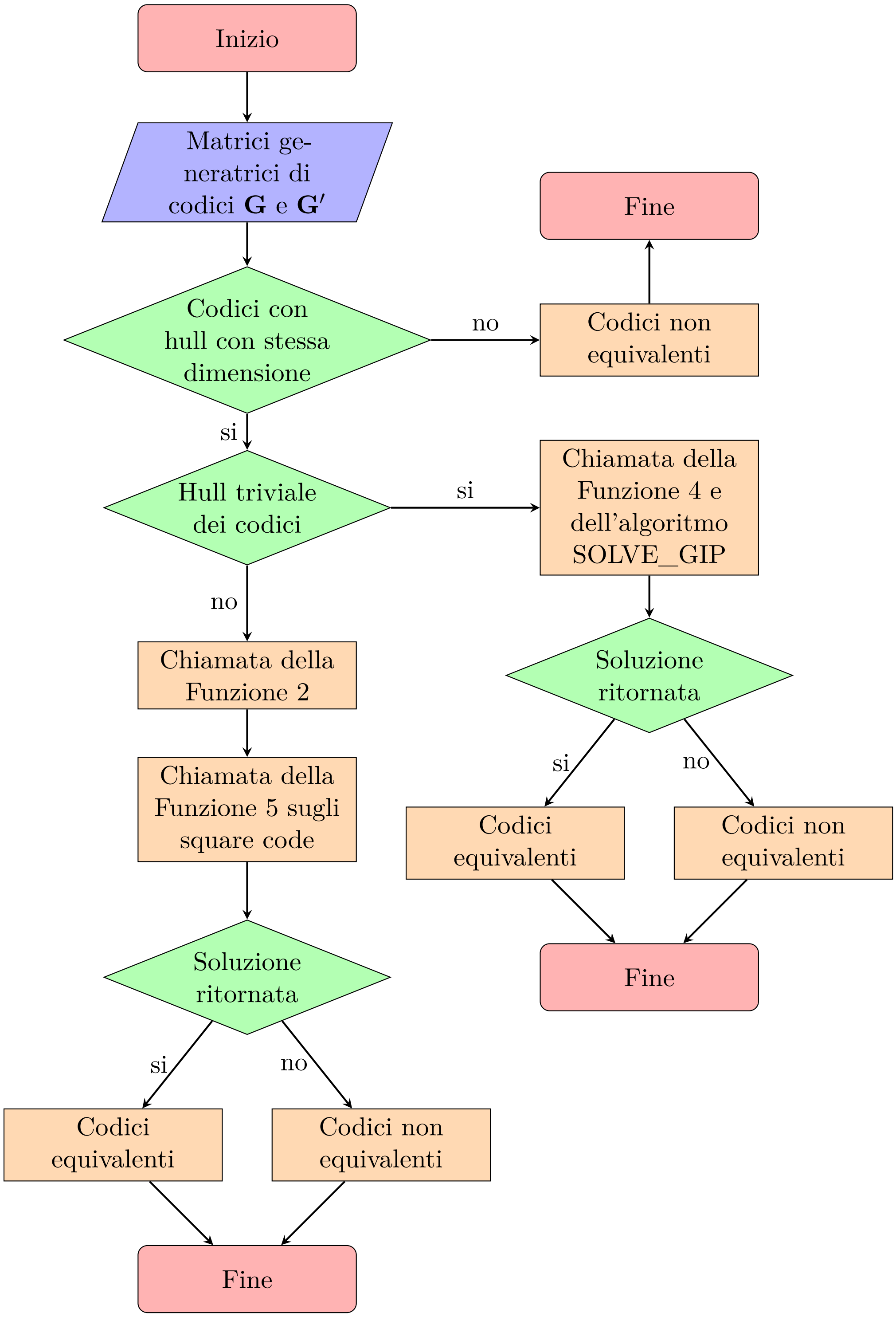

Figura 3.1: Diagramma di flusso per l'algoritmo decisionale

non equivalenti. Se sono uguali invece si controlla la dimensione. Nel caso in cui l'hull sia triviale vengono calcolati i grafi associati alle matrici generatrici chiamando la Funzione 4 e poi viene chiamato il solutore per il problema dell'isomorfismo dei grafi e la soluzione viene ritornata. Nel caso in cui invece la dimensione degli hull sia maggiore di 0 allora va trovata una trasformazione del problema che richiede tanto più tempo quanto più è grande l'hull. Per prima cosa viene determinato un information set della matrice $\mathbf{G}$ di dimensione pari a quella dell'hull. Un information set è un insieme di k indici di colonne di $\mathbf{G}$ linearmente indipendenti. Viene poi chiamata la Funzione 3 sulla matrice $\mathbf{G}$ e sull'information set determinato. Questa funzione calcola un codice che ha dimensione ridotta della dimensione dell'hull cioè:

$$\mathbf{G}_{shortened} \in \mathbb{F}_q^{k+dim(hull)\times n-dim(hull)}.$$

A questo punto per ogni information set di $\mathbf{G}'$ si calcola il codice ridotto allo stesso modo del caso precedente. Vengono calcolati i grafi associati ai codici ridotti e viene chiamato il solutore del problema dell'isomorfismo dei grafi. Se si trova un problema ridotto con soluzione affermativa allora anche il problema principale ha soluzione affermativa, altrimenti no.

Funzione 1 Hull generator

function HULL_GENERATOR($\mathbb{F}_q$ a finite field, $\mathbf{G}$ a generator matrix in the field)
  $\mathbf{H} \leftarrow$ parity check matrix of $\mathbf{G}$
  $\mathbf{M} \leftarrow \begin{bmatrix} \mathbf{G} \\ \mathbf{H} \end{bmatrix}$
  $Hull \leftarrow$ right kernel of $\mathbf{M}$
  return $Hull$
end function

3.3 Algoritmo di ricerca

L'algoritmo di ricerca è stato per ora sviluppato solo per risolvere il caso di codici con hull triviale, quindi dove la trasformazione del problema è immediata, e per $q = 2$ essendo questo il caso più complesso. Sfrutta ovviamente la logica del caso decisionale per determinare se i codici sono equivalenti ma in più determina come sono legati i codici.

La prima parte di questo algoritmo permette quindi di risolvere il problema decisionale tramite la Funzione 6, che permette però anche di restituire l'insieme delle possibili matrici di permutazione $\mathbf{P}$ sotto forma del dizionario *possible_permutation*, che ha per chiave gli indici delle righe di $\mathbf{A}_1$ che possono andare in righe di $\mathbf{A}_2$. Il dizionario è ordinato in base alla dimensione dell'insieme delle righe di $\mathbf{A}_1$ che possono andare in $\mathbf{A}_2$. In questo modo gli spostamenti certi, cioè quelli che fanno corrispondere ad una sola riga di $\mathbf{A}_1$ una sola riga di $\mathbf{A}_2$, si trovano per primi.

Funzione 2 Square code

```
function SQUARE_CODE(G a generator matrix in the field)
    k ← number of rows of G
    n ← number of columns of G
    F_q ← finite field of G
    square_code(G) ← F_q^{k+k×(k−1)/2 ×n}
    row_index ← 0
    for i ← 0 to k − 1 do
        for j ← i to k − 1 do
            for col ← 0 to n − 1 do
                square_code(G)[row_index, col] ← G[i, col] × G[j, col]
            end for
            row_index ← row_index + 1
        end for
    end for
    return square_code(G)
end function
```

Funzione 3 Code shortening

```
function CODE_SHORTHENING(F_q, G, columns)
    H ← matrice di parità di G
    num_colonne_eliminare ← numero di elementi di columns
    num_colonne_H ← numero di colonne di H
    A ∈ F_q^{num_colonne_eliminare×num_colonne_H} ← matrice di 0
    for i ← 0 to num_colonne_eliminare do A[i, columns[i]] ← 1
    end for
    hext ← [H; A]
    Gext ← matrice di parità di Hext
    G_shortened ← Gext da cui vengonno eliminate le colonne columns
    return G_shorted;
end function
```

Funzione 4 Graph from generator matrix

```
function GRAPH_FROM_GENERATOR_MATRIX(G)
    return G^T(GG^T)^{−1}G
end function
```

Funzione 5 Reduce PEP to GIP

function REDUCE_PEP_TO_GIP($\mathbb{F}_q$ a finite field, $\mathbf{G}$ a generator matrix in the field, $\mathbf{G}'$ a generator matrix in the field)
    hull_G ← HULL_GENERATOR($\mathbb{F}_q$, $\mathbf{G}$)
    hull_G' ← HULL_GENERATOR($\mathbb{F}_q$, $\mathbf{G}'$)
    dim_hull_G ← rango di *hull_G*
    dim_hull_G' ← rango di *hull_G'*
    if *dim_hull_G* ≠ *dim_hull_G'* **then**
        return False
    end if
    if *dim_hull_G* = 0 **then**
        graph_G ← GRAPH_FROM_GENERATOR_MATRIX($\mathbf{G}$)
        graph_G' ← GRAPH_FROM_GENERATOR_MATRIX($\mathbf{G}'$)
        solution ← SOLVE_GIP(**graph_G**, **graph_G'**)
        return *solution*
    end if
    if *dim_hull_G* > 0 **then**
        ▷ Cerca un information set di *hull_G* controllando tra le possibili combinazioni di colonne
        all_column_sets ← insieme delle combinazioni di *dim_hull_G* elementi da un insieme con cardinalità n
        current_set_G ← primo elemento di *all_column_sets*
        while La dimensione della matrice che ha per colonne l'insieme *current_set_G* delle colonne di *hull_G* < *dim_hull_G* **do**
            current_set_G ← elemento successivo di *all_column_sets*
        end while
        shortened_G ← CODE_SHORTENING($\mathbb{F}_q$, $\mathbf{G}$, *current_set_G*)
        graph_shortened_G ← GRAPH_FROM_GENERATOR_MATRIX(**shortened_G**)
        for *current_set_G'* in *all_column_sets* **do** ▷ Nel caso della matrice *hull_G'* cerca tra tutti gli information set
            if la dimensione della matrice che ha per colonne l'insieme *current_set_G'* delle colonne di *hull_G'* = *dim_hull_G'* **then**
                shortened_G' ← CODE_SHORTENING($\mathbb{F}_q$, $\mathbf{G}'$, *current_set_G'*)
                graph_shortened_G' ← GRAPH_FROM_GENERATOR_MATRIX(**shortened_G'**)
                solution ← SOLVE_GIP(**graph_shortened_G**, **graph_shortened_G'**)
                if *solution* **then return** *solution*
                end if
            end if
        end for
    end if
    return *False*
end function

Funzione 6 Risolutore del problema decisionale k

function SOLVER($\mathbf{A}_1$, $\mathbf{A}_2$)
▷ Siccome le matrici di adiacenza sono matrici simmetriche, i valori sulla diagonale rimangono gli stessi in ordine diverso a causa delle matrici di permutazione
    if La diagonale di $\mathbf{A}_1$ non ha gli stessi elementi della diagonale di $\mathbf{A}_2$ **then return** $False$, []
    end if
▷ Siccome le matrici di adiacenza sono matrici simmetriche, i valori sopra la diagonale rimangono gli stessi in ordine diverso a causa delle matrici di permutazione
    if Gli elementi sopra la diagonale di $\mathbf{A}_1$ non sono gli stessi elementi sopra la diagonale di $\mathbf{A}_2$ **then return** $False$, []
    end if
    $row\_labels_1 \leftarrow$ dizionario con chiavi (valore della diagonale della riga i, numero di 1 nella riga i) e valori liste di righe i della matrice $\mathbf{A}_1$
    $row\_labels_2 \leftarrow$ dizionario con chiavi (valore della diagonale della riga i, numero di 1 nella riga i) e valori liste di righe i della matrice $\mathbf{A}_2$
    $possible\_permutation \leftarrow$ dizionario che mappa le righe di $\mathbf{A}_1$ che possono andare in righe di $\mathbf{A}_2$
    for key_1 in chiavi di row_labels_1 **do**
        $row\_indices_1 \leftarrow row\_labels_1[key_1]$
        $row\_indices_2 \leftarrow row\_labels_2[key_1]$
▷ Se il numero di righe associate ad una stessa chiave non è lo stesso per $\mathbf{A}_1$ e per $\mathbf{A}_2$, allora le due matrici non sono legate da una permutazione
        if lunghezza di $row\_indices_1 \neq$ lunghezza di $row\_indices_2$ **then return** $False$, []
        end if
        $possible\_permutation[row\_indices_1] \leftarrow row\_indices_2$
    end for
    $possible\_permutation \leftarrow$ ordinato per dimensione delle chiavi o dei valori
return $True$, $possible\_permutation$
end function

Questo algoritmo è a forza bruta ma lavora su un sotto problema di dimensione almeno k, invece che lavorare sul problema di dimensione n, perché in tal caso la forza bruta non sarebbe stata applicabile.

Date due matrici generatrici di codici equivalenti $\mathbf{G}$ e $\mathbf{G}' \in \mathbb{F}_q^{k\times n}$, una matrice di cambio di base $\mathbf{S} \in \mathbb{F}_q^{k\times k}$ e una matrice di permutazione $\mathbf{P} \in \mathbb{F}_q^{n\times n}$ tale che $\mathbf{G}' = \mathbf{SGP}$, allora data una sotto matrice quadrata $\tilde{\mathbf{G}} \in \mathbb{F}_q^{k\times k}$ di $\mathbf{G}$ invertibile che tramite la permutazione va in $\tilde{\mathbf{G}'} \in \mathbb{F}_q^{k\times k}$ sotto matrice di $\mathbf{G}'$ tale che $\tilde{\mathbf{G}'} = \mathbf{S}\tilde{\mathbf{G}}\tilde{\mathbf{P}}$ ed essendo invertibile $\mathbf{S} = \tilde{\mathbf{G}'}\tilde{\mathbf{P}}^{-1}\tilde{\mathbf{G}}^{-1}$. Una volta trovato $\mathbf{S}$ risolvendo il sotto problema è possibile trovare anche $\mathbf{P}$ e ottenere la soluzione del problema di ricerca. Per trovare $\tilde{\mathbf{P}}$ si lavora sulle sotto matrici delle matrici dei grafi associati.

Quindi dopo aver determinato le possibili permutazioni è necessario ridurre il problema ad un sotto problema con la Funzione 7 che riceve le matrici generatrici, i grafi associati e il dizionario *possible_permutation* e ritorna le matrici ridotte; è presente anche un parametro, che di default è 0, che permette di ottenere un sotto problema di dimensione aumentata rispetto a k per cercare su più sotto matrici in modo da aumentare la probabilità di trovarne una invertibile.

La Funzione 7 ha permesso di determinare le sotto matrici da cui ottenere $\tilde{\mathbf{G}}$ e $\tilde{\mathbf{G}'}$ ma anche le sotto matrici di $\mathbf{A}$ e $\mathbf{A}'$ tramite le quali è possibile determinare $\tilde{\mathbf{P}}$. La matrice $\tilde{\mathbf{P}}$ viene determinata per forza bruta grazie al dizionario *sub_possible_permutation*, che contiene le righe di $\mathbf{A}_{1,sub}$ che possono andare nelle righe di $\mathbf{A}_{2,sub}$; quindi ci dà informazioni sulle posizioni degli 1 nella matrice di permutazione nel seguente modo: se la riga i di $\mathbf{A}_{1,sub}$ può andare nella riga j di $\mathbf{A}_{2,sub}$ allora $\tilde{\mathbf{P}}[i,j]$ potrebbe essere 1 siccome $\mathbf{A}_{2,sub} = \tilde{\mathbf{P}}^T \times \mathbf{A}_{1,sub} \times \tilde{\mathbf{P}}$. L'algoritmo quindi analizza in base al dizionario tutte le possibili combinazioni di 1 e determina l'insieme di possibili matrici di permutazione. La matrice $\tilde{\mathbf{P}}$ serve per determinare $\mathbf{S}$ da cui si può poi ricavare $\mathbf{P}$. La funzione utilizzata per determinare l'insieme dei $\tilde{\mathbf{P}}$ che formano un endomorfismo è la Funzione 8, che riceve $\mathbf{A}_{2,sub}, \mathbf{A}_{2,sub}$ e $sub\_possible_permutation$. Notiamo che in generale questa funzione potrebbe essere utilizzata anche senza diminuire la dimensione del problema ma l'efficienza sarebbe di molto inferiore.

Una volta ottenuta la lista di possibili $\tilde{\mathbf{P}}$ è necessario determinare, ove possibile, i corrispondenti $\mathbf{S} = \tilde{\mathbf{G}'} \times \tilde{\mathbf{P}}^{-1} \times \tilde{\mathbf{G}}^{-1}$. Se $\tilde{\mathbf{G}}$ ha dimensione minore di k allora non è invertibile. La Funzione 9 cerca di determinare $\tilde{\mathbf{G}}^{-1}$ e $\tilde{\mathbf{G}'}^{-1}$ a partire da $\mathbf{G}_{1,sub}$ e $\mathbf{G}_{2,sub}$ che siano però invertibili. Per ogni matrice $\tilde{\mathbf{G}}$ invertibile determina anche $\mathbf{S}$. Permette quindi di ottenere una lista di possibili matrici di cambio di base e va successivamente verificato quale di queste è corretta.

Una volta determinata la lista di $\mathbf{S}$ è possibile determinare le matrici di permutazione $\mathbf{P}$ associate. Si può a questo punto verificare quali di queste rappresentano soluzioni controllando che $\mathbf{G}' = \mathbf{SGP}$. Le matrici di equivalenza non sono infatti univoche, può quindi capitare che esistano più matrici che mandano un codice in un altro. Si considera pertanto un successo semplicemente quando viene trovata una soluzione. Trovare la $\mathbf{P}$ associata ad $\mathbf{S}$ è piuttosto semplice e viene effettuato tramite la Funzione 10. Questa funzione riceve due matrici con la stessa dimensione e

Funzione 7 Determina il sotto problema di dimensione almeno k

```
function GET_SUB_MATRICES(G_1 matrice generatrice, G_2 matrice generatrice, A_1 matrice del grafo associato, A_2 matrice del grafo associato, possible_permutation dizionario che mappa le righe di A_1 che possono andare in righe di A_2, k dimensione, aumented = 0 serve per lavorare su un sottoproblema più grande)      ▷ Per prima cosa è necessario determinare la dimensione minima del sotto problema tenendo conto che vanno tenuti uniti gli insiemi di righe definiti a partire dal dizionario possible_permutation
    counter ← 0                                        ▷ Dimensione del sotto problema
    list_of_keys ← lista vuota
    list_of_values ← lista vuota
    for chiave in chiave di possible_permutation do
        for i in chiave do
            inserisci ordinato i in list_of_keys
        end for
        for i in possible_permutation[chiave] do
            inserisci ordinato i in list_of_values
        end for
        counter ← counter+ lunghezza di chiave
        if counter ≥ k + aumented then
            break
        end if
    end for
                                        ▷ Creazione della sottomatrice di A_1 e A_2
    A_{1,sub} ← matrice di dimensione counter × counter
    A_{2,sub} ← matrice di dimensione counter × counter
    for i ← 0 to counter − 1 do
        for j ← 0 to counter − 1 do
            A_{1,sub}[i, j] ← A_1[list_of_keys[i], list_of_keys[j]]
            A_{2,sub} ← A_2[list_of_values[i], list_of_values[j]]
        end for
    end for
                                        ▷ Creazione della sottomatrice di G_1 e G_2
    G_{1,sub} ← matrice di dimensione k × counter
    G_{2,sub} ← matrice di dimensione k × counter
    for i ← 0 to k − 1 do
        for j ← 0 to counter − 1 do
            A_{1,sub}[i, j] ← A_1[list_of_keys[i], list_of_keys[j]]
            A_{2,sub} ← A_2[list_of_values[i], list_of_values[j]]
        end for
    end for
```

Funzione 7 Determina il sotto problema di dimensione almeno k (continua)

▷ Creazione del dizionario legato al sotto problema

```
    counter2 ← 0
    sub_possible_permutation ← dizionario
    for key in chiavi di possible_permutation do
        sub_key ← lista
        sub_value ← lista
        for i in key do
            for j ← 0 to counter − 1 do
                if list_of_keys[j] == i then
                    aggiungere j a sub_key
                    break
                end if
            end for
        end for
        for i in possible_permutation[key] do
            for j ← 0 to counter − 1 do
                if list_of_values[j] == i then
                    aggiungere j a sub_value
                    break
                end if
            end for
        end for
        sub_possible_permutation[sub_key] ← sub_value
        counter ← counter+ lunghezza di key
        if counter2 ≥ counter then
            break
        end if
    end forreturn G_{1,sub}, G_{2,sub}, A_{1,sub}, A_{2,sub}, sub_possible_permutation
end function
```

Funzione 8 Determina la lista di $\tilde{\mathbf{P}}$

function GET_P_TILDE($\mathbf{A}_1, \mathbf{A}_2, sub\_possible\_permutation$)
  ngets numero di colonne di $\mathbf{A}_1$
  sub_permutations dizionario che ha per chiave le chiavi di *sub_possible_permutation* e per valore una lista delle permutazioni equivalenti che legano le sotto matrici.
  for *key* in lista delle chiavi di *sub_possible_permutation* **do**
    $n\_sub \leftarrow$ lunghezza di *key*
    $\mathbf{A}_{1,sub} \leftarrow$ matrice di dimensione n_sub
    $\mathbf{A}_{2,sub} \leftarrow$ matrice di dimensione n_sub
    for $i \leftarrow 0$ to $n\_sub - 1$ **do**
      for $j \leftarrow 0$ to $n\_sub - 1$ **do**
        $\mathbf{A}_{1,sub} \leftarrow \mathbf{A}_1[key[i], key[j]]$
        $\mathbf{A}_{2,sub} \leftarrow \mathbf{A}_2[sub\_possible_permutation[i], sub\_possible\_permutation[j]]$
      end for
    end for
    permutations lista delle permutazioni di dimensione n_sub ▷ La permutazione è rappresentata come lista ordinata
    sub_permutation_list gets lista
    for *permutation* in *permutations* **do**
      $\mathbf{P}_{sub} \leftarrow$ matrice di permutazione associata a *permutation*
      if $\mathbf{A}_{2,sub} = \mathbf{P}_{sub}^T \mathbf{A}_{1,sub} \mathbf{P}_{sub}$ **then**
        $\mathbf{P}_{sub}$ aggiungere a *sub_permutation_list*
      end if
    end for
    $sub\_permutations[key] \leftarrow sub\_permutation\_list$
  end for
  combination_list $\leftarrow$ lista di dizionari dove ognuno rappresenta una combinazione di sotto permutazioni. Ogni dizionario ha quindi per chiavi le chiavi di *sub_permutations* e per valori una combinazione delle possibili sotto permutazioni.
  list_P_tilde $\leftarrow$ lista delle matrici **P**_*tilde*
  for *combination* in *combination_list* **do**
    $\mathbf{P}_0 \leftarrow$ matrice di dimensione n
    for *key* in *combination* **do**
      $n\_sub \leftarrow$ lunghezza di *key*
      for $i \leftarrow 0$ to $n\_sub - 1$ **do**
        for $j \leftarrow 0$ to $n\_sub - 1$ **do**
          $\mathbf{P}_0[key[i], sub\_possible\_permutation[key][j]] \leftarrow combination[key][i, j]$
        end for
      end for
    end for
    if thenA$_2 = \mathbf{P}_0^T \times \mathbf{A}_1 \times \mathbf{P}_0$
      aggiungere $\mathbf{P}_0$ alla lista *list_P_tilde*
    end if
  end for
  return *list_P_tilde*
end function

Funzione 9 Determina una lista di possibili S

function GET_S($\mathbf{G}_{1,sub}$, $\mathbf{G}_{2,sub}$, *list_P_tilde*, k)
    list_of_S lista delle soluzioni **S**
    for result in *list_P_tilde* **do**
        if numero di colonne di $\mathbf{G}_{1,sub} = k$ **then**
            if rango di $\mathbf{G}_{1,sub} = k$ **then**
                aggiungere $\mathbf{G}_{2,sub}\mathbf{result}^{-1}\mathbf{G}_{1,sub}^{-1}$
            end if
        else ▷ Se la matrice non è quadrata si cercano tutte le sottomatrici quadrate di rango k (se esistono)
            disposition_of_columns_to_delete ← lista delle disposizioni di numero di colonne di $\mathbf{G}_{1,sub} - k$ da un insieme di numero di colonne di $\mathbf{G}_{1,sub}$
            for *columns_to_delete* in *disposition_of_columns_to_delete* **do**
                $\tilde{\mathbf{G}}_1$ matrice di dimensione k
                $\tilde{\mathbf{G}}_2$ matrice di dimensione k
                $\tilde{\mathbf{P}}$ matrice di dimensione k
                columns_to_include ← colonne non in *columns_to_delete* ▷ Determinare le righe corrispondenti da eliminare sulla matrice result che corrispondono alle colonne da eliminare di $\mathbf{G}_{2,sub}$
                rows_to_delete ← lista delle righe da eliminare
                for *row* in *columns_to_delete* **do**
                    for $j \leftarrow 0$ to numero di colonne di $\mathbf{G}_{1,sub} - 1$ **do**
                        if $result[row, j] = 1$ **then**
                            aggiungere j alla lista *rows_to_delete*
                        end if
                    end for
                end for
                rows_to_include insieme delle righe non contenuto in *rows_to_delete*
                $\tilde{\mathbf{G}}_1 \leftarrow$ le colonne *column_to_include* di $\mathbf{G}_{1,sub}$
                $\tilde{\mathbf{G}}_2 \leftarrow$ le colonne *rows_to_include* di $\mathbf{G}_{2,sub}$
                $\tilde{\mathbf{P}} \leftarrow$ le colonne *column_to_include* e le righe *rows_to_include* di **result**
                if rango di $\tilde{\mathbf{G}}_1 = k$ **then**
                    aggiungere $\tilde{\mathbf{G}}_2\tilde{\mathbf{P}}^{-1}\tilde{\mathbf{G}}_1^{-1}$
                    break
                end if
            end for
        end if
    end for
    return *list_of_S*
end function

trova, se esiste, la permutazione $\mathbf{P}$ che lega $\mathbf{G}_1$ e $\mathbf{G}_2$ nel seguente modo: $\mathbf{G}_2 = \mathbf{G}_1\mathbf{S}$. Quindi la funzione dovrà essere chiamata su $\mathbf{SG}$ e $\mathbf{G}'$ per determinare $\mathbf{P}$ associata all'equivalenza.

Funzione 10 Determinare $\mathbf{P}$ associato ad $\mathbf{S}$

```
function FIND_PERMUTATION(G1, G2)
    n ← numero di colonne di G1
    pemutazione vettore di n elementi pari a −1
    G1_copy ← G1
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            if G1[: i] = G2[: j] then permutazione[i] = j + 1
                G1_copypo5[: j] ← Null
                break
            end if
        end for
    end for
    if permutazione contiene almeno un −1 then return Null
    end if
    P ← matrice di permutazione associata a permutazione
    return P
end function
```

Capitolo 4

Nuovo risolutore: risultati sperimentali

Questo capitolo riporta tutti i risultati sperimentali ottenuti a seguito degli esperimenti descritti nel capitolo precedente. I risultati sono stati riportati sotto forma di grafici e sono stati analizzati sulla base delle considerazioni teoriche.

4.1 Dimensione dell'hull

I primi esperimenti effettuati sono stati quelli sull'hull e sulla sua dimensione, visto che l'efficienza degli algoritmi dipende in gran parte da questo valore. Valori piccoli della dimensione dell'hull garantiscono una maggiore efficienza, inoltre se fosse 0, l'efficienza è ancora maggiore non essendo necessario calcolare lo square code, ma basta trasformare il problema di equivalenza nel problema di isomorfismo dei grafi.

Il primo esperimento, cioè quello sulla distribuzione della dimensione dell'hull per codici casuali, ha riportato i risultati visibili nei piani cartesiani delle Figure 4.1, 4.2, 4.3, 4.4 e 4.5, che sono divisi in base alla dimensione del campo finito rispettivamente per $q \in \{2, 3, 5, 7, 11\}$. Ogni piano cartesiano riporta sull'asse delle ascisse la dimensione dell'hull e sull'asse delle ordinate la probabilità, e rappresenta 5 curve empiriche al variare di n oltre alla curva teorica corrispondente al campo. La distribuzione teorica asintotica con n della dimensione dell'hull dovrebbe variare con q come descritto dalla Formula 3.8. Essendo un comportamento asintotico ovviamente è indipendente da n. Notiamo che teoricamente la probabilità di avere un hull con dimensione grande decade in fretta, quindi già per dimensioni superiori a 4 la probabilità è quasi 0. Inoltre, decade più in fretta all'aumentare di q.

I risultati sperimentali confermano proprio queste considerazioni. Infatti la probabilità all'aumentare della dimensione dell'hull decade molto in fretta e in modo molto più rapido all'aumentare di q. Inoltre nonostante il risultato teorico fosse un comportamento asintotico per n, il risultato sperimentale non si discosta molto nemmeno per valori piccoli di n. Anche osservando le dimensioni empiriche medie dell'hull possiamo notare questo comportamento, infatti per $q = 2$ la dimensione media dell'hull è circa 0.752, per $q = 3$ è 0.384, per $q = 5$ è 0.198, per $q = 7$ è 0.156 e per $q = 11$ è 0.098.

L'esperimento due effettuato per un numero maggiore di valori è servito per vedere come cambia la probabilità di avere hull triviale al variare di q e n. Serviva a verificare

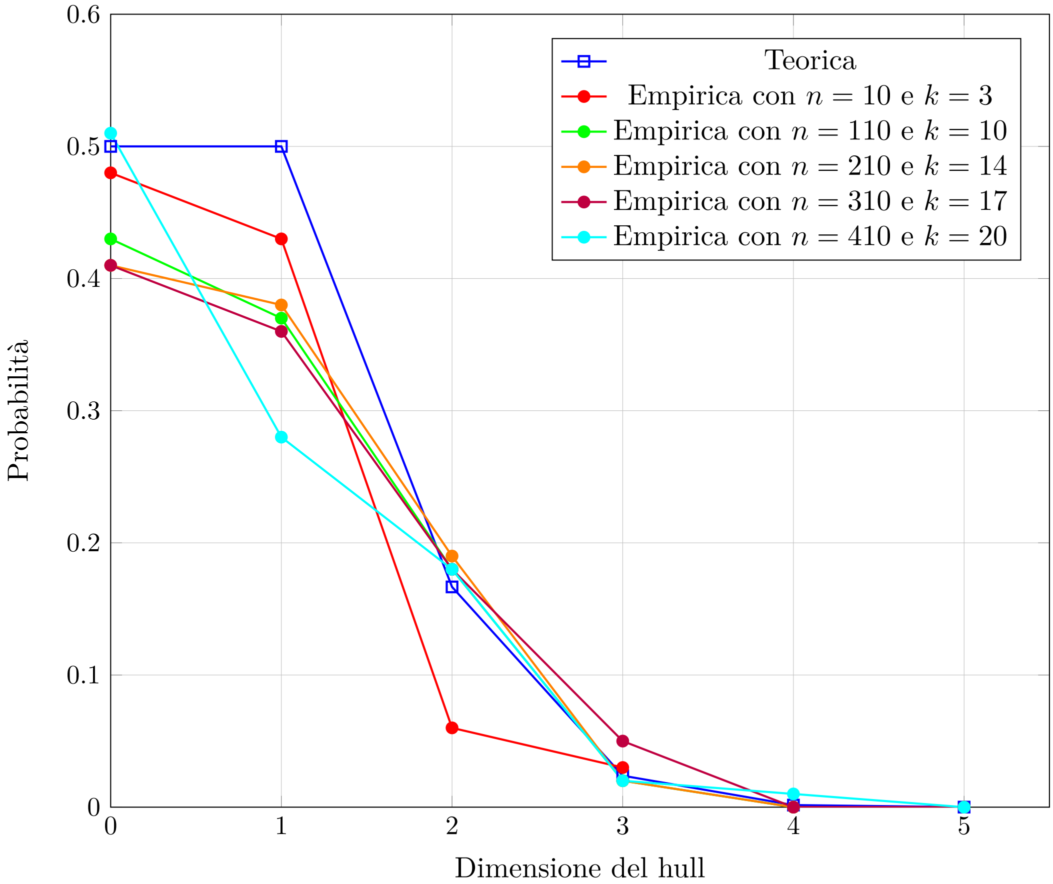

Figura 4.1: Confronto tra la distribuzione empirica e teorica della dimensione dell'hull per $q = 2$

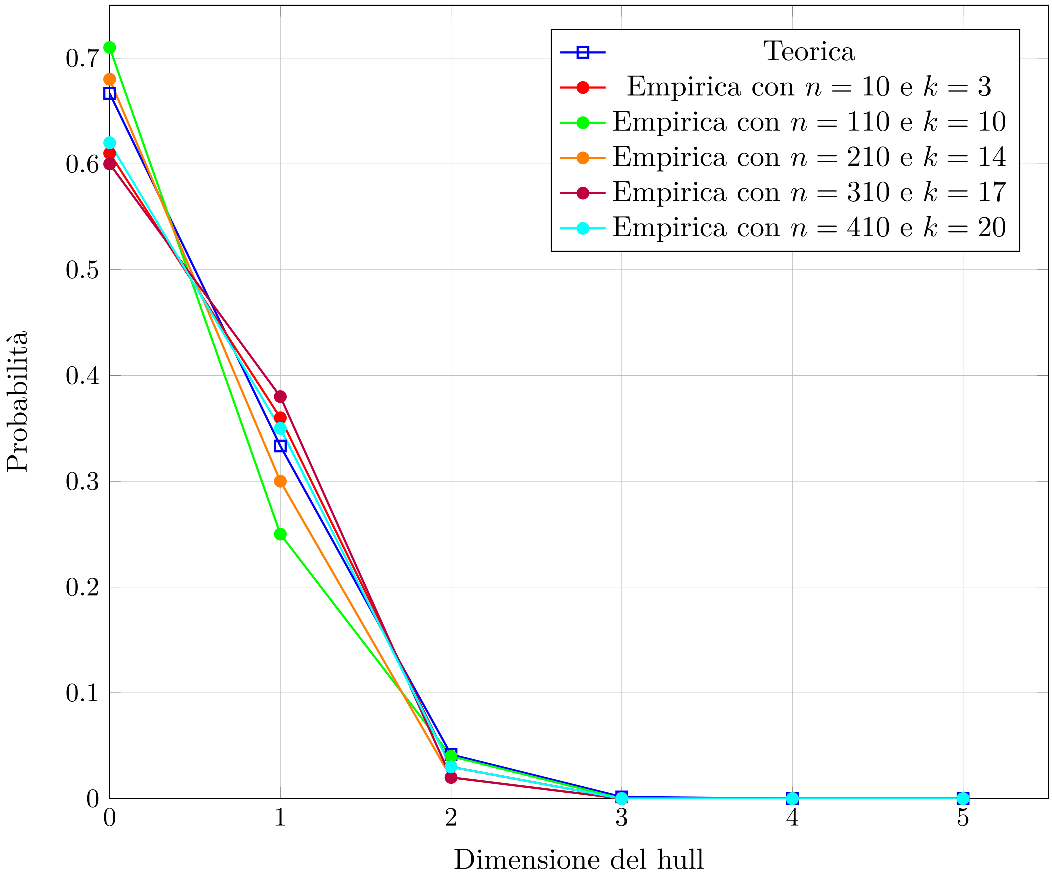

Figura 4.2: Confronto tra la distribuzione empirica e teorica della dimensione dell'hull per $q = 3$

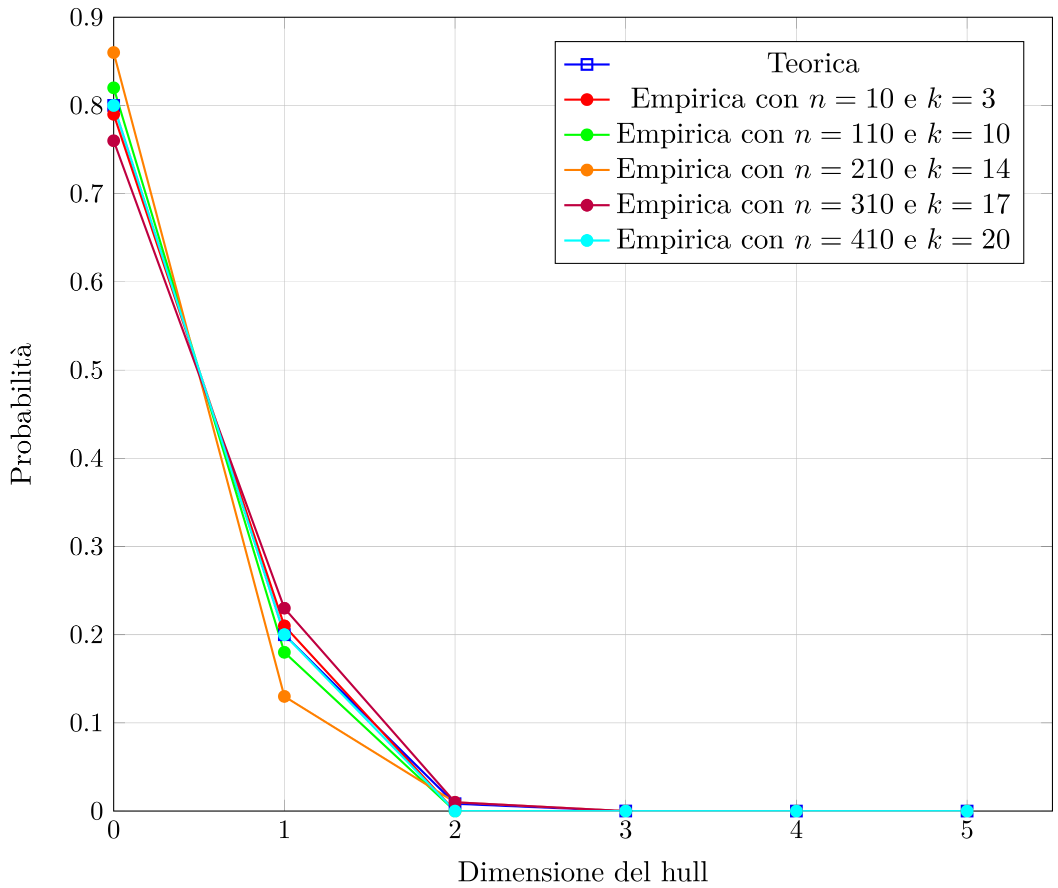

Figura 4.3: Confronto tra la distribuzione empirica e teorica della dimensione dell'hull per $q = 5$

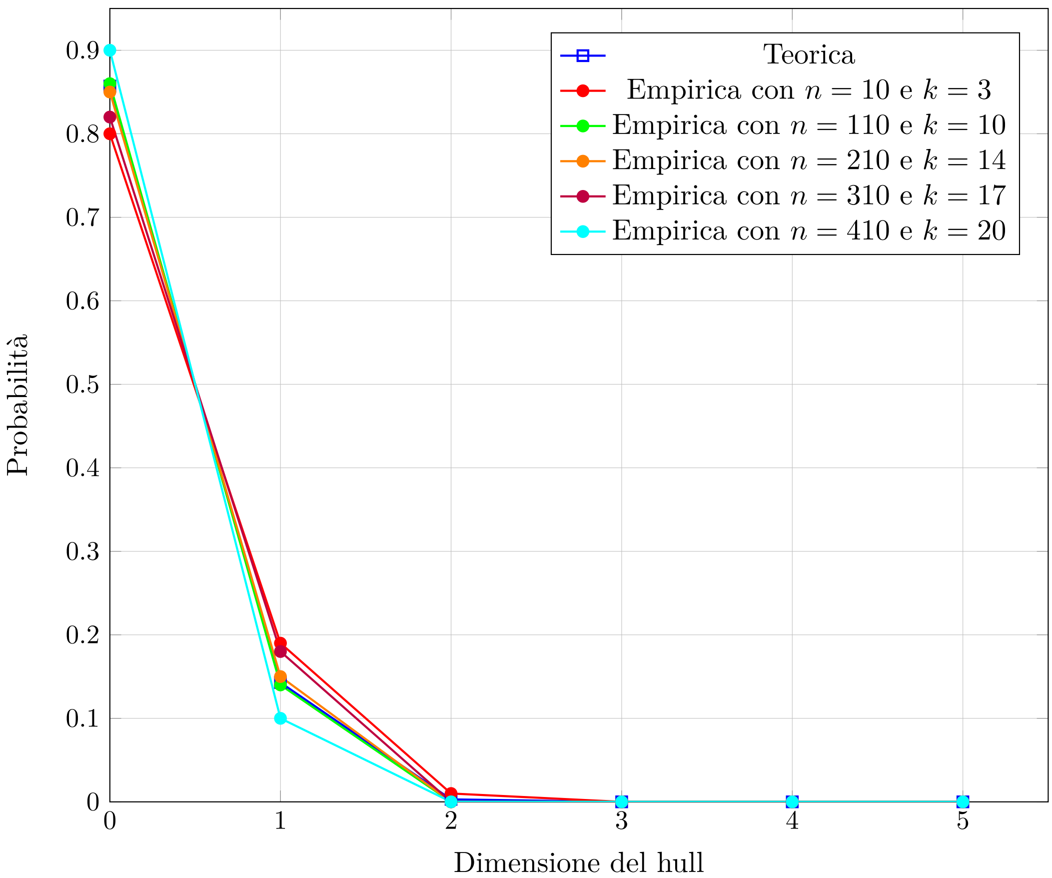

Figura 4.4: Confronto tra la distribuzione empirica e teorica della dimensione dell'hull per $q = 7$

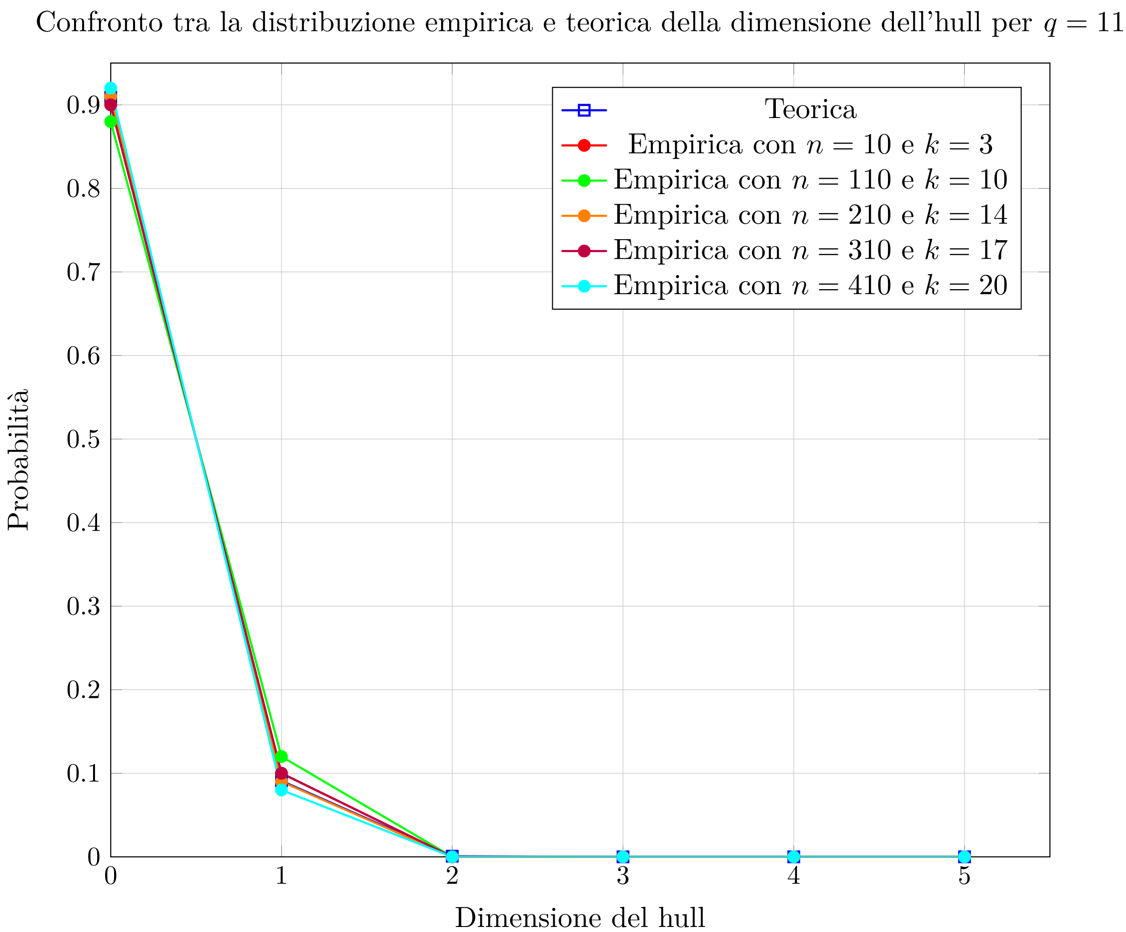

Figura 4.5: Confronto tra la distribuzione empirica e teorica della dimensione dell'hull per $q = 11$

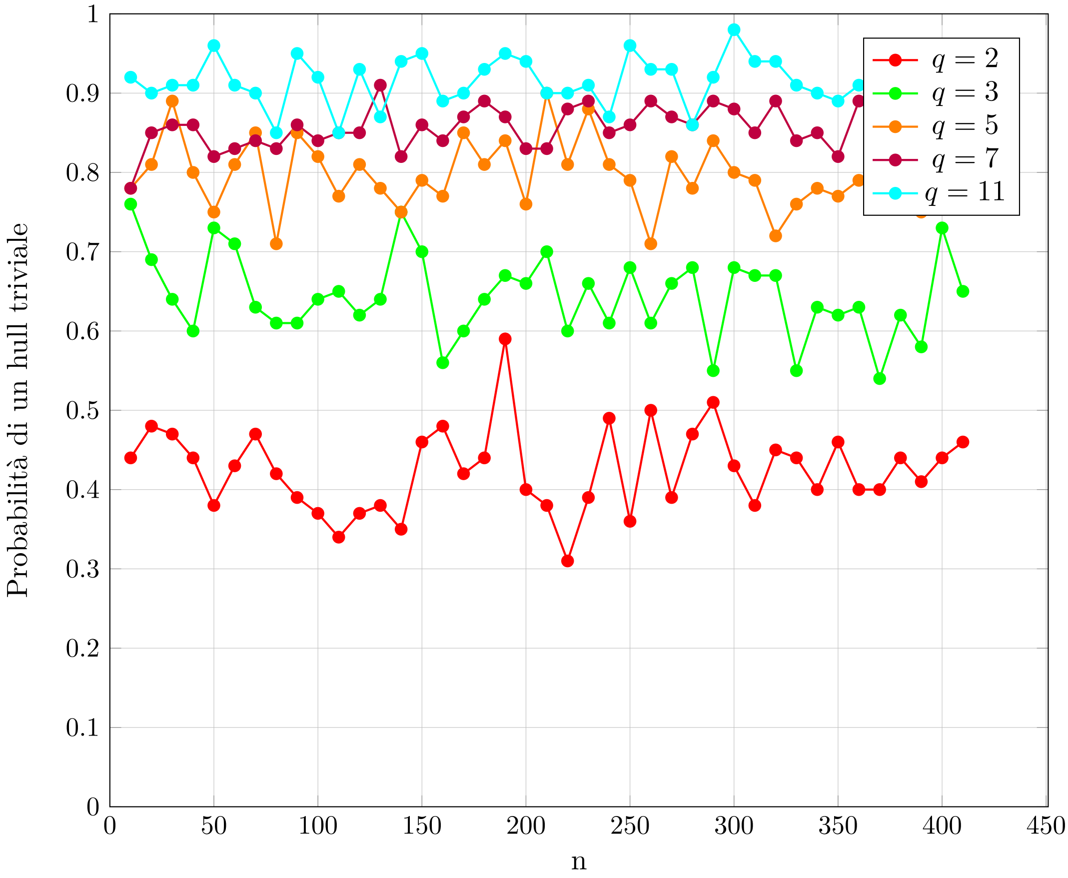

Figura 4.6: Probabilità di avere hull triviali, al variare di n

che non ci fosse una qualche tendenza nella variazione della probabilità al variare di n. I risulati vengono riportati nel piano cartesiano in Figura 4.6, che riporta sull'asse delle ascisse la dimensione n e sull'asse delle ordinate la probabilità e presenta 5 grafici per i vari valori di q. Le osservazioni portano a pensare che questa probabilità non vari al variare di n ma aumenti all'aumentare di q. Da questi risultati possiamo quindi osservare che anche nel caso peggiore di $q = 2$ la probabilità che l'hull sia triviale è piuttosto alta e indipendente da n quindi è possibile per moltissimi codici trasformare direttamente il problema senza necessità di calcolare lo square code. Infatti la probabilità media per $q = 2$ è $\frac{1743}{4100} \approx 0.425$, per $q = 3$ è $\frac{2643}{4100} \approx 0.645$, per $q = 5$ è $\frac{817}{1025} \approx 0.797$, per $q = 7$ è $\frac{878}{1025} \approx 0.857$, per $q = 11$ è $\frac{3739}{4100} \approx 0.912$. L'algoritmo di ricerca così come è stato implementato può essere utilizzato su poco meno della metà dei codici generati casualmente per $q = 2$.

Il caso più complesso da studiare è quello dei codici debolmente auto-duali, per i quali la dimensione dell'hull è pari a k. In questo caso non solo non sarebbe possibile applicare la trasformazione del problema direttamente, ma l'hull risulta anche di dimensione elevata e quindi una riduzione sarebbe lunga da calcolare. É possibile però calcolare lo square code del codice e lavorare sul problema di equivalenza dello square code.

q	**Dimensione media dell'hull di codici casuali**	**Dimensione media dell'hull dello square code di codici debolmente auto-duali**
2	0.752	1.622
3	0.384	0.57
5	0.198	0.208
7	0.156	0.17
11	0.098	0.088

Tabella 4.1: Confronto tra le dimensioni medie dell'hull di codici casuali o dello square code di codici debolmente auto-duali

Il terzo esperimento è stato quello sulla distribuzione di probabilità della dimensione dell'hull dello square code di codici debolmente auto-duali e ha riportato i risultati rappresentati nei piani cartesiani delle Figure 4.7, 4.8, 4.9, 4.10 e 4.11, che sono divisi in base alla dimensione del campo finito rispettivamente per $q \in 2, 3, 5, 7, 11$. Ogni piano cartesiano riporta sull'asse delle ascisse la dimensione dell'hull e sull'asse delle ordinate la probabilità e presenta 5 curve sperimentali al variare di n.

In questo caso bisogna distinguere i risultati per $q = 2$ e per $q \neq 2$. Infatti quando $q = 2$ la distribuzione di probabilità della dimensione varia in modo poco prevedibile ma sembra avere dei comportamenti simili per gruppi di n. Mentre nel caso $q \neq 2$ sembra che la distribuzione di probabilità sia paragonabile a quella dei codici casuali; questo è un grandissimo vantaggio perchè in questo modo anche nel caso peggiore di codici debolmente auto-duali in realtà ci si riconduce al caso dei codici casuali, che possono essere risolti rapidamente nella maggioranza dei casi, dato che è alta la probabilità che l'hull abbia dimensione nulla, ma anche in caso contrario la dimensione è piccola e quindi comunque più facilmente risolvibile. É possibile guardare anche la Tabella 4.1 per vedere che per $q = 2$ la dimensione media è molto differente, mentre per $q \neq 2$ la dimensione media si discosta poco dal caso aleatorio. Nonostante per $q = 2$ la distribuzione di probabilità è differente dal caso in cui i codici vengano generati in modo casuale, la dimensione dell'hull risulta comunque piccola e quindi il problema è risolvibile.

Il quarto esperimento, cioè quello sulla probabilità che l'hull dello square code di codici debolmente auto-duali sia triviale, ha prodotto i risultati rappresentati nel piano cartesiano della Figura 4.12, che presenta sull'asse delle ascisse la dimensione n e sull'asse delle ordinate la probabilità, e rappresenta 5 grafici al variare di $q \in 2, 3, 5, 7, 11$. Come per l'esperimento precedente, il caso $q = 2$ risulta differente dagli altri. Infatti per $q \neq 2$ la distribuzione della probabilità risulta uguale alla distribuzione casuale. Il caso $q = 2$, invece, ha degli intervalli di n per i quali la probabilità di avere hull triviale è nulla e altri intervalli in cui la probabilità è confrontabile con il caso di codici generati casualmente. Possiamo osservare questo comportamento guardando la Tabella 4.2, dove è possibile vedere che per $q \neq 2$ le probabilità sono simili, mentre per $q = 2$ sono diverse a meno che non consideriamo

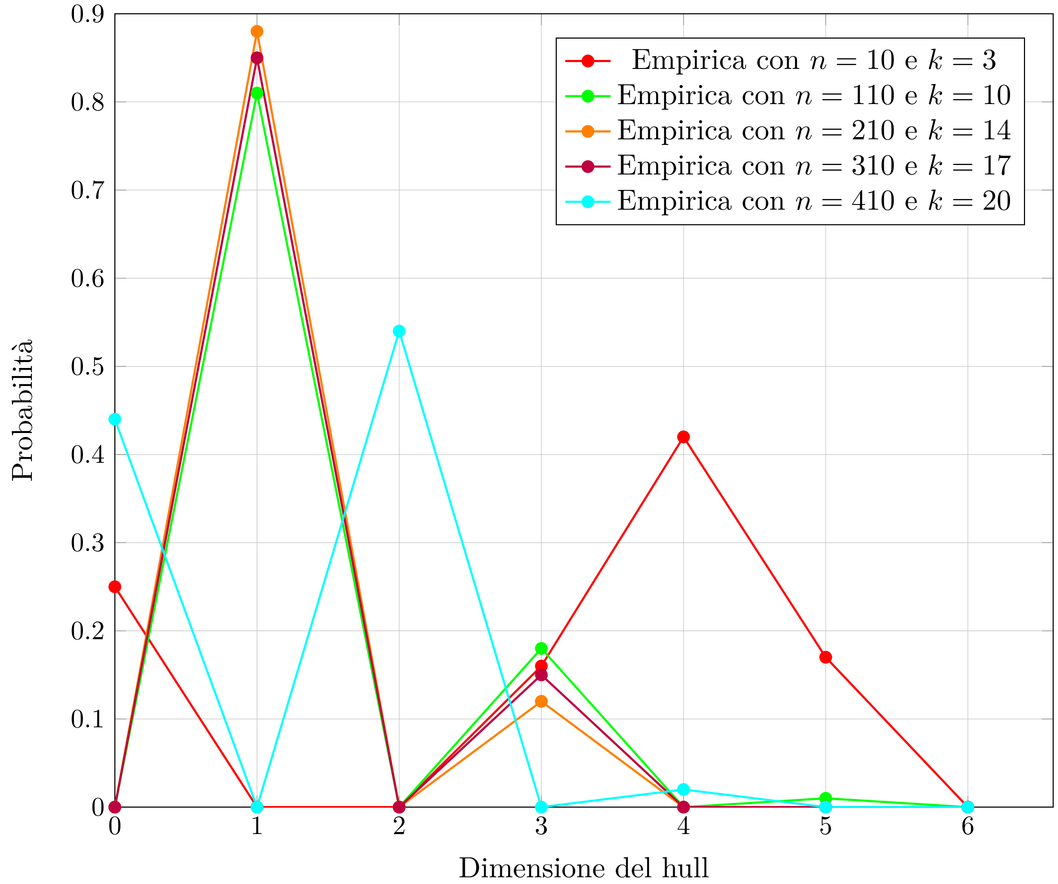

Figura 4.7: Distribuzione di probabilità della dimensione dell'hull dello square code per $q = 2$

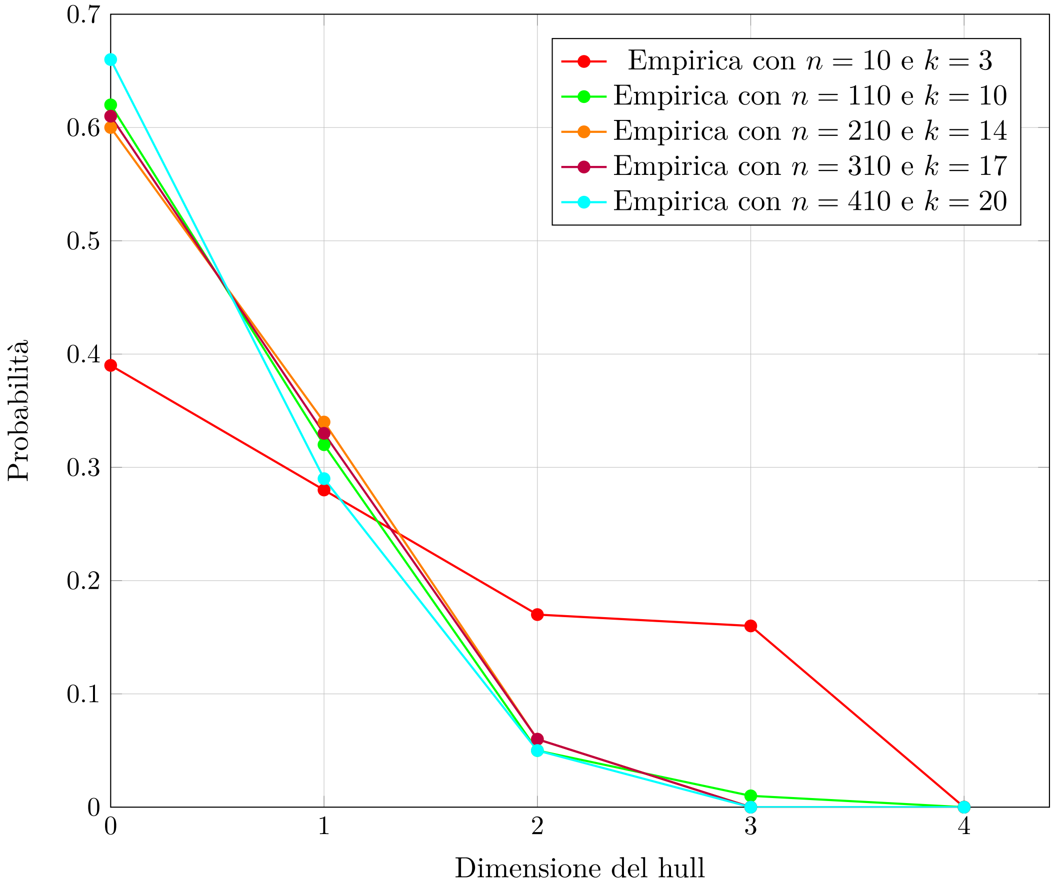

Figura 4.8: Distribuzione di probabilità della dimensione dell'hull dello square code per $q = 3$

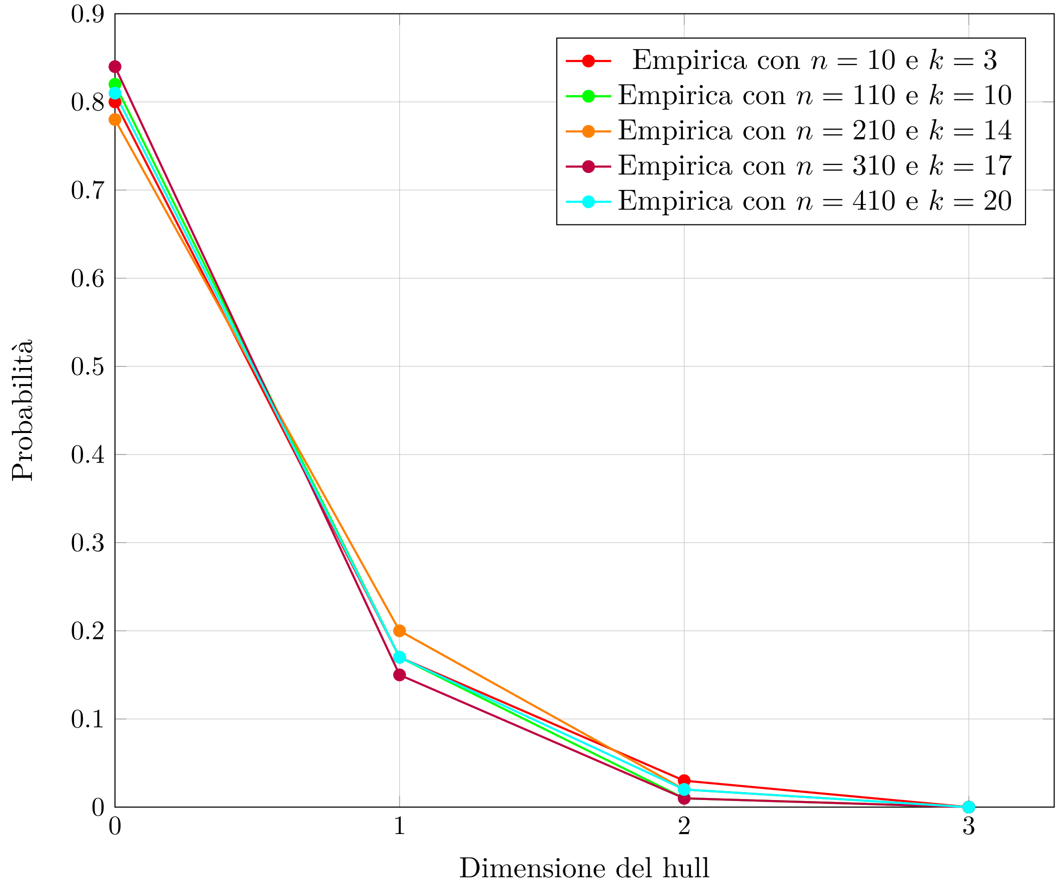

Figura 4.9: Distribuzione di probabilità della dimensione dell'hull dello square code per $q = 5$

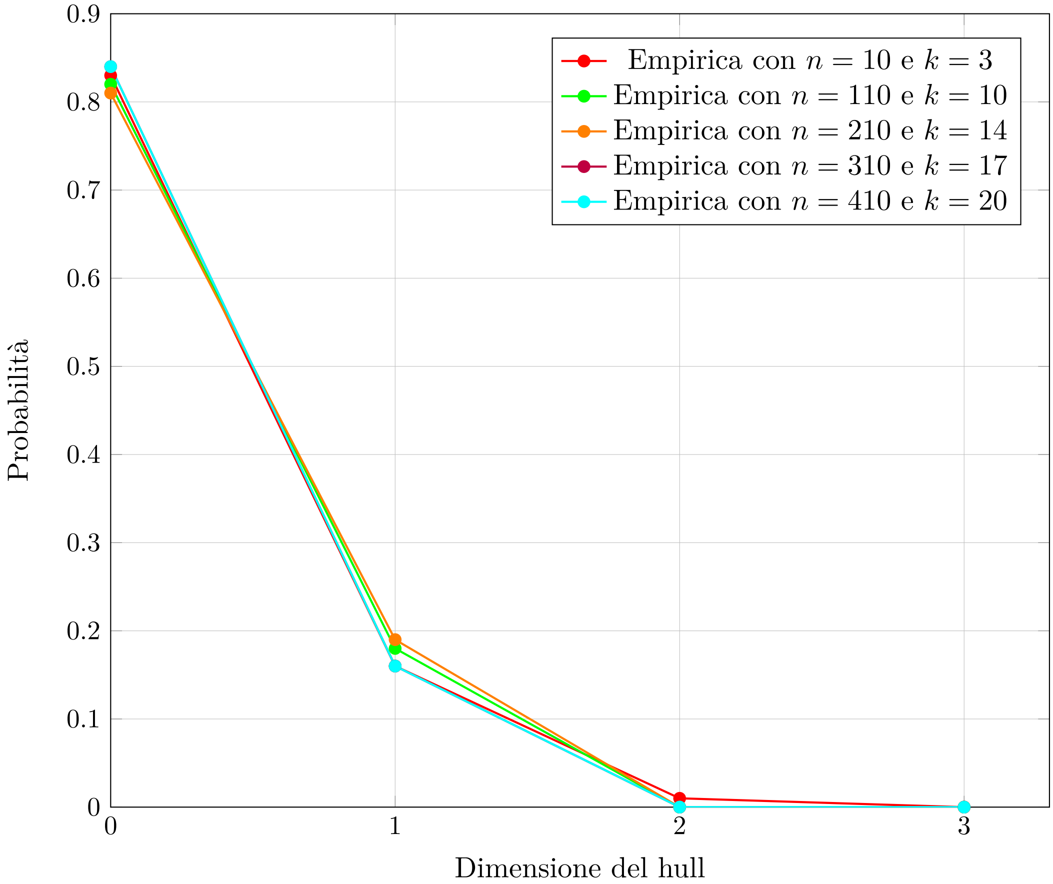

Figura 4.10: Distribuzione di probabilità della dimensione dell'hull dello square code per $q = 7$

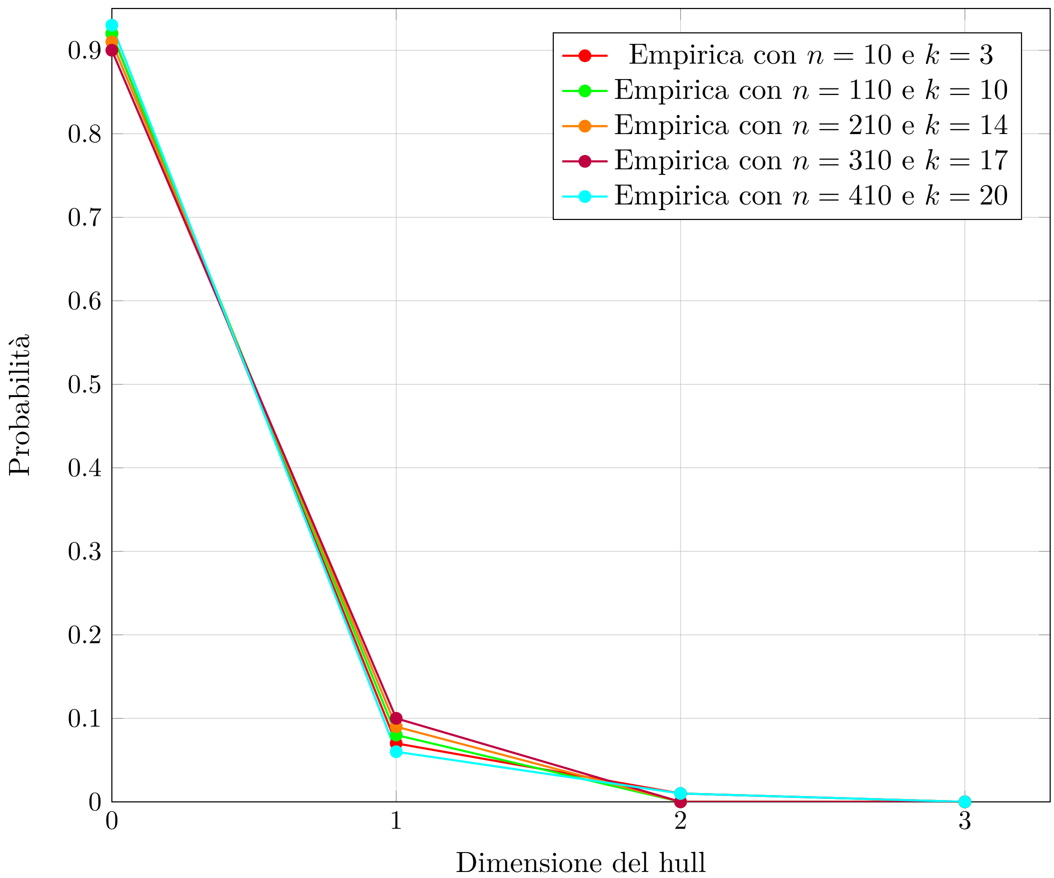

Figura 4.11: Distribuzione di probabilità della dimensione dell'hull dello square code per $q = 11$

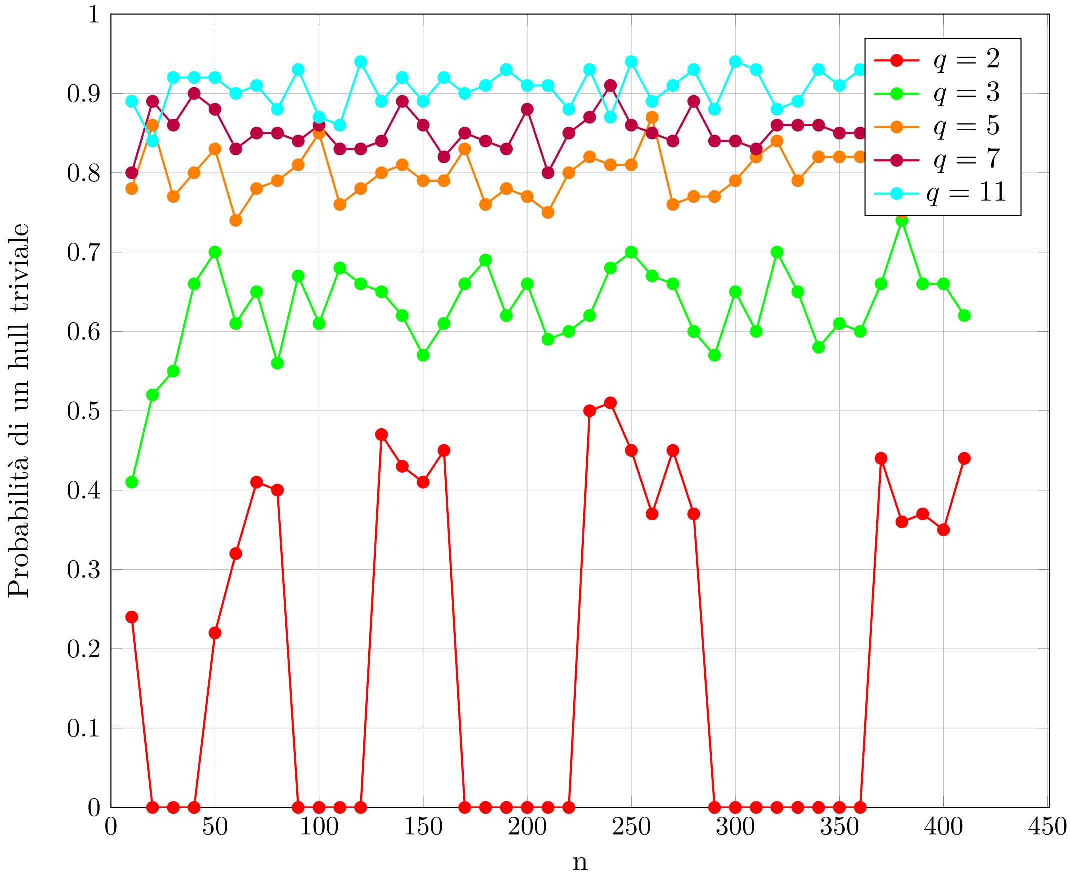

Figura 4.12: Probabilità di avere hull triviali al variare di n

la media solo dei casi in cui la probabilità non è nulla.

Questa differente distribuzione per il caso $q = 2$ può essere spiegata osservando il suo comportamento differente. Infatti $\forall c \in C c \times c = c \implies C \subset C_{squared}$. Quindi il codice è sempre contenuto nel codice squared solo per $q = 2$. Questo potrebbe portare ad un comportamento differente nelle dimensioni dell'hull, che non assume più una distribuzione simile al caso di codici generati casualmente, ma che dipende dal codice di partenza.

4.2 Algoritmo decisionale

Nel caso dell'algoritmo decisionale è stato analizzato il caso più complesso da risolvere, ovvero quello con codici debolmente auto-duali dove la dimensione dell'hull è massima. I risultati servono ad analizzare come cambia la probabilità di successo dell'algoritmo al variare di n, k e q, vedere quindi quante volte l'algoritmo riesce a riconoscere che i codici sono equivalenti o meno. I risultati sono riportati sul piano cartesiano della Figura 4.13, che ha sull'asse delle ascisse la dimensione n e sull'asse delle ordinate la probabilità di successo, e raffigura 5 grafici ognuno per un valore di q. La probabilità di successo di questo algoritmo è molto alta e aumentando la

q	**Probabilità media di hull triviale di codici casuali**	**Probabilità media di hull triviale dello square code di codici debolmente auto-duali**
2	$\frac{1743}{4100} \approx 0.425$	$\frac{199}{1025} \approx 0.194$ se consideriamo la media tra le probabilità non nulle otteniamo 0.398
3	$\frac{2643}{4100} \approx 0.645$	$\frac{1289}{2050} \approx 0.629$
5	$\frac{817}{1025} \approx 0.797$	$\frac{3269}{4100} \approx 0.797$
7	$\frac{878}{1025} \approx 0.857$	$\frac{3483}{4100} \approx 0.850$
11	$\frac{3739}{4100} \approx 0.912$	$\frac{928}{1025} \approx 0.905$

Tabella 4.2: Confronto tra le probabilità di hull triviale di codici casuali o dello square code di codici debolmente auto-duali

dimensione diventa quasi unitaria. A parità di q si ha minore probabilità di successo per minori dimensioni, questo forse perché per n più piccolo è più probabile che ci siano delle corrispondenze tra codici che sono stati generati in modo casuale e che vengono contati come equivalenti dall'algoritmo, quindi viene dato un falso positivo. All'aumentare di q la probabilità di successo aumenta e risulta anche più rapido l'algoritmo, perché mediamente è più probabile che l'hull sia triviale.

Questi risultati comunque garantiscono la possibilità di risolvere il problema decisionale con un ottimo grado di certezza anche nel caso peggiore.

4.3 Algoritmo di ricerca

Siccome l'algoritmo di ricerca è stato sviluppato solo per codici con hull triviale, i test sono stati effettuati solo su questi codici. Ovviamente le probabilità di successo attese erano inferiori rispetto a quelle dell'algoritmo decisionale perché la riduzione del problema a dimensione k ha un successo che è probabilistico e non deterministico; non c'è la certezza di trovare sempre la sotto matrice di dimensione k con rango pieno $\tilde{\mathbf{G}}$. Inoltre, potrebbe anche capitare di trovare la matrice $\tilde{\mathbf{P}}$ errata che non permette di trovare quindi il giusto $\mathbf{S}$. I risultati sono riportati sul piano cartesiano della Figura 4.14, che ha sull'asse delle ascisse la dimensione n e su quello delle ordinate la probabilità di successo, e vengono rappresentati due grafici uno per $k = \lfloor\sqrt{n}\rfloor$ e l'altro per $k = \lfloor\sqrt{2 \times n}\rfloor$. I risultati mostrano proprio quello che ci aspettavamo, cioè probabilità di successo inferiori ma comunque utili a risolvere il problema.

Inoltre, sarebbe stato normale attendersi che la probabilità di successo diminuisse al diminuire del rapporto $\frac{n}{k}$, perché le sottomatrici $\tilde{\mathbf{G}}$ sono pari al numero di combinazioni di k colonne da un insieme di dimensione n, che risulta maggiore

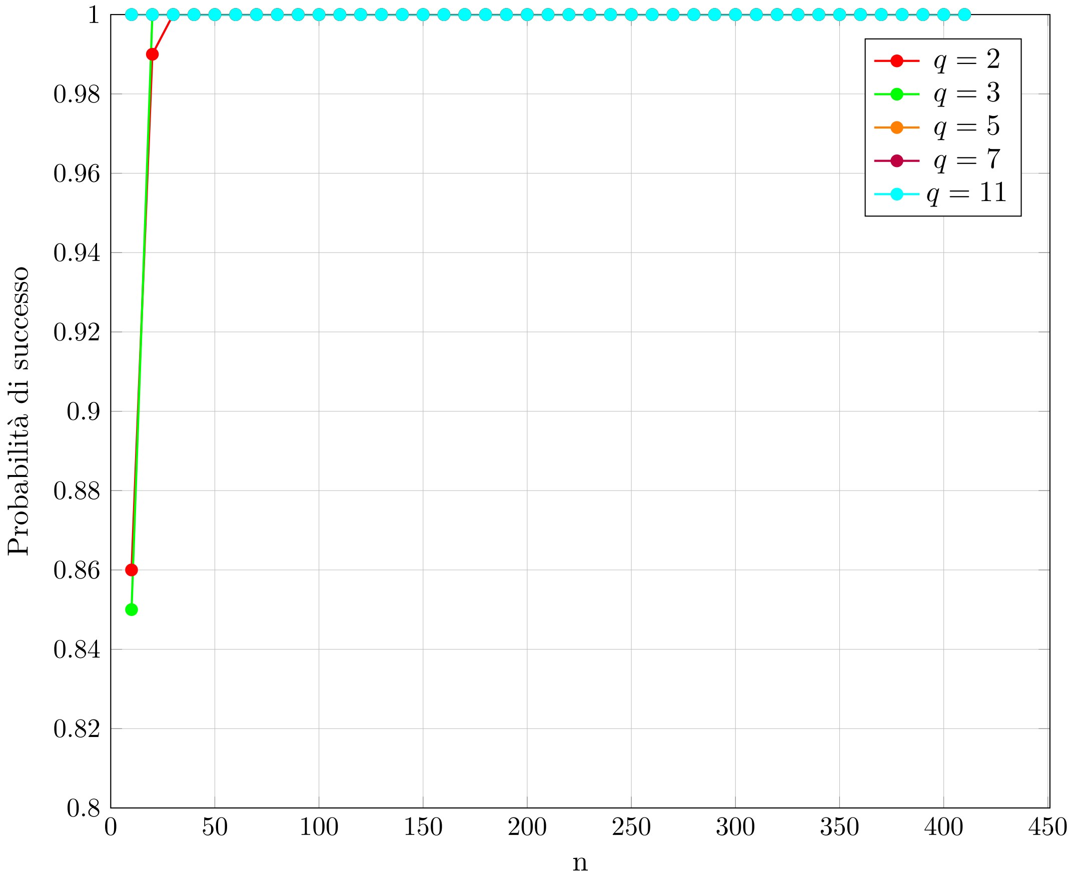

Figura 4.13: Probabilità di successo dell'algoritmo decisionale

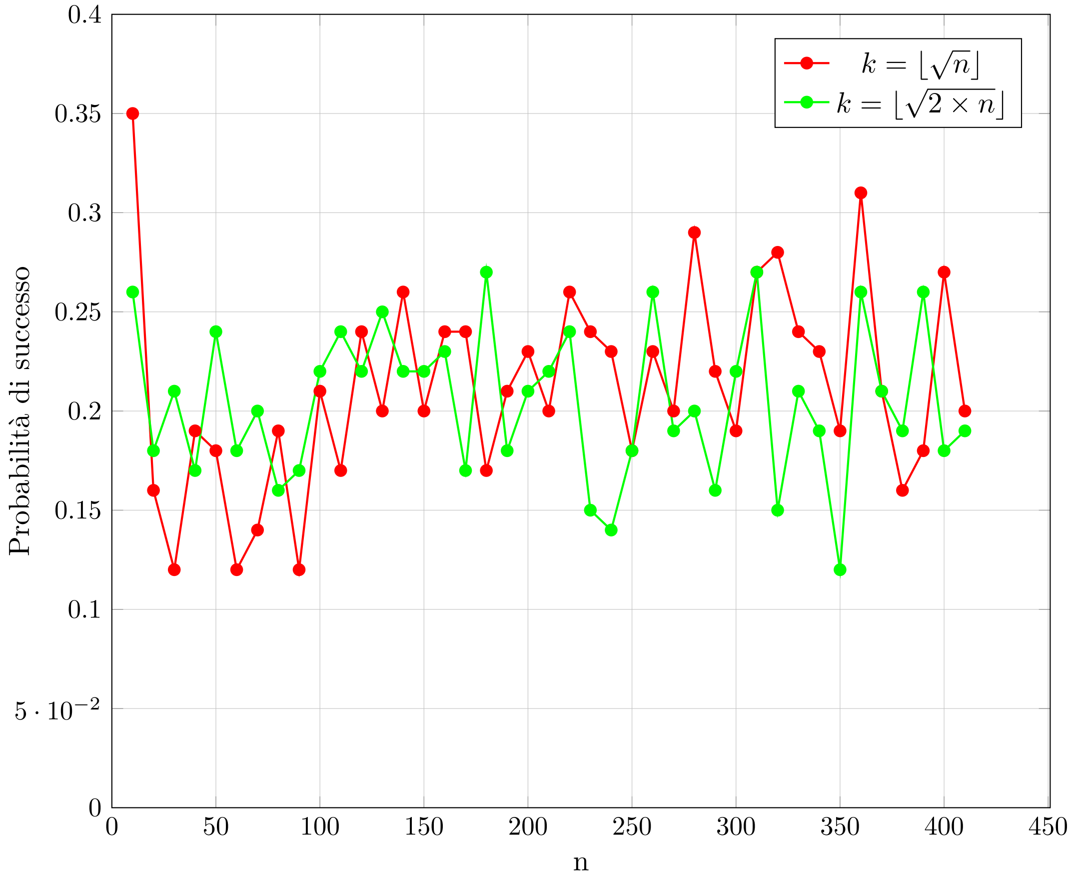

Figura 4.14: Probabilità di successo dell'algoritmo di ricerca

se il rapporto è minore, e quindi la probabilità di trovarne una a rango pieno sarà minore, e invece per entrambe le curve rimare più o meno allo stesso livello.

Inoltre sono stati rappresentati graficamente le probabilità di errore nel caclolo di $\tilde{\mathbf{P}}$ al variare di n, con due grafici per i differenti k, in Figura 4.15. Questa probabilità è piuttosto bassa e rappresenta la probabilità di trovare una permutazione valida per le sotto matrici $\tilde{\mathbf{G}}$ e $\tilde{\mathbf{G}}'$, ma che non è una sotto permutazione di $\mathbf{P}$.

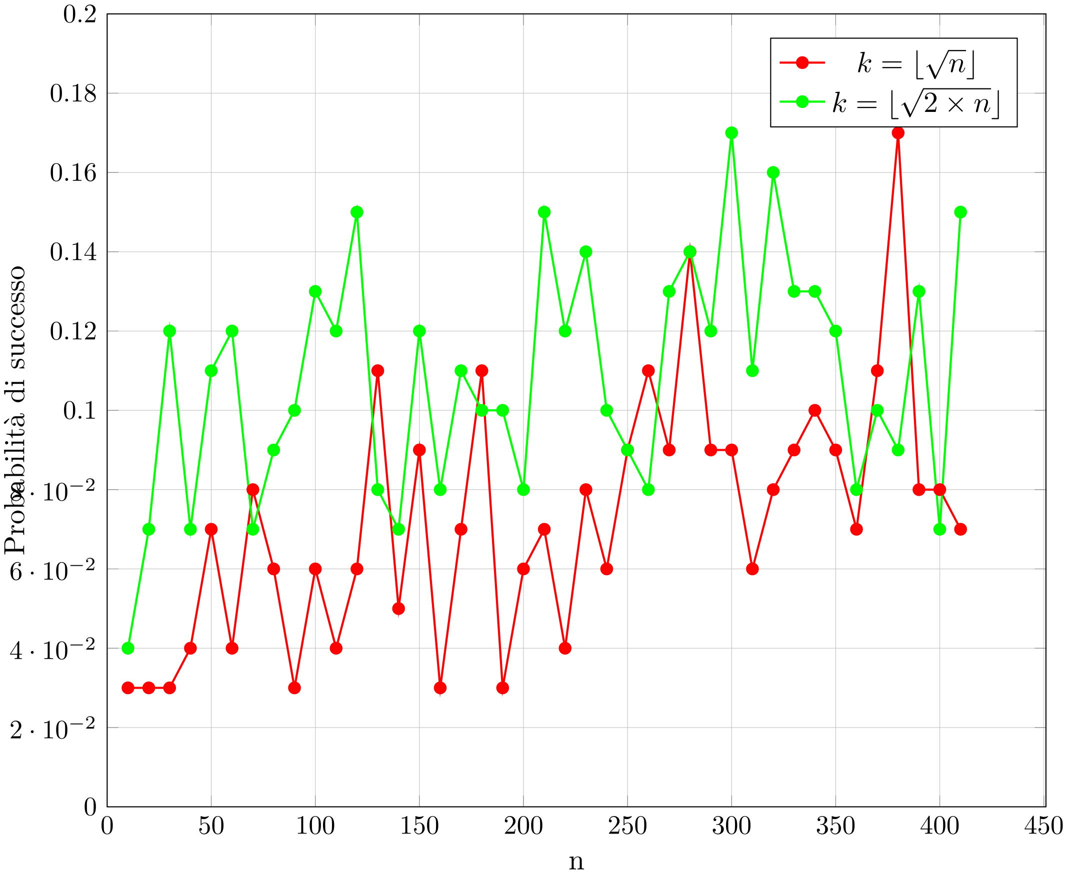

Figura 4.15: Probabilità di errore nel calcolo di $\tilde{P}$ dell'algoritmo di ricerca

Capitolo 5

Conclusioni e sviluppi futuri

Il lavoro svolto cerca di risolvere un sottoinsieme delle istanze del permutation equivalence problem considerate difficili, cioè quelle istanze con codici debolmente auto-duali, con un approccio basato sul calcolo dello square code.

Tramite gli esperimenti effettuati sull'hulla dello square code di codici debolmente auto-duali, è possibile osservare che la probabilità che l'hull abbia dimensione minore dell'hull del codice di partenza è molto alta; negli esperimenti non è mai capitato che l'hull dello square code avesse dimensione maggiore dell'hull del codice. Inoltre per $q = 2$ l'hull dello square code è triviale con una buona probabilità per molti valori di n, mentre per gli altri valori di q l'andamento della dimensione dell'hull dello square code è sempre confrontabile con il caso aleatorio ed è quindi triviale con una probabilità maggiore all'aumentare di q. É possibile notare che in tutti i casi in cui l'hull dello square code è triviale è applicabile la trasformazione al problema dell'isomorfismo dei grafi, che è risolvibile in un tempo quasi polinomiale. Bisogna però considerare anche il limite dato dalla Condizione 2.15 per il calcolo dello square code, che riduce l'insieme dei codici debolmente auto-duali sui quali è possibile applicare questo approccio a quelli con $k < \sqrt{2n}$.
Date le buone probabilità di avere hull dello square code triviale, il numero di istanze risolvibili in tempo quasi polinomiale con questo approccio è comuque piuttosto alto e inoltre anche nel caso in cui l'hull non sia triviale il problema da risolvere risulta meno complesso, dato che la dimensione dell'hull è diminuita.

Gli esperimenti effettuati sull'algoritmo decisionale permettono di concludere che questo approccio ha probabilità di successo molto alte, che tendono ad 1 all'aumentare di n, rispettando ovviamente la condizione di calcolo dello square code.

Il lavoro svolto sull'algoritmo di ricerca invece è stato più contenuto visto che è stato testato solo un algoritmo che risolve il problema per codici già con hull triviale. Questo algorimto, che lavora su un sotto problema di dimensione k, trova una soluzione con buona probabilità.

In conclusione, quindi, quando si realizza uno schema crittografico basato su PEP bisogna tenere conto che non tutti i codici debolmente auto-duali sono sicuri e utilizzabili, perché alcuni di questi possono essere attaccati in tempo quasi polinomiale con l'approccio presentato.

Sicuramente una possibile espansione di questo lavoro è legata all'algoritmo di ricerca, per il quale si può testare la probabilità di successo applicata a codici debolmente auto-duali sui quali viene effettuato lo square code.

Infine è necessario precisare che il lavoro è stato svolto solo sul permutation equivalence problem; si potrebbe quindi provare ad estendere questo approccio al più generale problema di equivalenza lineare.

Bibliografia

[1] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[2] National Institute of Standards and Technology. Nist asks public for help to future-proof electronic information. `https://www.nist.gov/news-events/news/2016/12/nist-asks-public-help-future-proof-electronic-information`, December 2016. Accessed: 10 October 2024.

[3] National Institute of Standards and Technology. Post-quantum cryptography standardization. `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`, January 2017. Updated: 25 September 2024, Accessed: 10 October 2024.

[4] National Institute of Standards and Technology. Nist announces first four quantum-resistant cryptographic algorithms. `https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms`, July 2022. Accessed: 10 October 2024.

[5] National Institute of Standards and Technology. Selected algorithms for post-quantum cryptography. `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`, July 2022. Accessed: 10 October 2024.

[6] National Institute of Standards and Technology. Round 4 submissions for post-quantum cryptography. `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`, July 2022. Accessed: 10 October 2024.

[7] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. Less-fm: Fine-tuning signatures from the code equivalence problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12841 LNCS:23 – 43, 2021.

[8] Jean-Francois Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. Cryptology ePrint Archive, Paper 2020/594, 2020. `https://eprint.iacr.org/2020/594`.

[9] Alessandro Barenghi, Jean-Francois Biasse, Tran Ngo, Edoardo Persichetti, and Paolo Santini. Advanced signature functionalities from the code equivalence problem. Cryptology ePrint Archive, Paper 2022/710, 2022. `https://eprint.iacr.org/2022/710`.

[10] Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. Cryptology ePrint Archive, Paper 2023/1533, 2023. `https://eprint.iacr.org/2023/1533`.

[11] Edoardo Persichetti and Paolo Santini. A new formulation of the linear equivalence problem and shorter LESS signatures. Cryptology ePrint Archive, Paper 2023/847, 2023. `https://eprint.iacr.org/2023/847`.

[12] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[13] Raymond Hill. *A first course in coding theory*. Oxford University Press, 1986.

[14] Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1998.

[15] Nicolas Sendrier. On the dimension of the hull. *SIAM Journal on Discrete Mathematics*, 10(2):282 – 293, 1997.

[16] Michael Artin. *Algebra*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[17] Hideki Imai, editor. *Essentials of Error-Control Coding Techniques*. Academic Press, Yokohama National University, Kanagawa, Japan, 1990.

[18] Jian Ren and Tongtong Li. Graph isomorphism—characterization and efficient algorithms. *High-Confidence Computing*, page 100224, 2024.

[19] Xi Li and Hanwu Chen. The quantum algorithm for graph isomorphism problem. *arXiv preprint arXiv:1901.06530*, 2019.

[20] Magali Bardet, Ayoub Otmani, and Mohamed Saeed-Taha. Permutation code equivalence is not harder than graph isomorphism when hulls are trivial. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2464–2468, 2019.

[21] James L. Massey. Linear codes with complementary duals. *Discrete Mathematics*, 106-107:337–342, 1992.

[22] E. Petrank and R.M. Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43(5):1602–1604, 1997.