Università degli Studi di Padova

Corso di Laurea Magistrale in Ingegneria Informatica

Quadratic Sieve
e
Metodo di Lanczos a Blocchi
nella
Fattorizzazione di Numeri Interi

Relatore: Prof. Alberto Tonolo
Studente: Nicola Zago
Matricola: 601487
Anno Accademico: 2009/2010

Indice

Capitolo 1

Introduzione

La moltiplicazione è un'operazione matematica di base, talmente semplice da essere insegnata addirittura alle Scuole Elementari come uno dei primi argomenti di Matematica. Certo, può essere complicata a piacere utilizzando FFT per eseguirla in modo efficiente per poter moltiplicare tra loro numeri di miliardi di cifre in pochi minuti, ma anche l'algoritmo tradizionale quadratico può moltiplicare tra loro numeri di centinaia di migliaia di cifre in pochi secondi. Raramente si pensa invece a quanto sia difficile l'operazione opposta: la fattorizzazione.

Se vogliamo recuperare i numeri precedentemente coinvolti nella moltiplicazione, difficilmente può venir in mente un metodo diverso dall'algoritmo banale che tenta la divisione per tutti i numeri primi minori della radice del numero n che si ha da fattorizzare. In realtà esistono metodi molto più sofisticati per svolgere questo compito, ma è comunque considerato un problema difficile, tanto che la resistenza di algoritmi di crittografia come RSA si basa proprio sulla difficoltà di recuperate due numeri moltiplicati tra di loro che fungono da chiave. In particolare si tratta di due numeri primi grandi circa come la radice quadrata del numero finale, in modo che siano effettivamente necessarie $\sqrt{n}$ operazioni col metodo banale. Inoltre i due fattori soddisfano alcune altre proprietà per rendere effettivamente difficile la fattorizzazione anche usando altri approcci intelligenti.

Da sempre in Teoria dei Numeri si studiano la struttura e le proprietà dei numeri e nonostante si siano fatte molte scoperte in ambiti in qualche modo legati alla fattorizzazione, questa non è mai stata presa veramente sul serio, in quanto si riteneva fosse una questione di 'far conti' e che fosse tutto sommato fine a sé stessa. Da quando è diventata una possibile arma per attaccare la sicurezza dei sistemi informatici è passata in primo piano negli studi di Teoria dei Numeri e sicurezza informatica.

Tornando alla soluzione banale del problema, si considerino i seguenti esempi numerici. Per un numero di 18 cifre sono necessarie fino a un miliardo di divisioni (maggiorandone il numero con $\sqrt{n}$), diciamo circa un secondo in un computer moderno (facendo una stima molto ottimistica). Un numero di 27 cifre richiede già 31.000 miliardi di operazioni, circa 9 ore di calcolo. Per un numero di 36 cifre servono un miliardo di miliardi di operazioni, più di 31 anni su una singola macchina. In [2] si stima che il numero di operazioni elementari eseguite da tutte le macchine costruite dall'uomo nell'ultimo secolo ammonti a qualcosa come 10^{24}. Queste sono sufficienti a fattorizzare *un singolo*

numero di 50 cifre. Quindi è difficile immaginare di andare altro a numeri di 30-40 cifre con questo metodo. Eppure le attuali chiavi RSA hanno 1024 bit per usi standard e 2048 bit per usi che richiedono sicurezza a lunga scadenza, cioè rispettivamente 309 e 617 cifre decimali.

Questo perchè da quando la fattorizzazione ha cominciato a essere studiata, sono stati creati algoritmi sempre più sofisticati, in grado si sfruttare tecniche probabilistiche e la grande potenza di calcolo dei moderni computer. Negli anni '70 sono nati algoritmi con complessità inferiore all'algoritmo banale, ad esempio il metodo Rho di Pollard, ma richiedevano ancora un tempo esponenziale. Quando nel 1977 nacque RSA, Gardner propose una sfida che consisteva nel rompere una chiave di 129 cifre decimali, RSA-129, stimando sarebbero stati necessari 40 quadrilioni d'anni con le tecniche e le tecnologie dell'epoca. Nel corso degli anni '80 fu inventato un metodo di fattorizzazione sub-esponenziale (ma non ancora polinomiale), il *Quadratic Sieve*. Questo fu via via perfezionato, permettendo nel 1994 di fattorizzare RSA-129 in soli 8 mesi, distribuendo il calcolo in Internet, usando circa 600 computer (stimando 5000 anni-MIPS di calcolo[1]). Vedremo nel seguito che gli algoritmi di fattorizzazione moderni sono costituiti da due fasi, la prima in cui si trovano delle particolari relazioni e una seconda dove bisogna risolvere una matrice per ricombinare le relazioni in modo opportuno. La fattorizzazione di RSA-129 richiese la ricerca di circa 524.000 relazioni e la soluzione di una matrice 180.000×180.000, affrontata con l'algoritmo di eliminazione Gaussiana opportunamente modificato ed eseguito su un super-computer.

Quello fu il canto del cigno sia per il QS sia per l'uso dell'algoritmo di Gauss in matrici di fattorizzazione, tutti i successivi record sono stati compiuti con il *General Number Field Sieve*, anch'esso con complessità sub-esponenziale, ma più efficiente del Quadratic Sieve per numeri con più di 120 cifre, e con algoritmi per la soluzione di matrice molto più efficienti e distribuibili, tanto da non richiedere più dei super-computer per essere eseguiti. Questi algoritmi hanno permesso di fattorizzare RSA-512 nel 1999 (155 cifre decimali) e RSA-768 nel 2009 (232 cifre decimali), in meno di un anno nel primo caso e in più di due anni nel secondo caso (8000 anni-MIPS e circa 2000 anni di processori AMD64 nel secondo, circa 3 ordini di grandezza più del precedente). In particolare con programmi reperibili gratuitamente in Internet è attualmente possibile rompere un numero di 512 bit in circa un paio di mesi su un PC di fascia medio alta. Questo perchè negli ultimi anni oltre a processori multi-core, si possono sfruttare anche le GPU (Graphical Processing Unit), che sono dotate di un parallelismo molto più spinto delle CPU (si parla di throughput dell'ordine del Teraflop[2] al secondo, sfruttando i 48-96 core di cui sono dotate).

Le statistiche dell'ultimo record di fattorizzazione sono impressionanti. RSA-768 in particolare ha richiesto la raccolta di oltre 64 miliardi di relazioni e la soluzione di una matrice $192.796.550 \times 192.795.550$. Per avere un'idea del raffinamento algoritmico si pensi che questa matrice può essere memorizzata in un centinaio di Gb negli algoritmi moderni, mentre avrebbe richiesto mezzo milione di Gb con l'algoritmo di Gauss. RSA-768 ha richiesto 10.000 volte più risorse di calcolo di RSA-512, mentre la differenza tra RSA-1024 e RSA-768 è un quarto della precedente. Si stima che entro i prossimi 10

[1] Un anno-MIPS è il numero di operazioni eseguite in un anno facendone un milione al secondo

[2] 10^{12} floating point operation.

anni anche le chiavi di 1024 bit diventeranno non sicure usando gli algoritmi attuali e sfruttando solo le innovazioni tecnologiche. Anche se nulla vieta che si trovino metodi più efficenti che riducano questa stima. Intanto i laboratori RSA si preparano a ritirare le chiavi da 1024 bit, passando a lunghezza minima di 2048 bit per il 2014.

In realtà numeri di oltre 1000 bit sono già stati fattorizzati tramite lo *Special Number Field Sieve*, in particolare nel 2007 è stato fattorizzato M1039 (il numero di Mersenne $2^{1039} - 1$). Ciononostante le chiavi di 1024 bit di RSA sono ancora sicure in quanto lo SNFS non è *general purpose*, ma può fattorizzare solo numeri di una particolare forma. Questo algoritmo comunque da una buona idea di quali sono i numeri attaccabili in un futuro prossimo con lo GNFS.

Come sviluppi futuri, sono già noti algoritmi per computer quantistici, che dovrebbero far finalmente diventare polinomiale la fattorizzazione. Purtroppo la costruzione di tali macchine è ancora un problema aperto e forse ancora per molto tempo la ricerca proseguirà sui tradizionali computer binari. Comunque è molto interessante vedere come l'evoluzione tecnologica ed algoritmica riescano a risolvere problemi prima ritenuti largamente inattaccabili. In particolare questo problema mostra anche come gli algoritmi sono 'figli' del tempo in cui sono nati: gli attuali algoritmi di fattorizzazione sono nati in modo da essere facilmente distribuibili su più macchine, usare operazioni su interi che sono più performanti sulle attuali macchine e così via.

Al di là dell'utilità in campo di sicurezza, ritengo che la fattorizzazione sia un problema veramente stimolante, dal momento che di per sé è un problema che può essere capito anche da un bambino delle Elementari, ma la sua soluzione richiede una profonda comprensione della struttura dei numeri interi e ha strettissimi legati con i numeri primi. Dal punto di vista computazionale, istanze relativamente piccole (512 bit) mettono in seria difficoltà macchine molto potenti. Per intenderci la moltiplicazione di due numeri di 256 bit richiede meno di un microsecondo anche usando un vecchio PC e algoritmi banali, ma l'operazione opposta richiede mesi usando macchine che eseguono miliardi di operazioni al secondo e algoritmi la cui comprensione richiede un percorso di Laurea Magistrale.

Ho approfittato di questa Tesi di Laurea per approfondire la conoscenza di questo affascinante campo di ricerca, in particolare proponendo un'overview generale dell'evoluzione del Quadratic Sieve, dalla versione base alla versione che ha permesso la rottura di RSA-129. Inoltre propongo i raffinamenti dell'algoritmo di Gauss utilizzati nelle matrici di fattorizzazione e i metodi più sofisticati usati negli attuali record di fattorizzazione. Ho realizzato un'implementazione di tutti questi metodi, eseguendo un confronto diretto di tutte le varie caratteristiche proposte in letteratura. Ritengo questo confronto sia significativo, in quanto i vari raffinamenti sono stati proposti nel corso di 15 anni e quindi, comparendo in tempi diversi, hanno dato origine a confronti che oltre al potenziamento dovuto alla nuova tecnica risentivano anche della maggior potenza di calcolo disponibile. Le tecniche più datate venivano date poi per scontate e non esiste in letteratura un lavoro pratico che confronti in modo diretto tutte le tecniche eseguite sullo stesso hardware.

Per la lettura sono richieste alcune conoscenze di Algebra Modulare, richiamate nel Capitolo 2, ma in generale l'algoritmo trattato è interessante proprio perchè concettualmente abbastanza semplice. La tesi vuole fornire le conoscenze necessarie per poter realizzare un'implementazione efficiente del Quadratic Sieve e delle sue evoluzioni. La

conoscenza di questo algoritmo può essere inoltre vista come propedeutica per lo studio del più sofisticato Number Field Sieve. Spero la lettura sia sufficientemente chiara e completa e possa essere utile a qualcuno che si sia avvicinato a questo interessante argomento. Per finire vorrei ringraziare il mio relatore, professor Alberto Tonolo, e il professor Alessandro Languasco per l'aiuto e per i consigli dati in fase di studio e realizzazione dei vari algoritmi proposti.

Capitolo 2

Richiami Teorici

Prima di iniziare la trattazione è utile richiamare alcuni risultati di algebra modulare e alcuni semplici algoritmi di fattorizzazione. Inoltre introduciamo il concetto di numero y-smooth, fondamentale nei moderni algoritmi di fattorizzazione.

2.1 Algebra Modulare

Questa sezione richiama alcuni concetti di Algebra Modulare, senza pretese di completezza. Si indirizzano i lettori interessati a testi specifici, come ad esempio [3].

2.1.1 Congruenze

Innanzitutto si ricordano i seguenti teoremi e definizioni.

Definizione 2.1. *Si dice che c è un divisore di a se $a = bc$ e si indica con $c|a$ (a, b e $c \in \mathbb{Z}$).*

Teorema 2.1. *Sia $d \in \mathbb{Z}$, $d > 0$. Per ogni $x \in \mathbb{Z}$ esiste un unico* resto $r \in \mathbb{N}$ *tale che*

$$x = qd + r$$

con $q \in \mathbb{Z}$ e $0 \leq r < d$.

Il resto della divisione di x per d si indica anche con $[x]_d$.

Definizione 2.2. *Siano a, b e $c \in \mathbb{Z}$. Allora a e b sono* congruenti modulo c *se c divide $b - a$ e si indica con $a \equiv b \pmod{c}$.*

Proposizione 2.1. *Si supponga che $x_1 \equiv x_2 \pmod{d}$ e $y_1 \equiv y_2 \pmod{d}$, $x_1, x_2, y_1, y_2 \in \mathbb{Z}$. Allora:*

1. $x_1 + x_2 = y_1 + y_2 \pmod{d}$
2. $x_1 x_2 = y_1 y_2 \pmod{d}$

2.1.2 Massimo Comun Divisore

Sia $\mathrm{div}(n) = \{d \in \mathbb{N} : d|n\}$ l'insieme dei divisori naturali di $n \in \mathbb{Z}$. Con il lemma di Euclide si dimostra che dati m e $n \in \mathbb{Z}$ esiste un unico $d \in \mathbb{N}$ tale che

$$\mathrm{div}(m) \cap \mathrm{div}(n) = \mathrm{div}(d).$$

Definizione 2.3. *L'unico $d \in \mathbb{N}$ che soddisfa $div(d) = div(m) \cap div(n)$ è detto Massimo Comun Divisore (greatest common divisor) di m ed n, e si indica con $gcd(m, n)$.*

Definizione 2.4. *I numeri a, $b \in \mathbb{Z}$ si dicono* relativamente primi *se*

$$gcd(a, b) = 1.$$

Dal momento che il *gcd* ha un ruolo fondamentale negli algoritmi di fattorizzazione, ricordiamo le proprietà che permettono di calcolarlo:

Proposizione 2.2. *Siano m, $n \in \mathbb{Z}$, allora:*

1. *$gcd(m, 0) = m$ se $m \in \mathbb{N}$.*
2. *$gcd(m, n) = gcd(m - qn, n)$ per ogni $q \in \mathbb{Z}$.*

Una proprietà importante di $\gcd(m, n)$ è che si può esprimere come combinazione lineare di m ed n a coefficienti interi.

Lemma 2.1. *Siano m, $n \in \mathbb{Z}$. Allora esistono λ, $\mu \in \mathbb{Z}$ tali che:*

$$\lambda m + \mu n = gcd(m, n)$$

Il massimo comun divisore e i coefficienti della combinazione lineare possono essere calcolati in contemporanea tramite l'*algoritmo di Euclide esteso* (*extended Euclidean algorithm*).

2.1.3 Teorema Cinese del Resto

Anche se non interviene direttamente negli algoritmi di fattorizzazione, è utile ricordare il Teorema Cinese del Resto (*Chinese Remainder Theorem*), in quanto permettere di fare molte considerazioni importanti in fase di analisi di questi.

Teorema 2.2 (Chinese Remainder Theorem)**.** *Sia $N = n_1 \cdots n_t$, con $n_1, \ldots, n_t \in \mathbb{Z}\backslash\{0\}$ e $gcd(n_i, n_j) = 1$ per $i \neq j$. Si consideri il sistema di congruenze:*

$$\begin{aligned} X &\equiv a_1 \pmod{n_1}, \\ X &\equiv a_2 \pmod{n_2}, \\ &\vdots \\ X &\equiv a_t \pmod{n_t}, \end{aligned} \tag{2.1}$$

per $a_1, a_2, \ldots, a_t \in \mathbb{Z}$. Allora:

1. *(2.1) ha una soluzione $X = x \in \mathbb{Z}$.*
2. *Se x, $y \in \mathbb{Z}$ sono soluzioni di (2.1) allora $x \equiv y \pmod{N}$.*
3. *Se x è di (2.1) e $y \equiv X \pmod{N}$ allora anche y è una soluzione di (2.1).*

2.1.4 Residui quadratici e radici quadrate

Un ultimo argomento dell'algebra modulare molto importante per gli algoritmi di fattorizzazione riguarda i residui quadratici.

Definizione 2.5. *Sia p un numero primo. Se $p \nmid a$ allora a è detto* residuo quadratico modulo p *se esiste x tale che $a \equiv x^2 \pmod p$. Altrimenti a è un* non-residuo quadratico modulo p. *Se $p|a$, a non è considerato né un residuo quadratico né un non-residuo quadratico. Il* simbolo di Legendre *viene definito come:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{se } p|a \\ 1 & \text{se } s \text{ è residuo quadratico modulo } p \\ -1 & \text{se } s \text{ è non-residuo quadratico modulo } p \end{cases}$$

Si ricorda che se p è un numero primo dispari metà dei numeri $1, 2, \ldots, p-1$ sono residui quadratici modulo p e l'altra metà sono non-residui quadratici modulo p. Un metodo efficiente per calcolare il valore del simbolo di Legendre è dovuto a Eulero.

Teorema 2.3 (Test di Eulero)**.** *Sia p primo dispari e sia a intero tale che $p \nmid a$; allora:*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} (mod\ p).$$

Una volta che si è individuato che a è un residuo quadratico modulo p, può essere interessante conoscere un x che permette di ottenere l'equivalenza $x^2 \equiv a \pmod p$. Nel caso in cui $(\frac{a}{p}) = 1$ e $p \equiv 3 \pmod 4$ (e quindi $p = 4k+3$), il test di Eulero dice che $a^t \equiv 1 \pmod p$, con $t = (p-1)/2$ (ovvero $t = 2k+1$). Allora $a^{t+1} \equiv a \pmod p$ e $t+1$ è pari, quindi in questo caso si può porre $x \equiv a^{(t+1)/2} \pmod p$.
È possibile generalizzare questo ragionamento al caso $p \equiv 1 \pmod 4$, ottenendo così un algoritmo per calcolare le radici quadrate modulare per i residui quadratici. Si rimandano i lettori al [2] per una descrizione completa di due algoritmi del genere: l'algoritmo di Tonelli e l'algoritmo di Cipolla.

Come si vedrà in seguito, i residui quadratici e le radici quadrate modulari intervengono in più modi nell'ambito degli algoritmi di fattorizzazione.

2.2 Algoritmi di fattorizzazione

In questa sezione si tratteranno alcuni semplici metodi di fattorizzazione esponenziali. Questi intervengono anche negli algoritmi più sofisticati; è infatti spesso necessario completare la fattorizzazione di numeri relativamente piccoli, ricorrendo a metodi semplici che si rivelano comunque efficaci per input limitati. Prima di intraprendere la fattorizzazione di un numero è opportuno verificare che questo sia effettivamente composto, dal momento che il processo di fattorizzazione è computazionalmente molto pesante.
Esistono algoritmi deterministici per stabilire la primalità di un numero, ad esempio l'AKS test, proposto da Agrawal, Kayal e Saxema nel 2002, molto importante soprattutto perchè mostra con la sua complessità quartica che la determinazione della primalità di un numero è un problema polinomiale.

In genere comunque si preferisce usare dei test probabilistici, che forniscono probabilità molto elevate di individuare numeri composti e tempi di calcolo molto efficienti, richiedendo inoltre minor complessità di implementazione. Dal momento che non danno garanzia di primalità assoluta sono detti test di pseudo-primalità. Nel seguito si richiameranno alcuni di questi test; per una trattazione più completa di test di primalità si rimanda a [2].

2.2.1 Test di pseudo-primalità

Test di Fermat

Il test di pseudo-primalità probabilmente più noto e semplice si rifà al Piccolo Teorema di Fermat:

Teorema 2.4 (Piccolo Teorema di Fermat)**.** *Sia p primo e a tale che $gcd(p, a) = 1$. Allora*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Sia ora n un numero di cui si vuole determinare la primalità (*candidato*); il test di Fermat prevede di calcolare $a^{n-1} \equiv 1 \pmod{n}$ per un qualche a (*base*) tale che $\gcd(n, a) = 1$. Se n non supera il test è composto, altrimenti si dice essere pseudo-primo in base a. Ci sono numeri composti che superano il test per ogni base a relativamente prima con n e si dicono *numeri di Carmichael*; comunque questi sono rari.

Computazionalmente parlando il test è molto leggero, in quanto consiste in un'esponenziazione modulo n, quindi è adatto per riconoscere con certezza un composto senza grande dispendio di risorse.

Pseudo-primalità forte

Per rendere più potente il test di Fermat è sufficiente conoscere il seguente lemma:

Lemma 2.2. *Sia p primo e $x \in \mathbb{Z}$. Se $x^2 \equiv 1 \pmod{p}$ allora $x \equiv \pm 1 \pmod{p}$.*

Grazie a questo lemma si può pensare, una volta trovato un numero pseudo-primo in base a, di continuare la sua analisi. Per esempio, si supponga di testare $n = 341$ per la base $a = 2$. Il candidato è pseudo primo nella base prescelta, in quanto $a^{340} \equiv 1 \pmod{341}$. Il lemma precedente afferma che se n è primo allora $a^{170} \equiv \pm 1 \pmod{341}$. Una volta che si verifica che $a^{170} \equiv 1 \pmod{341}$, sempre per il lemma precedente se 341 fosse primo dovrebbe valere anche $a^{85} \equiv \pm 1 \pmod{341}$; invece $a^{85} \equiv 32 \pmod{341}$ e permette di concludere che n non è primo.

È quindi possibile formulare la seguente proposizione.

Proposizione 2.3. *Sia p un numero primo dispari e sia $p-1 = 2^k q$ con q numero intero dispari. Se $a \in \mathbb{Z}$ e $gcd(a, p) = 1$ allora o $a^q \equiv 1 \pmod{p}$ oppure esiste $i = 0, \ldots, k-1$ tale che $a^{2^i q} \equiv -1 \pmod{p}$.*

Un numero che supera il test suggerito da questa proposizione per la base a si dice *pseudo-primo forte* per la base a (*strong pseudoprime*). La principale importanza di questo test è dovuta al fatto che un numero composto n può essere pseudoprimo forte

per al più $\frac{1}{4}$ delle basi comprese tra 1 e $n-1$ (Teorema di Rabin); ciò significa che un composto ha probabilità al più $\frac{1}{4}$ di superare il test per una base casuale e al più $\frac{1}{4^j}$ di superarlo per j basi casuali. Per $j > 30$ si ha una probabilità così bassa che n sia composto che è considerato primo per tutti gli scopi pratici, con un costo computazionale molto minore dei test di probabilità deterministici.

2.2.2 Trial division

La divisione per tentativi è l'algoritmo banale nel campo della fattorizzazione. Esso prevede di provare la divisione del candidato n per tutti i numeri primi minori della sua radice quadrata. Se questa operazione non ritorna fattori allora n è primo (è sufficiente eseguire il controllo fino alla radice quadrata in quanto se avesse un fattore maggiore di questa necessariamente ne avrebbe anche uno di minore).
L'algoritmo richiede nel caso peggiore $\sqrt{n}$ divisioni (per essere precisi per il Teorema dei Numeri Primi riportato nella sezione A.8 sono necessarie $\sqrt{n}/\frac{1}{2}\log n$ divisioni) e ricordando che le dimensioni del problema sono $\log(n)$, la complessità di questo metodo è esponenziale, infatti:

$$\sqrt{n} = e^{\frac{\log(n)}{2}}.$$

Ciononostante data la velocità degli attuali calcolatori è molto efficace per portare a termine quasi istantaneamente fattorizzazioni di numeri dell'ordine di 15 cifre.

2.2.3 Metodo di Fermat

Dato un numero composto $n = ab$, è possibile sempre scriverlo come differenza di quadrati:

$$n = ab = \left[\frac{1}{2}(a+b)\right]^2 - \left[\frac{1}{2}|a-b|\right]^2 = x^2 - y^2.$$

In particolare se n è dispari x e y sono interi. Se si dispone di una relazione $n = x^2 - y^2$ è possibile trovare immediatamente una fattorizzazione di $n = (x+y)(x-y)$, che è non banale nel caso che $x - y > 1$.

Se consideriamo $n = 2491$, il primo quadrato maggiore di n è $2500 = 50^2$ e la differenza con n è $9 = 3^2$. Questo conduce alla fattorizzazione non banale di $n = (50+3)(50-3) = 53 \cdot 47$. Per numeri con fattori vicini tra loro si può utilizzare una strategia di questo tipo per completare la fattorizzazione. Il metodo di Fermat prevede di prendere x nella sequenza $\lceil\sqrt{n}\rceil$, $\lceil\sqrt{n}\rceil + 1$, ... e verificare se $x^2 - n$ è un quadrato. In tal caso $y^2 = x^2 - n$ ed è possibile fattorizzare n come $(x+y)(x-y)$.
Se n è dispari e composto il procedimento termina con una fattorizzazione non banale; l'algoritmo può terminare la sua ricerca per $x = \lfloor(n+9)/6\rfloor$ e nel caso non abbia trovato fattori significa che n è primo. Il caso peggiore per cui si arriva ad $x = (n+9)/6$ è quando $n = 3p$ in quanto 3 è il minor numero dispari e dato che non ci sono altri fattori intermedi sarà il primo ad essere trovato. In questo caso $a = p$ e $b = 3$ e quindi $x = \frac{p+3}{2} = \frac{n+9}{6}$.

La complessità nel worst case è addirittura $O(n)$, ma il caso peggiore di questo metodo è il migliore della trial division. Si potrebbe quindi pensare di combinarli per ottenere un algoritmo più efficiente. Si è scelto di riportare questo algoritmo seppur così

inefficiente in quanto come si avrà modo di vedere l'idea di scomporre in una differenza di quadrati il candidato alla fattorizzazione è alla base dei migliori algoritmi.

2.2.4 Metodo Rho di Pollard

Si riporta infine l'algoritmo del metodo Rho di Pollard, in quanto ha un valore atteso di esecuzione $O(\sqrt[4]{n})$ e un utilizzo di spazio costante. Per questo può essere considerato un buon algoritmo per eseguire sotto-task di fattorizzazioni negli algoritmi di fattorizzazione più potenti. Si pensi che se opportunamente implementato può fattorizzare numeri di 20 cifre in pochi decimi di secondo.

Esso si basa sul seguente principio: sia $N = pq$ il numero da fattorizzare, con p il suo più piccolo fattore. Dati due numeri a e b, $0 \leq a, b < N$, tali che $a \equiv b \pmod{p}$, $a \neq b$, allora $\gcd(a - b, N)$ è un fattore non banale di N. Se si genera una sequenza di numeri casuali $X_1, X_2, \ldots,$ con $0 \leq X_i < N$, per il paradosso del compleanno[1] ci si aspetta che dopo circa $\sqrt{(\pi p)/2}$ assegnazioni alla variabile X si abbia una ripetizione di $X \bmod p$. Il metodo Rho di Pollard sfrutta questo principio specificando un modo che consente una ricerca lineare tra i valori della variabile X anzichè quadratico. In particolare ciò è permesso dal lemma:

Lemma 2.3. *Sia $f : M \mapsto M$ una funzione, con M insieme finito. Si scelga $x_0 \in M$ per generare la sequenza $x_0, x_1, \ldots$ ponendo $x_{i+1} = f(x_i)$. Esistono $i, j \in \mathbb{N}, i \neq j$ tali che $x_i = x_j$. Inoltre esiste $n > 0$ tale che $x_n = x_{2n}$.*

Ciò suggerisce di scegliere una funzione modulo N, ad esempio $f(x) = [x^2 + 1]_N$ e, fissati $x_0 = y_0$, calcolare ad ogni passo $x_i = f(x_{i-1})$ e $y_i = f(f(y_{i-1}))$, in modo che la sequenza $y_0, y_1, y_2, \ldots$ corrisponda alla sequenza $x_0, x_2, x_4, \ldots$ e possa essere usata nella verifica $\gcd(Y_i - X_i, N)$ per una ricerca dei fattori sulla solo dimensione i.

Per una trattazione completa del metodo si rimanda a [3] e [2].

2.3 Numeri y-smooth

Gli *smooth number* sono estremamente importanti nelle moderne tecniche di fattorizzazione.

Definizione 2.6. *Un numero intero positivo è detto y-smooth se tutti i suoi fattori primi non superano y.*

Si è preferito non tradurre il termine inglese *smooth* in quanto ogni traduzione possibile risulterebbe inadeguata. Nel seguito si illustreranno alcuni fatti di interesse sui numeri y-smooth, tratti da [2], [5] e [8].

[1] Se in una stanza sono presenti N persone, qual'è la probabilità che qualcuno sia nato lo stesso giorno? Si consideri il problema inverso, denotando con $P(N)$ la probabilità che nessuno dei presenti condivida il compleanno. Chiaramente $P(2) = \frac{364}{365}$, in quanto per il secondo individuo rimangono 364 giorni disponibili; così $P(3) = \frac{364}{365} \cdot \frac{363}{365}$ e in generale $P(N) = \frac{365 \cdot \cdots \cdot (365-N+1)}{365^N}$. La probabilità che due persone condividano il giorno di compleanno è quindi 1-$P(N)$. Modellando il sistema come un'estrazione con rimpiazzo da uno spazio di N oggetti il numero di estrazioni prima di avere una ripetizione è una variabile aleatoria che per N grandi ha valore atteso $\sqrt{\frac{\pi N}{2}}$.

Una funzione importante in questo campo è quella che conta quanti numeri y-smooth ci sono fino a un dato x. Questa si indica con $\psi(x, y)$; formalmente

$$\psi(x, y) = |\{1 \leq n \leq x : n \text{ è } y\text{-smooth}\}|.$$

Nel seguito si forniranno delle stime sul valore asintotico di questa funzione, in quanto necessarie in fase di analisi degli algoritmi trattati in seguito. Un primo risultato interessante e di facile dimostrazione è il seguente.

Lemma 2.4. *$lim_{x\to\infty} \frac{\psi(x,x^{\frac{1}{u}})}{x} = 1 - log(u)$ per $1 \leq u \leq 2, x > 0$.*

Dimostrazione. Si mostrerà il risultato per $u = 2$ e per ogni $u \in [1, 2]$ si può ragionare allo stesso modo.
Ogni $p > x^{\frac{1}{2}}$ ha esattamente $\lfloor \frac{x}{p} \rfloor$ multipli in $[1, x]$; questi non sono $x^{\frac{1}{2}}$-smooth e non possono essere contemporaneamente divisibili per altri primi maggiori di $x^{\frac{1}{2}}$. Infatti nessun numero minore o uguale a x è divisibile per due fattori maggiori di $x^{\frac{1}{2}}$. Quindi vale la seguente:

$$\psi(x, x^{\frac{1}{2}}) = \lfloor x \rfloor - \sum_{x^{\frac{1}{2}} < p \leq x} \left\lfloor \frac{x}{p} \right\rfloor. \tag{2.2}$$

Se si toglie il floor si commette un errore di al più $\pi(x)$ (di un'unità per ogni numero primo). Usando il Teorema dei Numeri Primi (A.8)

$$\pi(x) \sim \frac{x}{\log x}$$

e il risultato (si veda sezione A.3)

$$\sum_{p \leq x} \frac{1}{p} = \log \log x + C + O\left(\frac{1}{\log x}\right)$$

è possibile eseguire le seguenti semplificazioni:

$$\begin{aligned} \sum_{x^{\frac{1}{2}} < p \leq x} \frac{1}{p} &= \sum_{p \leq x} \frac{1}{p} - \sum_{p \leq x^{\frac{1}{2}}} \frac{1}{p} = \log \log x - \log \log x^{\frac{1}{2}} + O\left(\frac{1}{\log x^{\frac{1}{2}}}\right) \\ &= \log \left(\frac{\log x}{\frac{1}{2}\log x}\right) + O\left(\frac{1}{\log x}\right) = \log 2 + O\left(\frac{1}{\log x}\right). \end{aligned}$$

Quindi la 2.2 diventa

$$\psi(x, x^{\frac{1}{2}}) = (1 - \log 2)x + O\left(\frac{1}{\log x}\right),$$

che è facilmente generalizzabile a quanto enunciato. □

Questo lemma dice che con i numeri primi minori di $\sqrt{x}$ è possibile costruire più del 30% dei numeri minori di x, fatto tutt'altro che banale. Dickman negli anni trenta ha dimostrato più in generale che per ogni u positivo

$$\frac{\psi(x, x^{\frac{1}{u}})}{x} \sim \rho(u),$$

dove $\rho(u)$ è la funzione di Dickman-di Bruijn, calcolabile come soluzione dell'equazione differenziale

$$u\rho'(u) = -\rho(u-1)$$

con condizione iniziale $\rho(u) = 1$ in $[0, 1]$. La $\rho(u)$ è sempre positiva e al crescere di u va a zero come u^{-u}. In particolare in [9] si mostra che per $x \to \infty$, $u \to \infty$ e $x^{\frac{1}{u}} > (\log x)^{1+\epsilon}$, per ogni $\epsilon > 0$ si ha[2]:

$$\frac{\psi(x, x^{\frac{1}{u}})}{x} = u^{-(1+o(1))u}.$$

Un altro risultato interessante è il seguente.

Teorema 2.5. *Sia m_1, m_2, ... una sequenza di interi casuali scelti indipendentemente ed uniformemente in $[1, X]$. Sia N il minimo intero tale che nella sequenza $m_1, \ldots, m_N$ esista un insieme non vuoto di numeri il cui prodotto è un quadrato. Allora il valore atteso di N è $e^{\sqrt{(2+o(1))ln\ X ln\ ln\ X}}$. Lo stesso valore atteso vale se si vincola ogni m_i usato nel prodotto ad essere y-smooth con $y = e^{\sqrt{\frac{1}{2}ln\ X ln\ ln\ X}}$.*

In letteratura la funzione $e^{\sqrt{\ln X \ln\ln X}}$ si indica con $L(X)$. Il teorema afferma che da una sequenza casuale di numeri minori di X, è molto probabile, nel momento in cui si sono individuati $L(X)^{\sqrt{2}}$ numeri $L(X)^{\frac{1}{\sqrt{2}}}$-smooth, riuscire a trovare un quadrato. Nel prossimo capitolo vedremo come sia possibile usare questo risultato in un algoritmo per generare una sequenza di numeri y-smooth con cui generare una differenza di quadrati in un tempo sub-esponenziale.

[2] È possibile vedere il risultato anche come $\psi(x, z) = x \cdot u^{-u}$ con $z = x^{1/u}$ e $u = \frac{\log x}{\log z}$

Capitolo 3

Quadratic Sieve

Il Quadratic Sieve (QS) è uno dei principali algoritmi per la fattorizzazione di numeri grandi, in particolare è il più adatto per numeri al di sotto delle 120 cifre decimali. Oltre questo limite si ottengono migliori prestazioni con il Number Field Sieve. A differenza di quest'ultimo il Quadratic Sieve è molto più intuitivo e di più facile implementazione. È stato introdotto da Pomerance nel 1982 in [1], come evoluzione di alcuni algoritmi, in particolare del metodo delle frazioni continue. Nel corso del tempo il metodo originario è stato via via raffinato, aumentado di potenza. In seguito si dà una descrizione completa di quest'algoritmo e delle sue varianti, mostrando anche alcuni aspetti di cui tener conto per ottenere implementazioni efficienti. Ci si rifarà principalmente a [2], citando di volta in volta eventuali altre fonti.

3.1 Idee di base del QS

In questa sezione si cercherà di dare un'idea intuitiva degli elementi di base di questo algoritmo. Alcune parti saranno presentate in modo semplificato a favore della chiarezza, lasciando i dettagli al seguito della trattazione.
Sia n un numero composto di cui si desidera trovare la fattorizzazione; come visto nell'algoritmo di fattorizzazione di Fermat, $n = ab = [\frac{1}{2}(a+b)]^2 - [\frac{1}{2}(a-b)]^2$. Quindi per ogni n composto esistono x e y tali che $n = x^2 - y^2 = (x+y)(x-y)$ (interi nel caso in cui a e b siano entrambi pari o entrambi dispari). Il metodo di Fermat ricerca in modo diretto una relazione $y^2 = x^2 - n$, con valore atteso di complessità esponenziale.
Si consideri ora una relazione del tipo:

$$x^2 \equiv y^2 (\text{mod } n). \tag{3.1}$$

Se $x \not\equiv \pm y (\text{mod } n)$ è possibile fattorizzare n semplicemente calcolando $\gcd(n,\, x-y)$[1].

Se invece si dispone di un certo numero di relazioni $u_i^2 \equiv v_i (\text{mod } n)$, si può pensare di combinarle in modo da ottenere una relazione nella forma della (3.1). Ad esempio,

[1] Per convincersene si pensi che $n | x^2 - y^2 = (x+y)(x-y)$ ma non divide nessuno dei due fattori, quindi n deve dividere in parte $x+y$ e in parte $x-y$

dalle relazioni

$$\begin{aligned} u_1^2 &\equiv a \pmod{n} \\ u_2^2 &\equiv ab \pmod{n} \\ u_3^2 &\equiv b \pmod{n} \end{aligned}$$

si ottiene:

$$(u_1 u_2 u_3)^2 \equiv (ab)^2 (\text{mod } n).$$

Il Quadratic Sieve propone un modo efficiente per trovare relazioni di questo tipo e nella prossima sezione si mostrerà con argomentazioni euristiche che un simile procedimento dà un tempo di esecuzione sub-esponenziale.

Si ricorda che un numero u B-smooth è un numero i cui fattori primi sono tutti minori o uguali a B. Vale quindi:

$$u = \prod_{i \leq \pi(B)} p_i^{e_i}$$

Nel seguito si indicherà con $\vec{v}(u)$ il vettore

$$\vec{v}(u) = \big([e_1]_2, \ldots, [e_{\pi(B)}]_2\big) \in \mathbb{F}_2^{\pi(B)}.$$

Un intero B-smoooth u è quadrato se e solo se i termini della sequenza degli esponenti sono tutti pari, ovvero se e solo se $\vec{v}(u) = 0$.

Definizione 3.1. *Una relazione del tipo* $x^2 \equiv u \mod n$ *è detta* B-smooth *se* u *è un intero* B*-smooth.*

Assegnate $\pi(B) + 1$ relazioni B-smooth

$$x_i^2 \equiv u_i \mod n, \quad i = 1, ..., \pi(B) + 1,$$

i corrispondenti vettori $\vec{v}(u_i)$, $i = 1, ..., \pi(B) + 1$, sono linearmente dipendenti in $\mathbb{F}_2^{\pi(B)}$ e pertanto esiste una loro combinazione lineare a coefficienti in $\mathbb{F}_2 = \{0, 1\}$ che produce il vettore nullo. Sia

$$\sum_{j=1}^{\ell} \vec{v}(u_{i_j}) = 0;$$

essendo $\sum_{j=1}^{\ell} \vec{v}(u_{i_j}) = \vec{v}(\prod_{j=1}^{\ell} u_{i_j})$, allora l'intero $\prod_{j=1}^{\ell} u_{i_j}$ è un quadrato e si trova cosí, posto $x = \prod_{j=1}^{\ell} x_{i_j}$ e $y = (\prod_{j=1}^{\ell} u_{i_j})^{1/2}$ la relazione cercata

$$x^2 \equiv y^2 \mod n.$$

Questa relazione potrebbe portare ad avere $x \equiv \pm y (\text{mod } n)$, e fornire quindi una fattorizzazione banale di n. Scritta la $x^2 \equiv y^2 \mod n$ come $x^2 y^{-2} \equiv 1 \mod n$, avere $x \equiv \pm y (\text{mod } n)$ equivale ad avere trovato una delle due soluzioni banali dell'equazione $X^2 \equiv 1 \mod n$. Tale equazione peró , se n ha k fattori primi distinti, ammette 2^k

soluzioni[2], quindi $2^k - 2$ radici sono non banali. Dunque per ogni equivalenza trovata si ha una probabilità $\frac{2^k-2}{2^k}$ di trovare una fattorizzazione. Dal momento che in genere non è noto a priori quanti fattori ha un numero da fattorizzare, se si considera il caso peggiore con $k = 2$, trovando m equivalenze si ha una probabilità $p \geq 1 - (1/2)^m$ di fattorizzare n.

Per trovare relazioni B-smooth, il QS propone di considerare le relazioni $X^2 \equiv [X^2]_n$ al variare di X tra gli interi nell'intervallo $[\lceil\sqrt{n}\rceil, \lceil\sqrt{2n}\rceil[$. Per le restrizioni poste su X si ha $[X^2]_n = X^2 - n$ e quindi le relazioni da considerare hanno il seguente aspetto:

$$X^2 \equiv X^2 - n \mod n.$$

Si tratta dunque di scoprire per quali $X = x$, $\sqrt{n} \leq x \leq \sqrt{2n}$, il numero $f(x) := x^2 - n$ è B-smooth, studiandone la fattorizzazione in primi. Si osservi che se $p|f(x)$, allora $p|f(x + kp)$; questo permette di sfruttare per l'individuazione dei fattori degli $f(x)$ al variare di x, un crivello (*sieve*) simile a quello di Eratostene per la ricerca di numeri primi. Il fatto che si usi questo crivello sul polinomio quadratico $f(X) = X^2 - n$ spiega il nome dell'algoritmo.

Queste idee stanno alla base della variante più semplice dell'algoritmo, in seguito si vedranno tecniche che permettono la ricerca più efficace di relazioni B-smooth e la combinazione veloce delle relazioni. Ora si mostrerà che l'algoritmo ha una complessità sub-esponenziale.

3.2 Complessità computazionale

Le principali componenti da quantizzare per ricavare una stima della complessità dell'algoritmo sono: il numero di relazioni B-smooth da trovare per ricavare una relazione della forma (3.1), la frequenza con cui una valutazione $f(x)$ è B-smooth e la quantità di lavoro necessaria per sapere se una $f(x)$ è B-smooth o no.

Per quanto riguarda il primo punto, precedentemente era stato proposto di considerare i $\pi(B)$ numeri primi minori di B; in realtà è sufficiente considerare i primi $p \leq B$ per cui $f(X) = X^2 - n \equiv 0 \mod p$ ha soluzione (si veda la sottosezione 2.1.4) e quindi i p tali che $(\frac{n}{p}) = 1$. Sia K la cardinalità dell'insieme di primi con questa proprietà, sono necessarie all'algoritmo $K + 1$ relazioni. Euristicamente si ha che K vale circa $\frac{1}{2}\pi(B)$ (se si considera $\frac{1}{2}$ la probabilità che n sia un residuo quadratico per un dato p).

Il secondo punto richiede di specificare in funzione di n e B la frequenza con cui si incontrano numeri B-smooth. Considerando quanto detto nella sezione 2.3, si potrebbe assumere u^{-u} come frequenza euristica con cui un numero $x^2 \mod n$ è B-smooth, con $u = \ln n \,/\, \ln B$.
In realtà, nel caso del QS, setacciando $f(X) = X^2 - n$ per valori $\sqrt{n} < x < \sqrt{n} + n^\epsilon$, si ha che $f(x) \approx 2n^{1/2+\epsilon}$, con $\epsilon > 0$. Per questo nella frequenza u^{-u} con cui un $f(x)$ analizzato dall'algoritmo è B-smooth è più corretto usare $u = \ln \sqrt{n} \,/\, \ln B = \frac{1}{2} \ln n \,/\, \ln B$.

[2]Sia $n = {f_1}^{e_1} {f_2}^{e_2} \ldots {f_k}^{e_k}$ con ogni f_i fattore primo distinto. Per il teorema cinese del resto si ottiene il sistema $x^2 \equiv 1 (\mathrm{mod} {f_i}^{e_i})$ per $i = 1, \ldots, k$; ciascuna equazione ammette esattamente 2 soluzioni distinte, le equazioni sono k e quindi si hanno 2^k possibili soluzioni.

Infine rimane da determinare il lavoro speso per decidere se un certo $f(x)$ sia B-smooth. Usando un approccio banale come la trial division, sono necessarie al più $\pi(B)$ divisioni. Questa stima può essere fatta crollare a log log B operazioni se si utilizza il seguente metodo di setaccio. Viene mantenuto un vettore di una certa lunghezza (circa B), in modo che quando si trova un certo $f(x)$ divisibile per p è sufficiente eseguire un ciclo su k che imposti tutti i $f(x+kp)$ come divisibili per p. In questo modo per ogni primo p considerato si esegue un'operazione ogni p elementi dell'array e quindi in media $1/p$ operazioni per ogni elemento. Gli elementi per cui si sono individuati un certo numero di fattori molto probabilmente sono B-smooth, ma per accertarsene è necessario eseguire una trial division per tutti i primi nella *factor base*. Per gli altri elementi non è necessario nessuno ulteriore controllo, risparmiando così molto lavoro. La percentuale degli elementi per cui è necessaria la trial division è sufficientemente piccola da essere ammortizzata dalla velocità con cui si trovano relazioni candidate ad essere B-smooth. Perciò il lavoro richiesto per scoprire se una valutazione è B-smooth è proporzionale a:

$$\sum_{p \leq B} \frac{1}{p} \sim \ln \ln(B).$$

Si veda la sottosezione (3.4.2) per dettagli su questa tecnica di setaccio e l'Appendice A per la risoluzione della formula della sua complessità.

Già a questo punto si può vedere come un B minore porti a dover trovare meno relazioni, con conseguente ricombinazione delle relazioni B-smooth più veloce, ma diminuisca anche la probabilità di trovarle, mentre viceversa un B maggiore aumenti probabilità di trovare relazioni e numero di relazioni da trovare, aumentando poi la complessità della combinazione delle relazioni in una della forma di (3.1).
Si cercherà nel seguito di stimare il tempo di esecuzione del QS in funzione di B e n, cercando di trovare il B per cui si ha tempo ottimo.

Innanzitutto occorrono $K+1$ relazioni B-smooth, per trovare ciascuna della quali si devono analizzare circa u^u valori di $f(x)$, per un totale di $(K+1)u^u$ valori setacciati. Ogni elemento costa in media ln ln B, per un costo totale di

$$T(B) = u^u(K+1)\text{ln ln B} \qquad \text{con } u = \frac{\ln n}{2 \ln B}.$$

Poichè $K \approx \frac{1}{2}\pi(B)$, per il Teorema dei Numeri Primi (si veda Teorema A.8) vale anche $K \sim B/\ln B$.
Sia ora $S(B) = u \ln u + \ln B$, vale $\ln T(B) \sim S(B)$, infatti:

$$\begin{aligned} \ln T(B) &= \ln\left(u^u \frac{B}{\ln B}\text{ln ln B}\right) = u \ln u + \ln B - \ln \ln B + \ln \ln \ln \text{B} \\ &\sim u \ln u + \ln B = S(B). \end{aligned}$$

Sostituendo u con il suo valore $\frac{1}{2} \ln n \ / \ \ln B$ e derivando si ottiene:

$$\begin{aligned} \frac{dS}{dB} &= \frac{d}{dB}\left(\frac{\ln n}{2 \ln B} \ln\left(\frac{\ln n}{2 \ln B}\right) + \ln B\right) \\ &= \frac{d}{dB}\left(\frac{\ln n}{2 \ln B}(\ \ln \ln n - \ \ln 2 - \ \ln \ln B) + \ln B\right) \\ &= -\frac{\ln n}{2B \ln^2 B}(\ \ln \ln n - \ \ln 2 - \ \ln \ln B + 1) + \frac{1}{B}. \end{aligned}$$

Ponendo a zero questa, per studiare dove si ha il minimo, si ha:

$$\begin{aligned} 0 &= -\frac{\ln n}{2B\ln^2 B}(\ln\ln n - \ln 2 - \ln\ln B + 1) + \frac{1}{B} \\ \frac{1}{B} &= \frac{\ln n}{2B\ln^2 B}(\ln\ln n - \ln 2 - \ln\ln B + 1) \\ 2\ln^2 B &= \ln n(\ln\ln n - \ln 2 - \ln\ln B + 1) \\ 2\ln^2 B &= \ln n\left((1-\ln 2) + \ln\frac{\ln n}{\ln B}\right) \end{aligned} \tag{3.2}$$

e si vede come $\ln(B)$ vale tra un numero costante di volte $\sqrt{\ln n}$ e un costante numero di volte $\sqrt{\ln n \ln\ln n}$. In particolare vale $\ln\ln B \sim \frac{1}{2}\ln\ln n$.

Grazie a questa si vede che il B critico e le altre quantità d'interesse si comportano come[3]:

$$\ln B \sim \frac{1}{2}\sqrt{\ln n \ln\ln n}, \quad u \sim \sqrt{\ln n/\ln\ln n}, \quad S(B) \sim \sqrt{\ln n \ln\ln n}.$$

In particolare si ha che la scelta ottima per B nella ricerca di numeri B-smooth è circa $B = \exp\left(\frac{1}{2}\sqrt{\ln n \ln\ln n}\right)$ e con questa scelta si ha una complessità per il setaccio di circa B^2, cioè il tempo di esecuzione è proporzionale a $\exp\left(\sqrt{\ln n \ln\ln n}\right) = L(n)$.

Ignorando la fase di combinazione delle relazioni B-smooth (che si può dimostrare anch'essa essere dell'ordine di B^2) e assumendo vere le euristiche usate nel quantificare le varie entità dell'algoritmo, questo diventa un algoritmo deterministico per la fattorizzazione di numeri interi con una complessità $L(n)^{1+o(1)}$, che è una funzione sub-esponenziale[4].

3.3 Esempio

In questa sezione viene proposta l'esecuzione del Quadratic Sieve per un numero n piccolo, alla quale si farà riferimento nel corso delle altre sezioni del capitolo come esempio concreto per illustrare le varie migliorie apportabili all'algoritmo.

Si supponga di voler fattorizzare $n = 832553$ usando $B = 100$. Si eseguirà un setaccio del polinomio $f(X) = X^2 - n$ per $X \geq 913$ (dato che $\lceil\sqrt{n}\rceil = 913$).
Ci sono 25 numeri primi minori di B, ma come già visto si è interessati ai p tali che $(\frac{n}{p}) = 1$, in quanto sono gli unici che dividono gli $f(X)$ (infatti per definizione sono gli unici che possono risolvere $X^2 \equiv n \pmod p$). In questo esempio essi sono 13 e vanno a formare la *factor base* $FB = [2, 7, 19, 37, 47, 53, 59, 61, 73, 79, 83, 89, 97]$.

[3]Sostituendo $\ln\ln B = \frac{1}{2}\ln\ln n$ in (3.2) si ha $2\ln^2 B \sim \frac{1}{2}\ln n \ln\ln n$, cioè $\ln B \sim \frac{1}{2}\sqrt{\ln n \ln\ln n}$, nella $u = \frac{\ln n}{2\ln B}$ si sostituisce il risultato appena ottenuto per ottenere quello citato nel testo, mentre $S(B) = u\ln u + \ln B$ diventa $\sqrt{\ln n/\ln\ln n}\ln(\sqrt{\ln n/\ln\ln n}) + \frac{1}{2}\sqrt{\ln n\ln\ln n} \sim \frac{1}{2}\sqrt{\ln n\ln\ln n} + \frac{1}{2}\sqrt{\ln n\ln\ln n} = \sqrt{\ln n\ln\ln n}$.

[4]Le complessità più comuni sono funzioni polinomiali (x^n) o esponenziali (e^x); una funzione sub-esponenziale (o super-polinomiale) ha la proprietà di essere $e^{o(x)}$, mentre ogni x^i è *o piccolo* di una funzione sub-esponenziale. In questo caso si vede facilmente che $e^{\sqrt{\ln n \ln\ln n}}/n \to 0$ per $n \to \infty$ mentre $\log^i n/e^{\sqrt{\ln n \ln\ln n}} \to 0$. Si noti che le dimensioni del problema in questo caso sono $\ln n$, quindi n è una complessità esponenziale rispetto $\ln n$.

Il principio su cui si basa il setaccio è il seguente: una volta che si scopre che $f(913)$ è divisibile per 2, è noto anche che ogni $f(913 + 2k)$ è divisibile per 2. Allo stesso modo, poichè $f(916)$ è divisibile per 7, anche ogni $f(916 + 7k)$ lo sarà e così via. Questo permette di evitare l'uso della trial division per ogni valutazione del polinomio.
Per individuare in modo veloce la prima valutazione di $f(X)$ divisibile per un certo p_i, bisogna preventivamente calcolare le soluzioni $\pm x_i$ di $X^2 \equiv n \pmod{p_i}$. In questo modo è sufficiente usare $X = 913 + [-[913]_p + x_i]_p$ e $X = 913 + [-[913]_p - x_i]_p$ (con $-[913]_p$ si porta X ad avere resto 0 nella divisione per p, e aggiungengo $\pm x_i$ si ottiene il resto voluto).

Nella sottosezione 3.4.2 si tratterà nel dettaglio come eseguire in modo efficiente il setaccio, per ora si consideri il seguente procedimento semplificato: prendendo spunto da quanto detto nella sezione 3.2, si supponga di mantenere in memoria un array di 100 elementi, il cui elemento di indice i-esimo corrisponde alla valutazione $f(i + 913)$.

0	1		...		i		...		99
$f(913)$	$f(914)$		...		$f(913 + i)$		...		$f(1012)$

Tabella 3.1: Array delle prime 100 valutazioni di $f(X)$

Quando si scopre che p divide $f(913+i)$ si aggiunge $\log(p)$ alle celle $i+kp$ dell'array, per ogni k valido. Ogni elemento i che raggiunge il valore $\log(f(913 + i))$ è B-smooth. Nelle Tabelle 3.2, 3.3 e 3.4 si mostrano gli array ottenuti dopo il setaccio per il solo elemento 2, per gli elementi 2 e 7 e per tutti gli elementi della *factor base*.

	0	1	2	3	4	5	6	7	8	9
0	2.079	0	4.158	0	2.772	0	2.079	0	2.079	0
10	2.772	0	3.465	0	2.079	0	2.079	0	3.465	0
20	2.772	0	2.079	0	2.079	0	2.772	0	4.158	0
30	2.079	0	2.079	0	5.545	0	2.772	0	2.079	0
40	2.079	0	2.772	0	3.465	0	2.079	0	2.079	0
50	3.465	0	2.772	0	2.079	0	2.079	0	2.772	0
60	6.238	0	2.079	0	2.079	0	4.158	0	2.772	0
70	2.079	0	2.079	0	2.772	0	3.465	0	2.079	0
80	2.079	0	3.465	0	2.772	0	2.079	0	2.079	0
90	2.772	0	4.158	0	2.079	0	2.079	0	4.852	0

Tabella 3.2: Logaritmi dopo il setaccio per l'elemento 2.

In realtà il setaccio viene compiuto solo per la potenza prima dei primi della *factor base*, in modo da tenere bassa la complessità computazionale rispetto all'informazione raccolta (infatti setacciando per p^2 si raccoglierebbe un'ulteriore informazione $\log(p)$ ogni p^2 campioni) quindi non sempre si raggiungerà $\log(f(x))$ anche nel caso di $f(x)$ B-smooth. Ad esempio nella Tabella 3.2 tutti gli elementi maggiori di 0 sarebbero impostati a $\log(2) \approx 0.639$ non tenendo conto del grado reale con cui 2 divide un $f(x)$. La tabella 3.5 è quella che si ottiene utilizzando solo le potenze prime nel setaccio. Poichè con

	0	1	2	3	4	5	6	7	8	9
0	2.079	0.0	4.158	1.945	2.772	1.945	2.079	0.0	2.079	0.0
10	4.718	0.0	5.411	0.0	2.079	0.0	2.079	1.945	3.465	1.945
20	2.772	0.0	2.079	0.0	4.025	0.0	4.718	0.0	4.158	0.0
30	2.079	1.945	2.079	1.945	5.545	0.0	2.772	0.0	4.025	0.0
40	5.971	0.0	2.772	0.0	3.465	3.891	2.079	1.945	2.079	0.0
50	3.465	0.0	4.718	0.0	4.025	0.0	2.079	0.0	2.772	1.945
60	6.238	1.945	2.079	0.0	2.079	0.0	6.104	0.0	4.718	0.0
70	2.079	0.0	2.079	1.945	2.772	1.945	3.465	0.0	2.079	0.0
80	4.025	0.0	5.411	0.0	2.772	0.0	2.079	1.945	2.079	3.891
90	2.772	0.0	4.158	0.0	5.971	0.0	4.025	0.0	4.852	0.0

Tabella 3.3: Logaritmi dopo il setaccio per gli elementi 2 e 7.

	0	1	2	3	4	5	6	7	8	9
0	2.079	0.0	8.449	1.945	2.772	1.945	9.393	4.110	9.660	3.850
10	4.718	2.944	5.411	0.0	2.079	0.0	5.690	1.945	3.465	1.945
20	2.772	3.970	2.079	0.0	4.025	5.888	4.718	0.0	4.158	4.077
30	5.023	1.945	2.079	6.023	5.545	4.418	2.772	0.0	4.025	0.0
40	5.971	0.0	2.772	4.574	6.410	11.35	2.079	1.945	6.498	2.944
50	3.465	4.488	4.718	3.610	4.025	0.0	5.929	0.0	6.883	1.945
60	6.238	5.916	2.079	2.944	6.448	0.0	6.104	0.0	11.77	0.0
70	6.369	4.574	2.079	1.945	6.742	6.236	3.465	0.0	2.079	0.0
80	4.025	4.488	11.96	0.0	2.772	4.369	2.079	4.890	6.156	3.891
90	6.383	0.0	12.08	0.0	5.971	0.0	4.025	0.0	4.852	0.0

Tabella 3.4: Logaritmi reali delle prime 100 valutazioni di $f(x)$

questa non è possibile distinguere i numeri B-smooth con certezza osservando i logaritmi raccolti, è necessaria una fase di trial division dei numeri corrispondenti agli elementi dell'array che hanno un numero sufficientemente vicino a $\log(f(x))$. Dal momento che si sono introdotte delle approssimazioni si preferisce usare semplicemente $\log(\sqrt{n})$ come soglia, anzichè calcolare $\log(f(i))$ per ogni i, in quando per n grandi hanno lo stesso ordine di grandezza.

Si ha che $\log(\sqrt{n}) \simeq 6.816$; tenendo presente delle imprecisioni dovute al fatto che non si setaccia per le potenze dei primi della *factor base*, si potrebbe usare 4.5 come soglia per tentare la trial division. In questo modo si tenta di fattorizzare 20 valutazioni anzichè 100. Al crescere di n la quantità di numeri non B-smooth che superano questo metodo di setaccio diventa molto bassa.
Nel caso in cui con il primo array non si raccolgano sufficienti relazioni se ne crea un secondo, stavolta con gli elementi che rappresentano le 100 valutazioni successive a quelle appena considerate (vedi Tabella 3.6), rieseguendo il setaccio per questi elementi. Il procedimento va avanti finchè non si sono raccolte sufficienti relazioni (in questo caso se ne raccoglieranno $|FB| + 3 = 16$).

	0	1	2	3	4	5	6	7	8	9
0	0.693	0.0	*4.983*	1.945	0.693	1.945	*8.007*	4.110	*8.274*	3.850
10	2.639	2.944	2.639	0.0	0.693	0.0	4.304	1.945	0.693	1.945
20	0.693	3.970	0.693	0.0	2.639	2.944	2.639	0.0	0.693	4.077
30	3.637	1.945	0.693	*6.023*	0.693	4.418	0.693	0.0	2.639	0.0
40	2.639	0.0	0.693	*4.574*	3.637	*9.406*	0.693	1.945	*5.111*	2.944
50	0.693	4.488	2.639	3.610	2.639	0.0	*4.543*	0.0	*4.804*	1.945
60	0.693	*5.916*	0.693	2.944	*5.062*	0.0	2.639	0.0	*9.694*	0.0
70	*4.983*	*4.574*	0.693	1.945	*4.663*	*6.236*	0.693	0.0	0.693	0.0
80	2.639	4.488	*9.194*	0.0	0.693	4.369	0.693	*4.890*	*4.770*	1.945
90	4.304	0.0	*8.620*	0.0	2.639	0.0	2.639	0.0	0.693	0.0

Tabella 3.5: Logaritmi stimati delle prime 100 valutazioni di $f(x)$

0	1		...		i		...		99
$f(1013)$	$f(1114)$		...		$f(1013+i)$		...		$f(1112)$

Tabella 3.6: Array delle seconde 100 valutazioni di $f(X)$

Con il crivello si selezionano le relazioni B-smooth riportate in Tabella 3.7, che corrispondono ai vettori in $\mathbb{F}_2^{13}$ riportati nell'ultima colonna della stessa. Si ricorda che le componenti del vettore i corrispondono agli esponenti della fattorizzazione di $f(x_i)$ modulo 2 (in quanto si è interessati solo a tener traccia di esponenti pari ed esponenti dispari).

In questo esempio è già presente un vettore nullo, associato alla valutazione $f(290) = 2^8 \cdot 7^4$, che permette di sapere che $(913+290)^2 = 1203^2 \equiv 784^2 (\text{mod } 832553)$. Si può così trovare un fattore di n con $\gcd(832553, 1203 - 784) = 419$; la fattorizzazione completa è in questo caso $419 \cdot 1987$.

Nel caso generale non si riesce a trovare un vettore nullo tra quelli setacciati ed è necessario combinare le relazioni trovate per formarne uno. Ad esempio:

$$\vec{v}(f(2)) + \vec{v}(f(45)) + \vec{v}(f(68)) + \vec{v}(f(92)) + \vec{v}(f(143)) + \vec{v}(f(164)) + \\ + \vec{v}(f(180)) + \vec{v}(f(220)) + \vec{v}(f(265)) = \vec{0}$$

come si può vedere facilmente nella Tabella 3.8 (le colonne sommano a zero).

A questo punto si può trovare una relazione nella forma (3.1) ponendo:

$$x^2 = \big((913+2)\cdot(913+45)\cdot(913+68)\cdot(913+92)\cdot(913+143)\cdot(913+164)\cdot \\ \cdot(913+180)\cdot(913+220)\cdot(913+265)\big)^2 \pmod{n}$$

e

$$y^2 = f(2)\cdot f(6)\cdot f(8)\cdot f(45)\cdot f(68)\cdot f(143)\cdot \\ \cdot f(164)\cdot f(180)\cdot f(220)\cdot f(265) \pmod{n}$$

$f(x)$	Fattori				Vettore
$f(913+2)=4672$	2^6	73^1			(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
$f(913+6)=12008$	2^3	19^1	79^1		(1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
$f(913+8)=15688$	2^3	37^1	53^1		(1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0)
$f(913+45)=85211$	7^2	37^1	47^1		(0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)
$f(913+68)=129808$	2^4	7^1	19^1	61^1	(0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
$f(913+82)=157472$	2^5	7^1	19^1	37^1	(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
$f(913+92)=177472$	2^6	47^1	59^1		(0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0)
$f(913+140)=276256$	2^5	89^1	97^1		(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1)
$f(913+143)=282583$	7^2	73^1	79^1		(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0)
$f(913+164)=327376$	2^4	7^1	37^1	79^1	(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)
$f(913+180)=362096$	2^4	7^1	53^1	61^1	(0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0)
$f(913+201)=408443$	7^1	19^1	37^1	83^1	(0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0)
$f(913+220)=451136$	2^6	7^1	19^1	53^1	(0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
$f(913+265)=555131$	59^1	97^2			(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
$f(913+290)=614656$	2^8	7^4			(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
$f(913+312)=668072$	2^3	37^2	61^1		(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)

Tabella 3.7: Relazioni trovate dal crivello e rispettivi vettori

$\vec{v}(f(2))$	(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
$\vec{v}(f(45))$	(0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)
$\vec{v}(f(68))$	(0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
$\vec{v}(f(92))$	(0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0)
$\vec{v}(f(143))$	(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0)
$\vec{v}(f(164))$	(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)
$\vec{v}(f(180))$	(0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0)
$\vec{v}(f(220))$	(0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
$\vec{v}(f(265))$	(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)

Tabella 3.8: Vettori la cui somma ritorna il vettore nullo

È facile estrarre le radice di x^2 e y^2 in quanto sono noti tutti i loro fattori, quindi è sufficiente moltiplicarli tra loro con esponente dimezzato. Dall'esempio si ottengono i valori $x \equiv 411253 (\text{mod } 832553)$ e $y \equiv 480798 (\text{mod } 832553)$, che portano a trovare il fattore $gcd(832553, 411253 - 480798) = 1987$ e completare come prima la fattorizzazione di n.

3.4 Algoritmo di base

In questa sezione si tratterà la versione base dell'algoritmo del QS. Si propone qui in seguito lo schema generale dell'algoritmo, ogni parte sarà poi approfondita nelle prossime sottosezioni, sia dal punto di vista matematico, sia da quello implementativo (ad alto

livello).

1. **Inizializzazione:** si trovano B, i p tali che $(\frac{n}{p}) = 1$ e le radici $\pm a_i$ di $a_i^2 \equiv n(\text{mod } p)$.

2. **Sieving:** in cui si setaccia il polinomio $f(X) = X^2 - n$ alla ricerca di relazioni B-smooth. Prosegue finchè non ne sono state collezionate K+1.

3. **Algebra Lineare:** per ogni relazione B-smooth trovata $x^2 \equiv u(\text{mod } p)$, genera il corrispondente vettore $\vec{v}(u)$ e ricerca una combinazione di vettori che dia quello nullo: $\vec{v}(x_1) + \cdots + \vec{v}(x_k) = \vec{0}$.

4. **Fattorizzazione:** calcola $x = x_1 x_2 \dots x_k$, $y = \sqrt{f(x_1) \dots f(x_k)}$ e cerca di scoprire fattori di n con $\gcd(n, x - y)$.

3.4.1 Inizializzazione

L'inizializzazione prevede il calcolo delle seguenti quantità:

1. $B = \lceil L(n)^{1/2} \rceil$;

2. Calcolo della *factor base*: la factor base è costituita da 2 e dai primi p, dispari e minori di B, tali che $(\frac{n}{p}) = 1$, cioè i primi che possono effettivamente avere una soluzione a $X^2 - n \equiv 0$ (ovvero possono dividere $f(X)$).

3. Calcolo per ogni p_i in factor base delle radici $\pm a_i$ di $a_i^2 \equiv n(\text{mod } p)$, da usare poi in fase di setaccio del polinomio.

A questo punto è necessario specificare alcuni fatti riguardo a B. Ogni B che sia dell'ordine di $L(n)^{1/2}$ porta a una complessità ottima, ma la scelta di un B minore comporta diversi vantaggi di cui tener conto. Innanzitutto minore è B e meno relazioni B-smooth è necessario trovare, facilitando la fase di Algebra Lineare. Inoltre B minori consentono di sfruttare maggiormente la cache dei calcolatori, dando un ulteriore speed-up ai programmi di setaccio.

Un aspetto negativo di B minori è la minor probabilità di incontrare relazioni B-smooth, ma come si vedrà in seguito la fase di setaccio nel caso si usino più polinomi è altamente parallelizzabile. Dato che gli aspetti negativi possono essere colmati da quelli positivi, spesso si preferisce quindi l'utilizzo di B sub-ottimi in modo da agevolare la fase di Algebra Lineare, che invece è difficilmente parallelizzabile ed è favorita da B minori. Il B ottimo comunque non è definibile in modo assoluto, dipende molto dal modo in cui il QS è stato implementato e dalle caratteristiche delle macchine su cui dovrà essere eseguito.

Per quanto riguarda la fase 2 dell'inizializzazione, è risolvibile in modo efficiente con il Test di Eulero (si veda il teorema (2.3)), mentre per la 3 esistono l'algoritmo di Tonelli (1891) o il metodo di Cipolla (1907) che calcolano le radici quadrate modulo p in modo efficiente (si veda sottosezione 2.3.2 di [2]).

3.4.2 Sieving

Il setaccio del polinomio $f(X) = X^2 - n$ per $X > \sqrt{n}$ alla ricerca di relazioni B-smooth è in genere la parte più dispendiosa in termini temporali dell'algoritmo. In questa fase si raccolgono $K + 1$ coppie $(x_i, f(x_i))$ con $f(x_i)$ B-smooth.
Un metodo molto ingenuo consiste nel tentare una trial division con i primi $p \leq B$ degli $f(x_i)$, ma come già proposto in fase di analisi di complessità dell'algoritmo e nell'esempio, un modo molto più efficiente prevede l'adattamento del crivello di Eratostene alla ricerca di relazioni B-smooth anzichè numeri primi con $f(X)$ come dominio di setaccio.

Sia $f(x_1)$, $f(x_2)$, ..., $f(x_m)$ la sequenza di valutazioni del polinomio di cui si deve scoprire la B-smoothness, questa può essere messa in relazione con un array di m componenti. Ora entrano in gioco le radici $\pm a_i$ individuate in fase di inizializzazione: infatti si ha che $a_i^2 - n \equiv 0 (\text{mod } p)$, ma anche $(a_i + kp)^2 - n \equiv 0 (\text{mod } p)$ e $(-a_i + kp)^2 - n \equiv 0 (\text{mod } p)$. È quindi possibile senza eseguire divisioni, aggiungere $\log(p)$ a ogni casella dell'array corrispondente a un $f(x)$ di cui si scopre la divisibilità per p con questa tecnica (semplicemente calcolando il prossimo $x \equiv \pm a_i \ (\text{mod } p)$ e scorrendo da esso con un ciclo l'array a salti di p). Una volta eseguito il processo per tutti i p della factor base, le caselle dell'array che hanno collezionato $\log(x^2 - n)$ sono le relazioni B-smooth cercate. Si noti che è molto conveniente in termini di prestazioni e di spazio utilizzato un approccio che considera i logaritmi (e quindi usa somme, sottrazioni e una sola cella di memoria per ogni $f(x)$), anzichè eseguire operazioni pesanti come divisioni e moltiplicazioni in precisione multipla.

Un ultimo accorgimento va tenuto sul numero di relazioni da trovare. $K+1$ relazioni B-smooth garantiscono di trovare un'equivalenza della forma della (3.1). Si può assumere che questa abbia probabilità $\frac{1}{2}$ di portare a una fattorizzazione di n, come visto nella sezione 3.1.
Trovando a relazioni B-smooth più del minimo necessario permette di trovare almeno a equivalenze e quindi aumenta la probabilità di terminare l'algoritmo.
In caso nessuna equivalenza porti a una fattorizzazione non banale è sufficiente trovare qualche relazione B-smooth aggiuntiva e ritentare la fattorizzazione, continuando con questo ciclo finchè non si ha successo.

Divisione in blocchi

Per quanto riguarda la lunghezza m dell'array di setaccio, anzichè usare array troppo lunghi è preferibile utilizzarne di relativamente piccoli (rispetto al numero di elementi di $f(x)$ necessari per completare la fattorizzazione), in modo che siano contenuti nelle memorie cache L1 o L2. Ognuno di questi array viene detto *blocco*, e la loro unione ritorna tutta la sequenza corrispondente a $f(x_1)$, $f(x_2)$, ..., $f(x_m)$. A prima vista si potrebbe pensare che così venga compromessa la complessità $\log \log B$, in quando se l'array su cui si setaccia è molto minore di B, per i fattori della factor base maggiori di m si esegue un numero di operazioni proporzionale alla costante $1/m$ anzichè a $1/p$. Questo è in parte vero, ma in realtà la maggior parte del peso computazionale è dovuto ai p piccoli, quelli maggiori incidono meno. Inoltre poter utilizzare gli array nelle cache L1 o L2 garantisce velocità 5-10 volte maggiori che dover accedere nella RAM.

Approccio con i logaritmi

Se si utilizza un approccio con logaritmi, questi devono essere precalcolati, in modo che siano sempre disponibili senza bisogno di ricalcolarli ogni volta che servono. Dal punto di vista implementativo bisogna tenere in considerazione il fatto che l'aritmetica intera è più performante di quella floating point. Per questo potrebbe essere conveniente inizializzare le celle dell'array a $\lceil \log(x^2 - n) \rceil$. A queste si sottraggono i logaritmi approssimati all'intero più vicino dei fattori degli $f(x)$ corrispondenti alle celle, individuati con la tecnica precedente. Per tener conto dell'errore introdotto in realtà si inizializzano le celle a $\lceil \log(x^2 - n) \rceil$ meno un'opportuna costante E, che oltre agli arrotondamenti deve tener conto anche del fatto che il setaccio è eseguito solo per i multipli dei p della factor base, ma non per le loro potenze. Quest'ultimo fatto è legittimo in quanto se si setacciasse anche per i vari p^k si avrebbe un notevole appesantimento computazionale se paragonato alle informazioni ottenute (ad esempio nelle potenze quadre solo un numero ogni p^2 otterrebbe un contributo $\log(p)$ aggiuntivo, che per molti numeri della factor base significherebbe solo un contributo ogni molti blocchi analizzati).

Il vantaggio di inizializzare le celle a $\log(x^2 - n)$ ed eseguire sottrazioni, anzichè inizializzare a 0 e sommare, è una sottigliezza che riguarda l'ottimizzazione del codice. Alla fine si dovrà eseguire il controllo `f[i]<=0` nel primo caso e `f[i]>=soglia` nel secondo. Nel primo caso esistono primitive macchina che svolgono direttamente il confronto, nel secondo sono necessarie invece una sottrazione e un confronto. Ancora meglio, con questa strategia à possibile utilizzare un ingegnoso accorgimento proposto da Contini in [12]. Anzichè controllare una ad una le celle dell'array per vedere se contengono un valore negativo alla fine del setaccio, è possibile controllarle a gruppi per esempio di 4, supponendo ogni cella occupi un intero a 16 bit e il computer abbia word di 64 bit. Ciò è possibile verificando che l'AND di 4 celle consecutive con una maschera 0x80808080 sia non nullo. Così sono necessari solo un quarto dei confronti.
Le celle che valgono meno di zero alla fine del setaccio, contengono i numeri che con buona probabilità sono B-smooth, ma per ciascuno di queste è necessario eseguire una trial division per verificare che abbiano veramente questa proprietà. La trial division ovviamente ha un costo, ma eseguire tutta la fase di setaccio con aritmetica intera da un grande vantaggio di velocità e data la sparsità delle relazioni B-smooth, poter individuare in un certo tempo molte relazioni che molto probabilmente sono B-smooth è vantaggioso rispetto a trovarne molte in meno che però lo sono certamente.

Ci si potrebbe a questo punto chiedere se si possa trarre vantaggio dalla memorizzazione dei fattori dei vari $f(x)$ individuati dal setaccio. Ciò servirebbe per alleggerire le trial division di controllo, in modo da effettuarne solo qualche decine anzichè una per ogni numero nella *factor base* (e quindi diverse migliaia). Nella sezione 3.8 si faranno delle considerazioni sulla frequenza delle relazioni che mostrano come la memorizzazione sia addirittura controproducente.

Introduzione di valori negativi

Un ulteriore raffinamento del setaccio consiste nel considerare un intorno di $\sqrt{n}$ anzichè i valori $x > \sqrt{n}$. In questo modo si avranno numeri leggermente minori in media, ma si dovranno gestire anche numeri negativi. Se si aggiunge nella factor base un elemento

corrispondente al segno, -1, e quindi una coordinata in più nei vettori corrispondenti alle relazioni trovare, l'inconveniente di avere numeri negativi è facilmente gestibile, come se si trattasse di un ulteriore fattore della *factor base*. I numeri negativi avranno infatti valore 1 nella coordinata corrispondente al segno, mentre quelli positivi avranno valore 0. Il vantaggio di avere numeri da analizzare leggermente minori è maggiore del bit necessario per gestire il segno e dell'ulteriore relazione da trovare (infatti la nuova *factor base* avrà $K + 1$ elementi e saranno necessarie $K + 2$ relazioni).

Small Prime Variation

Questa variante di setaccio prende lo spunto dal nome della Large Prime Variation (che però è molto più potente e verrà trattata nella sezione 3.7). La Small Prime Variation considera il fatto che i numeri minori della factor base danno un contributo informativo basso ai blocchi del setaccio (infatti 2 contribuisce per 1 bit, 3 e 5 per 2 bit ecc...), per contro questi sono responsabili di molto lavoro (infatti 2 è responsabile di $m/2$ accessi in memoria, 3 di $m/3$ ecc...). Per questo si decide di non setacciare per i numeri minori di una certa soglia (in genere circa 30-35), aumentando opportunamente il coefficiente di errore E. Un'alternativa è setacciare per la prima potenza di questi numeri che supera la soglia decisa (per esempio si setaccia per $2^6 = 64$, $3^4 = 81$ ecc...).

Un altro metodo per aumentare la velocità di setaccio consiste nell'uso di un moltiplicatore per n, ma dato che utilizza un approccio diverso dai metodi precedenti (che vanno a influire su come si setaccia), verrà trattato a parte nella sezione 3.6.

3.5 Algebra Lineare

La fase di Algebra Lineare ha il compito di trovare un insieme di relazioni B-smooth tra quelle individuate nella fase precedente che moltiplicate tra loro diano un quadrato.
Come già detto, esistono metodi per eseguire questa fase con una complessità $O(B^{2+o(1)})$, facendo si che la complessità complessiva del QS sia ancora $O(B^{2+o(1)})$. In particolare i metodi con questa caratteristica sono il *metodo di Lanczos* adattato ai campi finiti, il *metodo del gradiente* e il *coordinate recurrence method* (noto anche come metodo di Wiedemann).
La tradizionale eliminazione Gaussiana permette di trovare un vettore nullo in tempo $O(B^3)$ (supponendo la matrice $B \times B$). Nonostante questa complessità vada a aumentare la complessità totale, ci sono molte buone ragioni per prenderla in considerazione per fattorizzazioni non troppo grandi. Innanzitutto gli elementi della matrice sono in $\mathbb{F}_2$, quindi si tratta solo di 0 e 1. La loro somma è in realtà uno XOR, una delle operazioni base dei computer, che in genere sono implementate con grande efficienza. Inoltre, utilizzando un computer con parole di n bit (con n che attualmente vale 32 o 64 per i PC) è possibile processare fino a n elementi dei vettori con una sola operazione. In questo modo la notazione $O(\cdot)$ nasconde una costante moltiplicativa molto minore di uno.
Sono possibili inoltre miglioramenti che sfruttando la sparsità della matrice e che rendono l'eliminazione Gaussiana un metodo valido per numeri anche di discrete dimensioni.
Nel Capitolo 4 si tratteranno nel dettaglio la tradizionale riduzione Gaussiana e il metodo di Lanczos a blocchi.

3.6 Moltiplicatore

L'uso di un moltiplicatore è una variante nell'inizializzazione del QS che prevede di moltiplicare prima della fattorizzazione il candidato n per una piccola costante priva di quadrati k, con l'intento di portare un maggior numero di primi 'piccoli' nella *factor base*. Si ricorda che i primi p nella *factor base* di n sono quelli tali che $(\frac{n}{p}) = 1$, quindi il numero kn avrà una *factor base* in generale diversa da n.
Nel QS per un dato kn ci si aspetta di avere dei residui di dimensione $\sqrt{kn} = e^{log\sqrt{n}+log\sqrt{k}}$.
Si può considerare qual è in media il contributo che un primo nella *factor base* porta alla fattorizzazione delle valutazioni degli $f(x)$. Il valore atteso del contributo delle potenze di $p = 2$ a $\log(x^2 - kn)$ vale

$$E_2 = \begin{cases} \frac{1}{2}\log(2) & \text{se } kn \equiv 3 \pmod 4 \\ \log(2) & \text{se } kn \equiv 5 \pmod 8 \\ 2\log(2) & \text{se } kn \equiv 1 \pmod 8 \end{cases}$$

mentre per un generico $p > 2$ tale che $\left(\frac{kn}{p}\right) = 1$ si ha $E_p = (2\log p)/(p-1)$ [5] . Se $p|k$ invece $E_p = (\log p)/p$ [6].
Si può quindi migliorare la stima media delle dimensioni dei residui, sottraendo il contributo medio dei primi della *factor base*, ottenendo residui di dimensioni medie

$$e^{\log\sqrt{n}+\log\sqrt{k}-\sum_{p\in FB} E_p}.$$

Al variare di k si ha quindi il valore atteso per i residui minimo massimizzando:

$$F(k,n) = -\frac{1}{2}\log|k| + \sum_{p\leq B} E_p \qquad \text{con } p = 2, \left(\frac{kn}{p}\right) = 1 \text{ o } p|k$$

Un opportuno k permette una fattorizzazione più veloce in quanto massimizza il contributo atteso alla fattorizzazione degli $f(x)$ da parte della *factor base*. Nonostante questo bisogna stare attenti a non sceglierlo troppo grande, in quanto aumenta il numero di bit dei numeri setacciati di un fattore $\sqrt{k}$ (di cui si tiene conto nella $F(k,n)$ con il termine $-\frac{1}{2}\log|k|$), riducendo la frequenza con cui si incontrano numeri B-smooth.
Nella pratica, non essendo necessario conoscere con esattezza $F(n,k)$, si massimizza la funzione solo per i numeri primi della *factor base* inferiori a una certa soglia, ad esempio 10000.
L'utilizzo del moltiplicatore può portare ad una velocizzazione nella fattorizzazione anche di un fattore 2 o 3, specie per numeri con una *factor base* con pochi primi bassi. Per maggiori dettagli si veda la sezione 5.1.2 oppure [7].

[5] Dato p tale che $(\frac{kn}{p}) = 1$, si ha che $p|X^2 - kn \Leftrightarrow X^2 \equiv kn \pmod p$. Quest'ultima ha due soluzioni in $[0, p-1]$. Ciascuna di queste in $\frac{1}{p} - \frac{1}{p^2}$ dei casi dà contributo $\log(p)$, in $\frac{1}{p^2} - \frac{1}{p^3}$ dei casi dà contributo $2\log(p)$ e così via. Quindi $E_p = 2log\text{p}\sum_{i=1}^{\infty} i\left(\frac{1}{p^i} - \frac{1}{p^{i+1}}\right) = \frac{2\log p}{p-1}$.

[6] Per $p|k$, k senza quadrati, si ha che $x^2 - kn$ è divisibile per p solo se $x = vp$ e in questo caso, dato che $kn = p\alpha n$, si ha $x^2 - kn = p(pv^2 - \alpha n)$. Il termine tra parentesi non è ulteriormente divisibile per p, quindi per $1/p$ dei valori di x si ha una contributo $\log(p)$ e quindi $E_p = \frac{\log p}{p}$.

3.7 Large Prime Variation

La *Large Prime Variation* (LPV) è un metodo intelligente per aumentare in parte il *bound* B della factor base senza risentire delle conseguenze negative che questo comporta.

Come si è già avuto modo di discutere, l'aumento di B provoca un aumento del numero di relazioni da trovare, oltre che a una perdita di efficenza del setaccio (in quanto per p grandi si esegue nel blocco usato per il setaccio un lavoro costante $1/m$ anzichè $1/p$); le conseguenze peggiori riguardano comunque la fase di algebra lineare, dove risolvere matrici più grandi porta a un rallentamento degli algoritmi di soluzione dovuto al fatto che devono essere eseguiti su più macchine, con alti costi di comunicazione.

Durante il setaccio, oltre che a relazioni B-smooth, vengono fattorizzate completamente anche molte altre a cui il crivello ha attribuito una somma dei fattori prossima a $\log(f(x))$. Se una relazione dopo la trial division ritorna un resto nell'intervallo $(B, B^2]$, possiamo affermare con certezza che il resto è un numero primo (in quanto non è divisibile per nessun primo minore della sua radice). Una relazione di questo tipo di dice *parziale* (*partial*).

Queste relazioni finora erano scartate, nonostante comportassero un lavoro non indifferente; la LPV prevede invece di sfruttarle per ottenere nuove relazioni B-smooth a costo quasi nullo. Si supponga di salvare le relazioni parziali trovate e a un certo punto di disporre di k di queste con lo stesso resto P, $P \in (B, B^2]$; siano queste $x_i^2 \equiv y_i P \pmod n$, $i = 1, \ldots, k$. Allora è possibile utilizzare le relazioni

$$(x_1 x_i)^2 \equiv y_1 y_i P^2 \pmod n, \qquad i = 2, \ldots, k$$

nella fase di algebra lineare, in quanto la componente relativa a P è nulla, dato che è elevato al quadrato.

Questa è proprio la strategia usata dalla LPV: si raccolgono tutte le relazioni parziali trovate in fase di setaccio, salvandole assieme al numero primo P ottenuto come resto nella trial division. Periodicamente si ricombinano le relazioni con lo stesso P nel modo sopra specificato per ottenere relazioni B-smooth valide. Si noti che questo non è affatto equivalente ad utilizzare B^2 come bound per la *factor base*: un bound B^2 permetterebbe di considerare anche relazioni con più di un fattore maggiore di B, ma l'impatto sul numero delle relazioni da trovare e sulle dimensioni della matrice da risolvere sarebbe drammatico. La LPV si limita a sfruttare informazione già calcolata che altrimenti verrebbe persa.

Ci si potrebbe chiedere qual'è il contributo che questa tecnica apporta nella ricerca di relazioni. Nelle prime fasi di setaccio ci saranno poche relazioni parziali raccolte con lo stesso P, ma per il paradosso del compleanno, verso la fine della fase di setaccio quando il numero di relazioni salvate sarà consistente, la maggior parte delle relazioni B-smooth trovate avverrà grazie a questa tecnica. In particolare per numeri di medie dimensioni (60-80 cifre decimali) circa metà delle relazioni vengono trovate grazie alla LPV, e all'aumentare delle dimensioni quest'apporto è sempre più consistente. Nella sezione relativa ai risultati sperimentali si confrontano i tempi di esecuzione del QS sfruttando o meno la LPV.

È da specificare anche che il vettore corrispondente a $y_1 y_i$ non è più sparso come quelli corrispondenti a relazioni B-smooth trovate in modo diretto e questo va a penalizzare gli algoritmi nella fase di algebra lineare che sfruttano la sparsità delle relazioni, ma lo

speed-up garantito in fase di setaccio è tale che questo problema diventa di secondaria importanza.

In genere non si salvano tutte le relazioni parziali, ma solo quelle con $P \in (B, 32B]$ o $P \in (B, 128B]$, in quanto al crescere di P diminuisce la probabilità di trovare altre relazioni con lo stesso P.

3.7.1 Double Large Prime Variation

È possibile estendere il concetto della LPV alle relazioni con resto m nell'intervallo $(B^2, B^3]$. Queste o hanno per resto un numero primo o il prodotto di due primi maggiori di B. Viene eseguito un test di pseudo-primalità computazionalmente leggero (come $2^{m-1} \equiv 1 \pmod{m}$) su m; se lo passa è scartato, dato che probabilmente è un primo grande e quindi come detto prima poco probabilmente si troverà un altra relazione con lo stesso m. Se non lo passa invece viene fattorizzato completamente con qualche metodo veloce come il metodo Rho di Pollard (si veda [3], [2] o sottosezione 2.2.4) fornendo una relazione con due primi grandi (*double-partial relation*). Quest'operazione è molto più costosa che nel caso di un unico primo grande come nella LPV, dove si sfruttava una fattorizzazione che doveva essere eseguita comunque.

Queste relazioni possono essere sfruttate in più modi; il più semplice consiste nel combinare una relazione *double-partial* $x_1^2 \equiv y_1 P_1 P_2 \pmod{n}$ con due relazioni parziali $x_2^2 \equiv y_2 P_1 \pmod{n}$ e $x_3^2 \equiv y_3 P_2 \pmod{n}$ secondo lo schema $(x_1 x_2 x_3)^2 \equiv y_1 y_2 y_3 P_1^2 P_2^2 \pmod{n}$. In questo modo i primi grandi P_1 e P_2 sono elevati al quadrato e non intervengono nella fase di algebra lineare. È possibile considerare altri metodi di combinazione, utilizzando anche più relazioni *double-partial* alla volta, ma in questo caso il costo di ricombinazione cresce.

Questo metodo dà buoni risultati per numeri molto grandi (ad esempio sopra le 90-100 cifre decimali), altrimenti per numeri minori l'overhead computazionale non da vantaggi rispetto la LPV.

3.8 Limiti dell'algoritmo di base

Man mano che si procede nel setaccio del polinomio $f(X) = X^2 - n$, trovare relazioni B-smooth diventa un evento sempre più raro, dato il legame tra frequenza dei numeri B-smooth e la taglia dei numeri stessi. Anche se si potenzia l'algoritmo con la Large Prime Variation, per numeri n di una certa dimensione pure le relazioni parziali risentono di questo problema e ci si può aspettare che l'algoritmo trovi relazioni sempre meno frequentemente.
L'euristica considerata assume che n sia B-smooth con probabilità u^{-u} con $u = ln(n)/ln(B)$. Quindi setacciando un polinomio $f(X)$ come nel QS, ci si aspetta di trovare una relazione ogni u^u valutazioni di $f(X)$ considerate, con $u = ln(f(X))/ln(B)$. Se si considera un esempio numerico, ad esempio per $n \simeq 10^{50}$ e $B \simeq 40000$, calcolando il valore u^u per

$x = 1$ ($f(1) \simeq 2\sqrt{n}$) e per $x = 10^6$ ($f(10^6) \simeq 2 \cdot 10^6\sqrt{n}$) si ha:

$$\begin{aligned} u &= \frac{ln(2\sqrt{n})}{ln(40000)} \simeq 5.4977 \Rightarrow u^u \simeq 11732 \\ u &= \frac{ln(2 \cdot 10^6\sqrt{n})}{ln(40000)} \simeq 6.8015 \Rightarrow u^u \simeq 460250 \end{aligned}$$

e addirittura per $x = 10^9$:

$$u = \frac{ln(2 \cdot 10^9\sqrt{n})}{ln(40000)} \simeq 7.4534 \Rightarrow u^u \simeq 3176997$$

La frequenza con cui si trovano relazioni crolla all'aumentare di x. Questi dati permettono anche di fare un'osservazione interessante. Ci si potrebbe chiedere se valga la pena memorizzare i fattori che il crivello ha individuato per le varie valutazioni del polinomio. Si pensi che una *factor base* può contenere migliaia di primi e una volta che si è selezionato un candidato ad essere B-smooth bisogna eseguire una trial division per ciascuno di questi; se si tiene traccia di tutti i fattori dei candidati si riducono le trial division a qualche decina. In realtà non è affatto conveniente memorizzare i fattori individuati dal setaccio: con $B = 40000$ si ha una *factor base* di circa 2000 elementi, mentre ln ln $(40000) \simeq 2,4$ è il numero di operazioni per stabilire la B-smoothness di ogni elemento. Supponendo di setacciare blocchi di 100.000 elementi, al crescere di x (per esempio $x = 10^6$), per ogni array verrebbero effettuati circa 250.000 accessi in memoria per salvare i fattori, mentre solo circa ogni 4-5 blocchi si troverà una relazione B-smooth. Supponendo che il setaccio selezioni k candidati ad essere B-smooth per ogni blocco (in una situazione reale $k \sim 5$), conviene tentare la divisione dei candidati per tutti i 2000 elementi della factor base (con $k \cdot O(1000)$ accessi in memoria) piuttosto che salvare i fattori per ogni elemento e conoscere già i fattori dei candidati (O(100000) accessi inutili per salvare i fattori e $k \cdot O(10)$ accessi nella trial division). In fase di setaccio, al crescere di x questa differenza nel numero di accessi risulta sempre maggiore. Per numeri piccoli (sotto le 45 cifre) la differenza non si nota, ma per n grandi le relazioni B-smooth sono così rare che fin da subito memorizzare i fattori sarebbe molto penalizzante.

Per raccogliere tutte le relazioni necessarie su un unico polinomio già per numeri di 50 cifre decimali è necessario considerare un numero di valutazioni dell'ordine di 10^{10}. Tutte le precedenti considerazioni mostrano come sia auspicabile tenere i valori di $f(X)$ il più bassi possibile. Si potrebbe pensare nel momento in cui le relazioni trovate in $f(X)$ diventano troppo rare, di cambiare polinomio da setacciare, scegliendone uno in modo che le quantità con cui si ha a che fare tornino dell'ordine di $\sqrt{n}$.

Ci si potrebbe chiedere se i polinomi della forma $f_k(X) = X^2 - kn$ per $X \geq \lceil\sqrt{kn}\rceil$ possano fare al caso. La risposta è no, in quanto il polinomio $f(X) = X^2 - n$ nella sua *factor base* ha i primi p tali che $(\frac{n}{p}) = 1$ (cioè per i quali $X^2 - n \equiv 0 \pmod p$ ha soluzione), mentre il generico $f_k(X) = X^2 - kn$ ha nella sua *factor base* i primi p tali che $(\frac{kn}{p}) = 1$, che non sono necessariamente gli stessi del precedente. Questo comporterebbe all'espansione della *factor base* (che potenzialmente potrebbe coinvolgere tutti i primi minori di B) con conseguente aumento del numero di relazioni da trovare (che diventerebbero il doppio), ma soprattutto con aumento della complessità della matrice (che aumenterebbe di quattro volte).

Nel seguito si illustreranno le strategie proposte nel tempo per permettere di velocizzare la ricerca di relazioni tramite cambio di polinomi mantenendo la stessa *factor base*.

3.9 Polinomi Multipli

La possibilità di sostituire X^2-n con più polinomi è stata proposta in modo indipendente da Davis, Holdridge e Montgomery. In particolare nel seguito si propone il metodo di Montgomery che è quello più usato nelle correnti implementazioni di QS, in quanto più pratico ed efficiente degli altri. In letteratura va sotto il nome di *Multiple Polynomials Quadratic Sieve* (MPQS).
L'idea di base è sostituire X con un'opportuna combinazione lineare. In particolare, dati a, b e c tali che $b^2 - ac = n$, si considerano polinomi della forma $f(X) = aX^2 + 2bX + c$. In questo modo

$$af(X) = a^2X^2 + 2abX + ac = a^2X^2 + 2abX + b^2 - n = (aX + b)^2 - n$$

e vale inoltre

$$(aX + b)^2 \equiv af(X) \pmod{n}. \tag{3.3}$$

Se a è costituito da una componente quadrata e da un numero B-smooth, è possibile ricercare relazioni B-smooth su $f(X)$ e poi usare le relazioni trovate nella fase di algebra lineare allo stesso modo di quelle trovate nel QS tradizionale. Infatti la componente quadrata di a non partecipa ai vettori degli esponenti, mentre la composizione della parte B-smooth di a e di $f(X)$ è ancora B-smooth. La caratteristica più importante di questa famiglia di polinomi è che il loro resto $f(X)$ è divisibile per gli stessi p del QS, cioè quelli tali che $(\frac{p}{n}) = 1$.
Le condizioni da porre per a, b e c per ottenere i valori di $f(X)$ minimi possibili dipendono dall'intervallo in cui si vuole setacciare il polinomio. Si supponga di voler setacciare per un intervallo lungo circa $2M$. È utile prendere $|b| \leq \frac{1}{2}a$ (assumendo a positivo), in modo che l'intervallo sia $[-M, M]$. Sfruttando la relazione

$$f(X) = \frac{(aX + b)^2 - n}{a}$$

si può trovare il valore massimo di $f(X)$ agli estremi dell'intervallo, dove $f(M) \simeq (a^2M - n)/a$, e minimo in $X = 0$, con $f(0) \simeq -n/a$. Se si suppongono questi valori circa uguali in modulo si ha $a^2M \approx 2n$ e cioè $a \approx \sqrt{2n}/M$.
Per questo valore di a, $f(X)$ è limitato nell'intervallo $[-M, M]$ da

$$f(M) = \frac{(\frac{\sqrt{2n}}{M}M + b)^2 - n}{\frac{\sqrt{2n}}{M}} \approx \frac{2n - n}{\frac{\sqrt{2n}}{M}} = \frac{M\sqrt{n}}{\sqrt{2}}.$$

Il massimo raggiunto dal polinomio del QS originale X^2-n nell'intervallo $[\sqrt{n}-M, \sqrt{n}+M]$ è invece circa

$$(\sqrt{n} + M)^2 - n \approx 2M\sqrt{n},$$

maggiore di un fattore $2\sqrt{2}$ rispetto a quello raggiunto dal polinomio proposto ora.
In realtà non si risparmia solo questo valore: mentre nel QS base si continua a setacciare lo stesso polinomio fino a trovare tutte le relazioni, nel MPQS una volta raggiunti i limiti di setaccio $-M$ e M si cambia polinomio, mantenendo sempre circa le stesse dimensioni delle valutazioni di $f(X)$. Nel QS invece le valutazioni continuano a crescere indefinitamente.
Se si pone $M \sim B \sim \sqrt{L(n)}$ nel MPQS e si stima $M \sim B^2 \sim L(n)$ nel caso si usi un solo polinomio, risulta che le valutazioni sono in media minori di un fattore B nel primo caso. Questo ha permesso a Montgomery di stimare euristicamente uno speed-up di circa $\frac{1}{2}\sqrt{\ln n \ln \ln n}$ per il MPQS rispetto al QS (il che significa un tempo di setaccio 18 volte minore per numeri di 100 cifre).

Montgomery suggerisce inoltre di scegliere $a = p^2$, usando vari $p \sim \sqrt{\frac{\sqrt{2n}}{M}}$ con $(\frac{n}{p}) = 1$. In questo modo è possibile risolvere $b^2 \equiv n \pmod{a}$ in modo efficiente e si può scegliere la soluzione $|b| \leq \frac{1}{2}a$. Inoltre si può scegliere $c = (b^2 - n)/a$.
Cambiare spesso polinomi permette di mantenere $f(x)$ minori e avere maggior probabilità di ottenere relazioni B-smooth. Nonostante questo, l'operazione di setaccio è molto veloce, mentre l'inizializzazione del polinomio ha un certo costo. Si ricorda che nella fase di inizializzazione vengono calcolate le radici di $X^2 \equiv n \pmod{p_i}$ per i primi della *factor base*, in modo da poterli usare per trovare velocemente il primo elemento dell'array di setaccio divisibile per un certo p_i, come mostrato nella sezione 3.4. Ora i polinomi utilizzati sono nella forma $(ax + b)^2 - n$, quindi date le radici $t_{p_i,1}$ e $t_{p_i,2}$ di $X^2 \equiv n \pmod{p_i}$, è necessario calcolare $a^{-1}(t_{p_i,1} - b) \pmod{p_i}$ e $a^{-1}(t_{p_i,2} - b) \pmod{p_i}$ per poter compiere lo stesso compito.
Calcolare $a^{-1} \bmod p_i$ per i primi della *factor base* è dispendioso, si preferisce quindi usare ogni polinomio per più tempo di quanto suggerito in fase di analisi dello speed-up ($M \approx 10^5 - 10^7$ anzichè B come proposto in precedenza) in modo da ammortizzare in parte il costo di inizializzazione.
Nella prossima sezione si vedrà un metodo che permette di riusare lo stesso a per più polinomi, diminuendo il costo di cambio di polinomio e permettendo di ridurre M e quindi di aumentare ulteriormente la velocità di setaccio attesa.

3.10 Self Initialization

Per risolvere il problema finale presentato nella sezione precedente, Pomerance et al. in [11] hanno proposto un metodo noto come *self initializing quadratic sieve* (SIQS). In [12], Contini mostra come il SIQS garantisca uno speed-up di un fattore 2 rispetto al MPQS.
In questo metodo viene utilizzato un polinomio della stessa forma del MPQS, ma a anzichè essere il quadrato di un primo grande è il prodotto di un certo numero m di primi della *factor base*. In questo modo dal Teorema Cinese del Resto è noto che l'equazione $B^2 \equiv n \pmod{a}$ presenta 2^k soluzioni, di cui 2^{k-1} utilizzabili per la creazione del polinomio d'interesse[7].

[7] Le altre 2^{k-1} sono le soluzioni reciproce modulo a e darebbe gli stessi polinomi.

In particolare sia

$$a = p_1 \cdot p_2 \cdot \cdots \cdot p_m \approx \sqrt{\frac{\sqrt{2n}}{M}}.$$

L'equazione $B^2 \equiv n \pmod{a}$ si scompone nel sistema $B^2 \equiv n \pmod{p_i}$ per $i = 1, \ldots, m$. Le soluzioni $\pm t_i$ delle equazioni prese singolarmente sono state già risolte nella fase di inizializzazione, quindi rimane da calcolare $\alpha_i = \left[(\frac{a}{p_i})^{-1}\right]_{p_i}$ come previsto dal Teorema Cinese del Resto. In questo modo, definito $B_i = \alpha_i \frac{a}{p_i} t_i$, si ha che tutti i possibili B si ottengono come $B = \pm B_1 \pm B2 \pm \cdots \pm B_m$ al variare dei segni.
Pomerance propone di scorrere queste soluzioni attraverso un codice Gray, in modo da poter passare da un b al successivo attraverso una semplice somma. Inoltre precalcolando alcune quantità si può rendere molto efficiente l'inizializzazione di ogni polinomio. Ad esempio precalcolando $2t_i a^{-1} \bmod p_i$ è possibile, una volta calcolato $a^{-1}(t_i - b) \bmod p_i$, trovare anche $a^{-1}(-t_i - b) \bmod p_i$ con una sottrazione, risparmiando così una moltiplicazione.
Il fatto di ammortizzare l'inizializzazione di un polinomio su 2^m polinomi permette di poter usufruire di M minori, e quindi di mantenere più ridotti i residui e avere una maggior probabilità che questi siano B-smooth.

Capitolo 4

Algebra Lineare

Questo capitolo presenta nel dettaglio alcuni algoritmi per la fase di Algebra Lineare. In particolare si mostreranno la riduzione Gaussiana e il metodo a blocchi di Lanczos. La prima si distingue per la sua facilità di programmazione, ma data l'elevata complessità computazionale è adatta solo per istante medio-piccole. Il secondo è invece un metodo euristico che permette di risolvere istanze molto maggiori, dati i suoi bassi costi computazionali.

4.1 Eliminazione Gaussiana

Si dà per noto l'algoritmo di eliminazione di Gauss. Per applicarlo alla ricerca delle combinazioni di vettori che danno il vettore nullo è necessario tener traccia della combinazione lineare corrispondente ad ogni riga. Nell'implementazione realizzata in questa tesi si è usato il seguente metodo. Parallelamente alla matrice degli l vettori trovati si è creata una matrice $l \times l$ dei coefficienti, memorizzati sotto forma di 0 e 1 (0 nella colonna j della riga i se il j-esimo vettore originario partecipa alla combinazione i, 1 altrimenti), sulla quale eseguire le stesse operazioni della matrice dei vettori.
Con questa realizzazione è possibile applicare le stesse considerazioni usate nella matrice dei vettori per sfruttare operazioni efficienti del computer proposte nella sezione 3.5.
Inizialmente la matrice dei coefficienti delle combinazioni non è altro che la matrice identità I_l. La riduzione Gaussiana porta la matrice dei vettori a una forma triangolare, e l'applicazione delle stesse operazioni alla matrice dei coefficienti fa si che questa contenga le combinazioni di vettori per ottenere una determinata riga della matrice dei vettori.
Nelle tabelle seguenti si mostrerà questo procedimento applicato concretamente alla matrice dell'esempio della sezione 3.3. Prima però si presenterà un metodo per rendere più gestibile una matrice così grande.

4.1.1 Riduzione Gaussiana strutturata

La matrice dei vettori iniziale è molto sparsa. Su alcune migliaia di componenti di ogni vettore solo qualche decina è non nulla (si veda sezione 5.4). Pomerance e Smith in [10] mostrano come sfruttare questa caratteristica per ridurre di molto le dimensioni delle matrici coinvolte, con un procedimento definito riduzione Gaussiana strutturata.

Matrice dei vettori
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
(1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
(1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)
(0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1)
(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0)
(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)
(0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0)
(0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)

Tabella 4.1: Matrice originale dell'esempio della sezione 3.3.

Se si osserva una tipica matrice si può inoltre notare come le prime colonne sono molto dense, mentre le colonne corrispondenti ai numeri della *factor base* più grandi hanno poche componenti non nulle. Anche la matrice dell'esempio, nonostante sia relativamente piccola, presenta questa caratteristica. Si potrebbe pensare prima di eseguire la riduzione Gaussiana di eliminare le colonne con tutti zeri (e quindi il corrispondente numero in *factor base*), in quanto la loro presenza è inutile. Se consideriamo le colonne contenenti un solo uno, la riga corrispondente all'occorrenza dell'uno non può intervenire nella creazione del vettore nullo, in quanto nessun'altra riga potrebbe azzerare l'uno; è possibile quindi eliminare anche queste righe e colonne. L'eliminazione di queste potrebbe creare altre colonne con queste caratteristiche, quindi bisogna ciclare il procedimento finchè tutte le colonne hanno almeno 2 uno.
L'eliminazione delle colonne nulle potrebbe sbilanciare molto la differenza tra il numero di righe e di colonne; potrebbe quindi essere utile eliminare un certo numero di righe, tornando però ad una situazione con colonne nulle o con un solo uno. A questo punto diventa necessario eliminarle, continuando ad eseguire queste operazioni in sequenza finchè non sono possibili ulteriori eliminazioni.
Seguendo questo procedimento, nella matrice d'esempio le ultime tre colonne vanno eliminate in quanto presentano un solo 1. Per tornare a bilanciare il numero di righe e colonne si potrebbe pensare di eliminare la prima riga, tornando a creare così una colonna con un solo uno da eliminare e così via.

L'ultimo passo prevede l'eliminazione delle righe con un solo uno (e della colonna corrispondente a quell'occorrenza di uno), con conseguente reiterazione dei passi precedenti.

L'algoritmo completo proposto è quindi:

(0, 0, 0, 0, 0, 0, 0, 0, 1, 0)	
(1, 0, 1, 0, 0, 0, 0, 0, 0, 1)	(1, 0, 1, 0, 0, 0, 0, 0, 1)
(1, 0, 0, 1, 0, 1, 0, 0, 0, 0)	(1, 0, 0, 1, 0, 1, 0, 0, 0)
(0, 0, 0, 1, 1, 0, 0, 0, 0, 0)	(0, 0, 0, 1, 1, 0, 0, 0, 0)
(0, 1, 1, 0, 0, 0, 0, 1, 0, 0)	(0, 1, 1, 0, 0, 0, 0, 1, 0)
(1, 1, 1, 1, 0, 0, 0, 0, 0, 0)	(1, 1, 1, 1, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 1, 0, 1, 0, 0, 0)	(0, 0, 0, 0, 1, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 1, 1)	(0, 1, 0, 1, 0, 0, 0, 0, 1)
(0, 1, 0, 1, 0, 0, 0, 0, 0, 1)	(0, 1, 0, 0, 0, 1, 0, 1, 0)
(0, 1, 0, 0, 0, 1, 0, 1, 0, 0)	(0, 1, 1, 0, 0, 1, 0, 0, 0)
(0, 1, 1, 0, 0, 1, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	(1, 0, 0, 0, 0, 0, 0, 1, 0)
(1, 0, 0, 0, 0, 0, 0, 1, 0, 0)	

Tabella 4.2: Matrice dopo l'eliminazione di colonne nulle o con un solo uno e delle righe corrispondenti. Se si elimina un'ulteriore riga (in modo da ristabilire una differenza relazioni-incognite di 3) è necessario iterare il procedimento di eliminazione, ottenendo la matrice rappresentata a destra, di dimensioni molto minori dell'originale.

(1, 0, 1, 0, 0, 0, 0, 0, 1)	
(1, 0, 0, 1, 0, 1, 0, 0, 0)	(1, 0, 1, 0, 0, 1)
(0, 0, 0, 1, 1, 0, 0, 0, 0)	(1, 0, 0, 1, 0, 0)
(0, 1, 1, 0, 0, 0, 0, 1, 0)	(0, 1, 1, 0, 1, 0)
(1, 1, 1, 1, 0, 0, 0, 0, 0)	(1, 1, 1, 0, 0, 0)
(0, 0, 0, 0, 1, 0, 1, 0, 0)	(0, 1, 0, 0, 0, 1)
(0, 1, 0, 1, 0, 0, 0, 0, 1)	(0, 1, 0, 1, 1, 0)
(0, 1, 0, 0, 0, 1, 0, 1, 0)	(0, 1, 1, 1, 0, 0)
(0, 1, 1, 0, 0, 1, 0, 0, 0)	(0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 1, 0, 0)	(1, 0, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0)	
(1, 0, 0, 0, 0, 0, 0, 1, 0)	

Tabella 4.3: Matrice dopo l'eliminazione di righe con un solo uno e matrice finale ottenuta iterando quanto necessario tutti i passi.

```
while (ci sono colonne con meno di 2 uno) {
    while (ci sono colonne con meno di 2 uno) {
        elimina colonne nulle;
        elimina colonne con un uno e la riga contenente l'uno;
    }
    cancella le righe in eccesso;
    cancella le righe con un uno e la colonna che lo contiene;
}
```

Sperimentalmente si può vedere come si ottengono matrici di dimensioni molto minori e più dense dell'originale, anche con solo un terzo di righe e colonne, permettendo di risolverle in 1/27 del tempo (usando la normale riduzione gaussiana su queste).
Va inoltre fatto notare che conviene molto invertire l'ordine delle componenti dei vettori. Infatti quelle corrispondenti ai numeri nella *factor base* maggiori, essendo più sparse, permettono una triangolarizzazione della matrice più veloce. Per avere degli esempi pratici di differenze di tempi di risoluzione della matrice con queste tecniche si veda la sezione 5.4.

(1, 0, 1, 0, 0, 1)	(1, 0, 0, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 1, 0, 0)	(0, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 1, 1, 0, 1, 0)	(0, 0, 1, 0, 0, 0, 0, 0, 0)
(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 0, 0, 0, 1)	(0, 0, 0, 0, 1, 0, 0, 0, 0)
(0, 1, 0, 1, 1, 0)	(0, 0, 0, 0, 0, 1, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 0, 0, 0, 0, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)
(1, 0, 0, 0, 1, 0)	(0, 0, 0, 0, 0, 0, 0, 0, 1)

Tabella 4.4: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 1.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(1, 0, 1, 0, 0, 1)	(1, 0, 0, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 1, 0, 0)	(0, 1, 0, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 0, 1, 0)	(0, 0, 0, 0, 0, 0, 0, 0, 1)
(0, 1, 1, 1, 0, 0)	(0, 0, 0, 0, 0, 0, 1, 0, 0)
(0, 1, 1, 0, 1, 0)	(0, 0, 1, 0, 0, 0, 0, 0, 0)
(0, 1, 0, 1, 1, 0)	(0, 0, 0, 0, 0, 1, 0, 0, 0)
(0, 1, 0, 0, 0, 1)	(0, 0, 0, 0, 1, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)

Tabella 4.5: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 2.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 0, 0, 0, 0, 0, 1, 0, 0)
(0, 1, 1, 0, 1, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 1)
(0, 1, 1, 0, 1, 0)	(0, 0, 1, 0, 0, 0, 0, 0, 0)
(0, 1, 0, 1, 1, 0)	(0, 0, 0, 0, 0, 1, 0, 0, 0)
(0, 1, 0, 0, 0, 1)	(1, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 0, 0, 0, 1)	(0, 0, 0, 0, 1, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)

Tabella 4.6: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 3.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 0, 1, 1, 0, 1)	(1, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 1, 1, 0, 1)	(0, 1, 0, 1, 1, 0, 0, 0, 0)
(0, 0, 1, 0, 1, 0)	(0, 1, 0, 1, 0, 1, 0, 0, 0)
(0, 0, 0, 1, 1, 0)	(0, 1, 0, 0, 0, 0, 0, 0, 1)
(0, 0, 0, 1, 1, 0)	(0, 1, 1, 1, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 0)	(0, 1, 0, 1, 0, 0, 1, 0, 0)

Tabella 4.7: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 4.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 0, 1, 1, 0, 1)	(1, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 1, 1, 1)	(1, 0, 0, 1, 0, 1, 0, 0, 0)
(0, 0, 0, 1, 1, 0)	(0, 1, 0, 0, 0, 0, 0, 0, 1)
(0, 0, 0, 1, 1, 0)	(0, 1, 1, 1, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 0)	(0, 1, 0, 1, 0, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0)	(1, 0, 0, 1, 1, 0, 0, 0, 0)

Tabella 4.8: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 5.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 0, 1, 1, 0, 1)	(1, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 1, 1, 1)	(1, 0, 0, 1, 0, 1, 0, 0, 0)
(0, 0, 0, 0, 0, 1)	(1, 1, 0, 1, 0, 1, 0, 0, 1)
(0, 0, 0, 0, 0, 1)	(1, 1, 1, 0, 0, 1, 0, 0, 0)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 0)	(0, 1, 0, 1, 0, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0)	(1, 0, 0, 1, 1, 0, 0, 0, 0)

Tabella 4.9: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 6.

(1, 1, 1, 0, 0, 0)	(0, 0, 0, 1, 0, 0, 0, 0, 0)
(0, 1, 1, 1, 0, 0)	(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 0, 1, 1, 0, 1)	(1, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 1, 1, 1)	(1, 0, 0, 1, 0, 1, 0, 0, 0)
(0, 0, 0, 0, 0, 1)	(1, 1, 0, 1, 0, 1, 0, 0, 1)
(0, 0, 0, 0, 0, 0)	(0, 0, 1, 1, 0, 0, 0, 0, 1)
(0, 0, 0, 0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 0)	(0, 1, 0, 1, 0, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0)	(1, 0, 0, 1, 1, 0, 0, 0, 0)

Tabella 4.10: Eliminazione Gaussiana per risolvere la matrice ridotta: passo 7. Si sono trovate 4 combinazioni lineari che portano a un vettore nullo.

4.2 Algoritmo di Lanczos a blocchi

In questa sezione si tratterà dell'algoritmo a blocchi di Lanczos per la risoluzione di sistemi sparsi in GF(2). La trattazione seguirà il lavoro di Montgomery [13], cercando di esplicitarne i dettagli, sfruttando anche [12]. È prima opportuno introdurre i seguenti concetti:

Definizione 4.1. *I vettori* $\mathbf{v}$ *e* $\mathbf{w}$ *in* $\mathbb{K}^n$ *si dicono ortogonali rispetto ad* $\mathbf{A}$ *se* $\mathbf{v}^T\mathbf{A}\mathbf{w} = 0$.

Definizione 4.2. *Un sottospazio* $\mathcal{W} \subseteq \mathbb{K}^n$ *si dice* $\mathbf{A}$*-invertibile se per una sua base* $\mathbf{W}$ *la matrice* $\mathbf{W}^T\mathbf{A}\mathbf{W}$ *è invertibile.*

Si tratta di una buona definizione; infatti l'invertibilità della matrice $\mathbf{W}^T\mathbf{A}\mathbf{W}$ non dipende dalla particolare base di $\mathcal{W}$ considerata. Se infatti $\mathbf{W}_1$ e $\mathbf{W}_2$ sono due basi di $\mathcal{W}$, esiste una matrice invertibile $\mathbf{C}$ tale che $\mathbf{W}_2 = \mathbf{W}_1\mathbf{C}$. Se $\mathbf{W}_1^T\mathbf{A}\mathbf{W}_1$ è invertibile allora anche $\mathbf{W}_2^T\mathbf{A}\mathbf{W}_2 = \mathbf{C}^T\mathbf{W}_1^T\mathbf{A}\mathbf{W}_1\mathbf{C}$ è invertibile in quanto prodotto di matrici invertibili. Se $\mathcal{W}$ è $\mathbf{A}$-invertibile e $\mathbf{W}$ è una sua base, ogni vettore $\mathbf{u} \in \mathbb{K}^n$ può essere scritto in modo unico come $\mathbf{v} + \mathbf{w}$ con $\mathbf{w} \in \mathbf{W}$ e $\mathbf{W}^T\mathbf{A}\mathbf{v} = 0$. La componente in $\mathcal{W}$ si trova ponendo $\mathbf{w} = \mathbf{W}(\mathbf{W}^T\mathbf{A}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{A}\mathbf{u}$. Infatti, scrivendo $\mathbf{w} = \mathbf{W}[(\mathbf{W}^T\mathbf{A}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{A}\mathbf{u}]$ è facile vedere che $\mathbf{w}$ è una combinazione delle colonne di $\mathbf{W}$ e quindi si trova in $\mathcal{W}$. Se poi si considera il vettore $\mathbf{v} = \mathbf{u} - \mathbf{w}$, vale:

$$\begin{aligned}\mathbf{W}^T\mathbf{A}(\mathbf{u} - \mathbf{w}) &= \mathbf{W}^T\mathbf{A}(\mathbf{u} - \mathbf{W}(\mathbf{W}^T\mathbf{A}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{A}\mathbf{u}) = \\ &= \mathbf{W}^T\mathbf{A}\mathbf{u} - \mathbf{W}^T\mathbf{A}\mathbf{W}(\mathbf{W}^T\mathbf{A}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{A}\mathbf{u} = \\ &= \mathbf{W}^T\mathbf{A}\mathbf{u} - \mathbf{W}^T\mathbf{A}\mathbf{u} = \mathbf{0},\end{aligned}$$

che dimostra l'ortogonalità di $\mathbf{v}$ rispetto a $\mathbf{W}$.

4.2.1 L'algoritmo

L'algoritmo a blocchi di Lanczos si usa per risolvere sistemi del tipo $\mathbf{A}\mathbf{x} = \mathbf{b}$, dove $\mathbf{A}$ è una matrice simmetrica $n \times n$ definita su un campo $\mathbb{K}$ (nel nostro caso si userà $\mathbb{K} = GF(2)$).
Questo metodo è iterativo e mira a creare una sequenza di sottospazi $\{\mathcal{W}_i\}$ di $\mathbb{K}^n$, con la proprietà di essere a due a due $\mathbf{A}$-ortogonali e tali che nessun vettore non nullo in un certo $\mathcal{W}_i$ è ortogonale rispetto ad $\mathbf{A}$ a tutti i vettori in $\mathcal{W}_i$ stesso. Dopo un certo numero di iterazioni, diciamo $m \leq n$, l'algoritmo necessariamente produce uno spazio $\mathcal{W}_m$ nullo.

Prima di iniziare il procedimento è necessario fissare un $N > 0$, che limita superiormente le dimensioni dei sottospazi $\mathcal{W}_i$ da determinare.

Innanzitutto si sceglie arbitrariamente una matrice $\mathbf{V}_0$ con n righe ed N colonne. Si seleziona il massimo numero possibile di colonne in modo da ottenere una sottomatrice $\mathbf{W}_0$ di $\mathbf{V}_0$ che sia A-invertibile. Poi, per $i > 0$, si costruisce una matrice $\mathbf{V}_i$, $n \times N$, in modo che sia $\mathbf{A}$-ortogonale a tutti i $\mathbf{W}_j$ già considerati. Dalla $\mathbf{V}_i$ si forma $\mathbf{W}_i$ in modo che contenga il massimo numero di colonne possibili di $\mathbf{V}_i$ e sia $\mathbf{A}$-invertibile. Gli spazi delle colonne $\mathcal{W}_i = \langle \mathbf{W}_i \rangle$ costituiscono la sequenza cercata.

Tale costruzione si riassume nella seguente iterazione:

$$\begin{aligned} \mathbf{W}_i &= \mathbf{V}_i\mathbf{S}_i, \\ \mathbf{V}_{i+1} &= \mathbf{A}\mathbf{W}_i\mathbf{S}_i^T + \mathbf{V}_i - \sum_{j=0}^{i} \mathbf{W}_j\mathbf{C}_{i+1,j} \qquad (i \geq 0), \\ \mathcal{W}_i &= \langle \mathbf{W}_i \rangle \end{aligned} \tag{4.1}$$

dove

- la matrice $\mathbf{S}_i$ è la responsabile della selezione delle colonne di $\mathbf{V}_i$ per costruire $\mathbf{W}_i$. Ha dimensioni $N \times N_i$, dove $N_i \leq N$ è scelto maggiore possibile. Essa è formata dalle N_i colonne della matrice identità $N \times N$ corrispondenti alle colonne di $\mathbf{V}_i$ scelte. Data la stuttura delle $\mathbf{S}_i$, si ha $\mathbf{S}_i^T\mathbf{S}_i = \mathbf{I}_{N_i}$ e $\mathbf{S}_i\mathbf{S}_i^T$ è la matrice ottenuta dalla matrice identità $N \times N$ sostituendo le colonne corrispondenti a quelle di $\mathbf{V}_i$ non scelte con delle colonne nulle.

- le matrici $\mathbf{C}_{i+1,j}$ si calcolano come:

  $$\mathbf{C}_{i+1,j} = (\mathbf{W}_j^T\mathbf{A}\mathbf{W}_j)^{-1}\mathbf{W}_j^T\mathbf{A}\left(\mathbf{A}\mathbf{W}_i\mathbf{S}_i^T + \mathbf{V}_i\right).$$

  In questo modo i termini $\mathbf{W}_j\mathbf{C}_{i+1,j}$ hanno come colonne le proiezioni A-ortogonali su $\mathcal{W}_j = \langle \mathbf{W}_j \rangle$ delle colonne della matrice $\mathbf{A}\mathbf{W}_i\mathbf{S}_i^T + \mathbf{V}_i$. Quindi $\mathbf{V}_{i+1}$ ha le colonne ortogonali a quelle delle matrici $\mathbf{W}_j$, $j \leq i$, e cioè $\mathbf{W}_j^T\mathbf{A}\mathbf{V}_{i+1} = \mathbf{0}$ per $j \leq i$.

La procedura iterativa termina raggiunto il valore $i = m$ per il quale $\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m = \mathbf{0}$.

Esempio 4.1. *Posto $N = 3$, moltiplicando a destra una matrice $\mathbf{V}$ con n righe e 3 colonne per la matrice $\mathbf{S} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$, vengono selezionate la seconda e terza colonna di $\mathbf{V}$. Si ha*

$$\mathbf{S}^T\mathbf{S} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{S}\mathbf{S}^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

È possibile dimostrare il seguente teorema, che ci permetterà poi di costruire una soluzione del sistema.

Teorema 4.1. *Se $\mathbf{V}_m = \mathbf{0}$, le formule (4.1) permettono di costruire una sequenza di sottospazi $\{\mathcal{W}_i\}$ che soddisfa:*

$$\begin{aligned} \mathcal{W}_i \qquad & \text{è } \mathbf{A}\text{-invertibile} \\ \mathcal{W}_j^T\mathbf{A}\mathcal{W}_i = 0 \qquad & \text{per } i \neq j \\ \mathbf{A}\mathcal{W} \subseteq \mathcal{W} \qquad & \text{con } \mathcal{W} = \mathcal{W}_0 + \cdots + \mathcal{W}_{m-1}. \end{aligned} \tag{4.2}$$

Dimostrazione. La prima relazione è valida per come si è scelto S_i, che assicura che $\mathcal{W}_i = \langle \mathbf{W}_i \rangle$ sia $\mathbf{A}$-invertibile.

La relazione $\mathbf{W}_j^T\mathbf{A}\mathbf{V}_i = \mathbf{0}$ per $j < i$ precedentemente mostrata, implica $\mathbf{W}_j^T\mathbf{A}\mathbf{W}_i = \mathbf{0}$. Inoltre la matrice $\mathbf{A}$ è simmetrica, quindi vale anche $\mathbf{W}_i^T\mathbf{A}\mathbf{W}_j = \mathbf{0}$, che implicano

la seconda relazione. Per quanto riguarda la terza, se si moltiplica la (4.1) per $\mathbf{S}_i$ si ottiene:

$$\begin{aligned}\mathbf{V}_{i+1}\mathbf{S}_i &= \mathbf{A}\mathbf{W}_i\mathbf{S}_i^T\mathbf{S}_i + \mathbf{V}_i\mathbf{S}_i - \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i \\ &= \mathbf{A}\mathbf{W}_i + \mathbf{W}_i - \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i.\end{aligned}$$

Da questa e sempre dalla (4.1) si può vedere che

$$\begin{aligned}\mathbf{A}\mathbf{W}_i &= \mathbf{V}_{i+1}\mathbf{S}_i - \mathbf{W}_i + \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i = \mathbf{V}_{i+1}\mathbf{S}_i + \mathbf{v}_1, \qquad (4.3)\\ \mathbf{V}_i &= \mathbf{V}_{i+1} - \mathbf{A}\mathbf{W}_i\mathbf{S}_i^T + \sum_{j=0}^{i}\mathbf{W}_j\mathbf{C}_{i+1,j}\mathbf{S}_i = \mathbf{V}_{i+1} - \mathbf{A}\mathbf{W}_i\mathbf{S}_i^T + \mathbf{v}_2,\end{aligned}$$

dove $\mathbf{v}_1$ e $\mathbf{v}_2$ sono vettori in $\mathcal{W}$. Inoltre per ipotesi si ha $\mathbf{V}_m = \mathbf{0} \in \mathcal{W}$. Applicando l'induzione all'indietro alle ultime relazioni trovate si può vedere che $\mathbf{A}\mathbf{W}_i$ e e $\mathbf{V}_i$ appartengono a $\mathcal{W}$ per $0 \le i \le m-1$ e quindi $\mathbf{A}\mathcal{W} \subseteq \mathcal{W}$. □

Proposizione 4.1. *Dato* $\mathbf{b} \in \mathcal{W}$*, il vettore colonna*

$$\overline{\mathbf{x}} = \sum_{i=0}^{m-1}\mathbf{W}_i(\mathbf{W}_i^T\mathbf{A}\mathbf{W}_i)^{-1}\mathbf{W}_i^T\mathbf{b}, \qquad (4.4)$$

è soluzione del sistema lineare $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Dimostrazione. Si può vedere che $\mathbf{W}_k^T(\mathbf{A}\overline{\mathbf{x}} - \mathbf{b}) = \mathbf{0}$ per ogni k, infatti:

$$\begin{aligned}\mathbf{W}_k^T\mathbf{A}\overline{\mathbf{x}} &= \mathbf{W}_k^T\mathbf{A}\sum_{i=0}^{m-1}\mathbf{W}_i(\mathbf{W}_i^T\mathbf{A}\mathbf{W}_i)^{-1}\mathbf{W}_i^T\mathbf{b} \\ &= \mathbf{W}_k^T\mathbf{A}\mathbf{W}_k(\mathbf{W}_k^T\mathbf{A}\mathbf{W}_k)^{-1}\mathbf{W}_k^T\mathbf{b} = \mathbf{W}_k^T\mathbf{b}.\end{aligned}$$

Dunque moltiplicando il trasposto di un vettore in $\mathcal{W}$ per $\mathbf{A}\overline{\mathbf{x}} - \mathbf{b}$ si ottiene il vettore nullo. Allora, per ogni $0 \le k \le m$, essendo $(\mathbf{A}\mathbf{W}_k)^T \in \mathcal{W}$ (vedi Teorema 4.1) si ha

$$\mathbf{0} = (\mathbf{A}\mathbf{W}_k)^T(\mathbf{A}\overline{\mathbf{x}} - \mathbf{b}) = \mathbf{W}_k^T\mathbf{A}(\mathbf{A}\overline{\mathbf{x}} - \mathbf{b}). \qquad (4.5)$$

Ora $\mathbf{b}$ e $\overline{\mathbf{x}}$ sono in $\mathcal{W}$, così come $\mathbf{A}\overline{\mathbf{x}}$; quindi esistono delle matrici colonne $\mathbf{C}_i$, $0 \le i \le m-1$, tali che

$$\mathbf{A}\overline{\mathbf{x}} - \mathbf{b} = \sum_{i=0}^{m-1}\mathbf{W}_i\mathbf{C}_i. \qquad (4.6)$$

Dunque, per la (4.5), sappiamo che per ogni k

$$\mathbf{0} = \mathbf{W}_k^T\mathbf{A}(\mathbf{A}\overline{\mathbf{x}} - \mathbf{b}) = \mathbf{W}_k^T\mathbf{A}\left(\sum_{i=0}^{m-1}\mathbf{W}_i\mathbf{C}_i\right) = \mathbf{W}_k^T\mathbf{A}\mathbf{W}_k\mathbf{C}_k.$$

Essendo $\mathbf{W}_k^T\mathbf{A}\mathbf{W}_k$ invertibile, si ottiene $\mathbf{C}_k = \mathbf{0}$. Quindi la (4.6) diventa $\mathbf{A}\overline{\mathbf{x}} - \mathbf{b} = \mathbf{0}$. □

Questa proposizione ci permette di sfruttare le basi $\mathbf{W}_i$ trovate dall'algoritmo per determinare una soluzione del sistema nel caso in cui $\mathbf{V}_m = \mathbf{0}$.
Purtroppo si potrebbe terminare anche con $\mathbf{V}_m^T\mathbf{A}\mathbf{V}_m = \mathbf{0}$ ma $\mathbf{V}_m \neq 0$. In questo caso si può considerare il seguente ragionamento euristico. Poniamo $\mathcal{W}_m = \langle \mathbf{V}_m \rangle$. La matrice $\mathbf{V}_m$ è per costruzione $\mathbf{A}$-ortogonale a se stessa e a tutti i $\mathbf{W}_i$ per $i < m$; essa, in genere, ha un rango basso (in [14] si valuta qual'è il valore atteso delle dimensioni di questo sottospazio prendendo $\mathbf{A}$ casuale, che certo non è la nostra situazione). Dalle formule 4.1 dell'iterazione di Lanczos è possibile anche vedere che i $\mathbf{V}_j$ e i $\mathbf{W}_j$ sono contenuti nel sottospazio

$$\mathcal{V} = \langle \mathbf{V}_0 \rangle + \langle \mathbf{A}\mathbf{V}_0 \rangle + \langle \mathbf{A}^2\mathbf{V}_0 \rangle + \dots \tag{4.7}$$

che è detto sottospazio di Krylov. Per N grandi (ad esempio $N > 16$) $\mathcal{W}_m = \langle V_m \rangle$ tipicamente contiene tutti i vettori di $\mathcal{V}$ che sono $\mathbf{A}$-ortogonali a tutti i vettori di $\mathcal{V}$, compresi se stessi. Quando questo avviene, si ha $\mathcal{V} = \mathcal{W} + \mathcal{W}_m$. Dalla definizione di $\mathcal{V}$ si ha

$$\mathbf{A}\mathcal{W}_m \subseteq \mathbf{A}\mathcal{V} \subseteq \mathcal{V} = \mathcal{W} + \mathcal{W}_m.$$

In realtà abbiamo $\mathbf{A}\mathcal{W}_m \subseteq \mathcal{W}_m$; infatti sia $\mathbf{w} = \mathbf{w}_0 + \dots + \mathbf{w}_m \in \mathbf{A}\mathcal{W}_m$, con $\mathbf{w}_j \in \mathcal{W}_j$. Verifichiamo che $\mathbf{w}_j = \mathbf{0}$ per $j = 0, ..., m-1$. Si ha

$$\begin{aligned} \mathbf{w}_j^T\mathbf{A}\mathcal{W}_j &= (\mathbf{w}_0 + \dots + \mathbf{w}_m)^T\mathbf{A}\mathcal{W}_j \subseteq (\mathbf{A}\mathcal{W}_m)^T\mathbf{A}\mathcal{W}_j \\ &= \mathcal{W}_m^T\mathbf{A}(\mathbf{A}\mathcal{W}_j) \subseteq \mathcal{W}_m^T\mathbf{A}(\mathcal{W} + \mathcal{W}_m) = \{0\}. \end{aligned}$$

Dato che $\mathcal{W}_j$ è $\mathbf{A}$-invertibile per $j < m$, si ha $\mathbf{w}_j = \mathbf{0}$. Quindi $\mathbf{A}\mathcal{W}_m \subseteq \mathcal{W}_m$; si osservi inoltre che, per quanto visto in precedenza, $\mathbf{A}(\mathbf{X} - \mathbf{Y})$ non è detto sia la matrice nulla, ma le sue colonne certo appartengono a $\mathcal{W}_m$, essendo ortogonali a tutti i $\mathcal{W}_i$ per $i < m$. I vettori colonna che compongono le matrici $\mathbf{X} - \mathbf{Y}$ e $\mathbf{V}_m$ hanno al più rango $2N$. È possibile usarli per trovare dei vettori nello spazio nullo di $\mathbf{A} = \mathbf{B}^T\mathbf{B}$ e di $\mathbf{B}$. Infatti indichiamo con $\mathbf{Z}$ la matrice $[\mathbf{X} - \mathbf{Y}|\mathbf{V}_m]$; essa ha n righe e $2N$ colonne. La matrice $\mathbf{A}\mathbf{Z}$ ha le colonne in $\mathcal{W}_m$ e quindi rango al più N. Esistono dunque combinazioni lineari delle colonne di $\mathbf{A}\mathbf{Z}$ che producono il vettore nullo. Allo stesso modo si può lavorare con $\mathbf{B}$. Praticamente si considera la matrice $[\mathbf{B}\mathbf{Z}|\mathbf{Z}]$ ottenuta affiancando a $\mathbf{B}\mathbf{Z}$ la matrice $\mathbf{Z}$ e si effettuano le operazioni elementari sulle colonne che portano $\mathbf{B}\mathbf{Z}$ nella sua forma ridotta di Gauss per colonne. Questo equivale a moltiplicare a destra per una matrice invertibile $\mathbf{U}$:

$$[\mathbf{B}\mathbf{Z}|\mathbf{Z}]\mathbf{U} = [(\mathbf{B}\mathbf{Z})\mathbf{U}|\mathbf{Z}\mathbf{U}]$$

Le colonne di $\mathbf{Z}\mathbf{U}$ corrispondenti alle colonne nulle di $(\mathbf{B}\mathbf{Z})\mathbf{U}$ producono una famiglia di generatori dello spazio nullo di $\mathbf{B}$.
Nel prossimo paragrafo si mostrerà come usare l'algoritmo nei sistemi ottenuti nella fattorizzazione di numeri, mostrando inoltre come si comporta sia per $\mathbf{V}_m = \mathbf{0}$ che per $\mathbf{V}_m \neq \mathbf{0}$.

4.2.2 Utilizzo con matrici di fattorizzazione

L'algoritmo finora descritto non è ancora completo, rimane infatti da specificare un metodo per determinare efficientemente le $\mathbf{S}_i$. Vediamo però come può essere applicato al caso delle matrici di fattorizzazione. Sia $\mathbf{B}$ una matrice $n_1 \times n_2$, dove $n_2 > n_1$. Per esempio si consideri la seguente matrice 4×6, ottenuta ad esempio con il QS trovando 6 relazioni con una *factor base* di 4 elementi (finora abbiamo rappresentato la matrice sempre nella forma $\mathbf{B}^T$, con i vettori esponente disposti lungo le righe):

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad \mathbf{B}^T = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Vogliamo trovare un vettore $\overline{\mathbf{x}}$ che ne combini le colonne in modo da trovarne una di nulla, ovvero risolvere il sistema $\mathbf{Bx} = \mathbf{0}$. La matrice non è però né quadrata né simmetrica. Questo problema può essere risolto usando $\mathbf{A} = \mathbf{B}^T\mathbf{B}$, ottenendo cioè la matrice simmetrica $n_2 \times n_2$ (6×6 nell'esempio):

$$\mathbf{A} = \mathbf{B^T B} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Ogni soluzione del sistema $\mathbf{Bx} = \mathbf{0}$ risolve anche $\mathbf{Ax} = \mathbf{0}$, il viceversa non è necessariamente vero se il rango di $\mathbf{A}$ è minore di n_1.
In genere si sceglie N pari al numero di bit di una word elaborabile dal computer su cui si va a programmare l'algoritmo, che usualmente vale 32 o 64. Si sceglie una matrice casuale $\mathbf{Y}$ con n righe ed N colonne, a coefficienti in $GF(2)$; si calcola la matrice $\mathbf{AY}$ e si cerca di trovare una matrice $\overline{\mathbf{X}}$ che risolve il sistema $\mathbf{AX} = \mathbf{AY}$. Si comincia ponendo $\mathbf{V}_0 = \mathbf{AY}$ e poi si procede con le iterazioni previste dall'algoritmo di Lanczos fino ad avere $\mathbf{V}_i^T\mathbf{AV}_i = \mathbf{0}$ per $i = m$. A questo punto, come visto prima, se $\mathbf{V}_m = \mathbf{0}$, si ha che $\mathbf{A}(\overline{\mathbf{X}} - \mathbf{Y}) = \mathbf{0}$; si possono quindi usare le colonne di $\overline{\mathbf{X}} - \mathbf{Y}$ per trovare delle combinazioni delle colonne di $\mathbf{A}$ che danno una matrice nulla.
Se procediamo nell'esempio ponendo $N = 2$ e

$$\mathbf{Y} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix},$$

per $m = 2$ si ha $\mathbf{V}_2 = \mathbf{0}$ e quindi $\mathbf{V}_2^T \mathbf{A} \mathbf{V}_2 = \mathbf{0}$.
Calcolando $\overline{\mathbf{X}}$ come precedentemente indicato si ha

$$\overline{\mathbf{X}} = \sum_{i=0}^{m-1} \mathbf{W}_i \left(\mathbf{W}_i^T \mathbf{A} \mathbf{W}_i \right)^{-1} \mathbf{W}_i^T \mathbf{V}_0 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{e quindi} \quad \overline{\mathbf{X}} - \mathbf{Y} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

In questo caso si è terminato con $\mathbf{V}_m = 0$ e si può verificare che tutte le colonne di $\overline{\mathbf{X}} - \mathbf{Y}$ e le loro combinazioni lineari permettono di ottenere combinazioni nulle delle colonne di $\mathbf{A}$ (e anche di $\mathbf{B}$).

Consideriamo ora le matrici

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{A} = \mathbf{B^T B} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Scegliendo ad esempio

$$\mathbf{Y} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \Rightarrow \quad \mathbf{V}_0 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},$$

si ha subito $\mathbf{V}_0^T \mathbf{A} \mathbf{V}_0 = \mathbf{0}$ e

$$\mathbf{X} - \mathbf{Y} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad \mathbf{A}(\mathbf{X} - \mathbf{Y}) = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},$$

In questo caso $\mathbf{V}_m \neq 0$ e anche $\mathbf{A}(\mathbf{X} - \mathbf{Y}) \neq \mathbf{0}$. Possiamo però considerare come argomentato precedentemente

$$\mathbf{Z} = [\mathbf{X} - \mathbf{Y} | \mathbf{V}_0] = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad \Rightarrow \quad \mathbf{BZ} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Se $\mathbf{V}_0$ contiene tutti i vettori $\mathbf{A}$-ortogonali, la matrice $\mathbf{Z}$ può contenere fino a $2N$ colonne di $\mathcal{V}$, mentre $\mathbf{BZ} \in \mathcal{W}_m$ ne ha al più N di indipendenti. Riducendo quest'ultima è possibile vedere che combinando le colonne 1 e 3, e le colonne 2 e 4 di $\mathbf{Z}$ si ottengono dei vettori nello spazio nullo di $\mathbf{B}$ ($(0\ 0\ 1\ 1\ 1\ 0)^T$ e $(1\ 1\ 0\ 0\ 0\ 1)^T$, entrambi risolvono $\mathbf{Bx} = \mathbf{0}$).

4.2.3 Semplificazione della ricorrenza

La ricorrenza in (4.1) così com'è definita richiede all'iterazione i il calcolo di $i+1$ coefficienti $\mathbf{C}_{i+1,j}$. Man mano che si avanza con le iterazioni il calcolo diventa sempre più pesante e si vuole trovare un modo per rendere più snella la ricorrenza da calcolare. Si noti che per costruzione $\mathbf{W}_j^T\mathbf{AV}_i = \mathbf{0}$ per $j < i$; questo costituisce una prima semplificazione per il calcolo del coefficiente $\mathbf{C}_{i+1,j}$. Per il termine $\mathbf{W}_j^T\mathbf{A}^2\mathbf{W}_i$ invece vale:

$$\begin{aligned}\mathbf{W}_j^T\mathbf{A}^2\mathbf{W}_i &= \mathbf{S}_j^T\mathbf{S}_j\mathbf{W}_j^T\mathbf{A}^2\mathbf{W}_i = \mathbf{S}_j^T(\mathbf{AW}_j\mathbf{S}_j^T)^T\mathbf{AW}_i = \\ &= \mathbf{S}_j^T(\mathbf{V}_{j+1} - \mathbf{V}_j + \mathbf{v})^T\mathbf{AW}_i = (\mathbf{S}_j^T\mathbf{V}_{j+1}^T - \mathbf{W}_j)\mathbf{AW}_i = \\ &= \mathbf{S}_j\mathbf{V}_{j+1}\mathbf{AW}_i\end{aligned}$$

dove $\mathbf{v} = \sum_{k=0}^{j}\mathbf{W}_k\mathbf{C}_{j+1,k}$ è un vettore in $\mathcal{W}_0 + \cdots + \mathcal{W}_j$ e quindi vale $\mathbf{vAW}_i = \mathbf{0}$ in quanto $j < i$. Se per caso $\mathbf{S}_{i+1} = \mathbf{I}_N$, si avrebbe $\mathbf{W}_{i+1} = \mathbf{V}_{i+1}$ e dato che $\mathbf{W}_{j+1}^T\mathbf{AW}_i = \mathbf{0}$ per $j < i-1$, $\mathbf{W}_j^T\mathbf{A}^2\mathbf{W}_i$ si annullerebbe per $j < i-1$, annullando così tutti i coefficienti $\mathbf{C}_{i+1,j}$ per $j < i-1$. In generale però $\mathbf{S}_{i+1} \neq \mathbf{I}_N$.
Nel paper di Montgomery si forzano i vettori di $\mathbf{V}_{j+1}$ a essere contenuti nelle basi $\mathbf{W}_{j+1}$ e $\mathbf{W}_{j+2}$, ovvero:

$$\langle\mathbf{V}_{i+1}\rangle \subseteq \mathcal{W}_0 + \cdots + \mathcal{W}_{j+2}.$$

In questo modo $\mathbf{V}_{j+1}$ è $\mathbf{A}$-ortogonale ai $\mathbf{W}_{j+3}, \ldots, \mathbf{W}_m$ e si ottiene che $\mathbf{C}_{i+1,j} = \mathbf{0}$ per $j \leq i-3$, semplificando la ricorrenza alla seguente forma:

$$\mathbf{V}_{i+1} = \mathbf{AW}_i\mathbf{S}_i^T + \mathbf{V}_i - \mathbf{W}_i\mathbf{C}_{i+1,i} - \mathbf{W}_{i-1}\mathbf{C}_{i+1,i-1} - \mathbf{W}_{i-2}\mathbf{C}_{i+1,i-2}. \tag{4.8}$$

Se definiamo

$$\mathbf{W}_i^{inv} = \mathbf{S}_i(\mathbf{W}_i^T\mathbf{AW}_i)^{-1}\mathbf{S}_i^T = \mathbf{S}_i(\mathbf{S}_i^T\mathbf{V}_i^T\mathbf{AV}_i\mathbf{S}_i)^{-1}\mathbf{S}_i^T$$

è possibile ottenere la (4.8) usando i coefficienti

$$\begin{aligned}\mathbf{D}_{i+1} &= \mathbf{I}_N - \mathbf{W}_i^{inv}(\mathbf{V}_i^T\mathbf{A}^2\mathbf{V}_i\mathbf{S}_i\mathbf{S}_i^T + \mathbf{V}_i^T\mathbf{AV}_i), \\ \mathbf{E}_{i+1} &= -\mathbf{W}_{i-1}^{inv}\mathbf{V}_i^T\mathbf{AV}_i\mathbf{S}_i\mathbf{S}_i^T, \\ \mathbf{F}_{i+1} &= -\mathbf{W}_{i-2}^{inv}(\mathbf{I}_N - \mathbf{V}_{i-1}^T\mathbf{AV}_{i-1}\mathbf{W}_{i-1}^{inv})\cdot \\ &\qquad (\mathbf{V}_{i-1}^T\mathbf{A}^2\mathbf{V}_{i-1}\mathbf{S}_{i-1}\mathbf{S}_{i-1}^T + \mathbf{V}_{i-1}^T\mathbf{AV}_{i-1})\mathbf{S}_i\mathbf{S}_i^T\end{aligned} \tag{4.9}$$

nella ricorrenza

$$\mathbf{V}_{i+1} = \mathbf{AV}_i\mathbf{S}_i\mathbf{S}_i^T + \mathbf{V}_i\mathbf{D}_{i+1} + \mathbf{V}_{i-1}\mathbf{E}_{i+1} + \mathbf{V}_{i-2}\mathbf{F}_{i+1}. \tag{4.10}$$

Per $j < 0$ si utilizzano $\mathbf{W}_j^{inv} = \mathbf{V}_j = \mathbf{0}$ e $\mathbf{S}_j = \mathbf{I}_N$. La verifica della correttezza di queste formule è contenuta in [13].
Con questa ricorrenza il calcolo di ogni $\mathbf{V}_j$ richiede circa lo stesso numero di operazioni anzichè aumentare di complessità col numero delle iterazioni.

4.2.4 Determinazione degli $\mathbf{S}_i$

La matrice $\mathbf{S}_i$ seleziona le righe di $\mathbf{V}_i$ da utilizzare in $\mathbf{W}_i$, in modo che questa sia $\mathbf{A}$-invertibile, abbia rango massimo e in modo che le colonne di $\mathbf{V}_{i-1}$ non usate in $\mathbf{W}_{i-1}$ siano selezionate ora.
Innanzitutto possono essere selezionate solo $\text{rank}(\mathbf{V}_i^T\mathbf{A}\mathbf{V}_i)$ colonne di $\mathbf{V}_i$; se ne fossero selezionate di più la corrispondente $\mathbf{W}_i^T\mathbf{A}\mathbf{W}_i$ non sarebbe invertibile.
Inoltre per prime si vogliono selezionare le colonne non selezionate in $\mathbf{V}_{i-1}$, che corrispondono a $\mathbf{V}_i(\mathbf{I}_N - \mathbf{S}_{i-1}\mathbf{S}_{i-1}^T)$. Se le corrispondenti colonne di $\mathbf{V}_i^T\mathbf{A}\mathbf{V}_i$ sono linearmente dipendenti, mentre sono indipendenti in $\mathbf{V}_i$ l'algoritmo fallisce e va ripetuto con un nuovo vettore aleatorio iniziale.
Montgomery propone il seguente algoritmo per il calcolo di $\mathbf{S}_i$ e $\mathbf{W}_i^{inv}$ in contemporanea. Sia $\mathbf{T} = \mathbf{V}_i^T\mathbf{A}\mathbf{V}_i$ (quindi $\mathbf{W}_i^{inv} = \mathbf{S}_i(\mathbf{S}_i^T\mathbf{T}\mathbf{S}_i)^{-1}\mathbf{S}_i^T$) e si consideri $\mathbf{M} = [\mathbf{T}|\mathbf{I}_N]$, matrice $N \times 2N$ ottenuta giustapponendo $\mathbf{T}$ e $\mathbf{I}_N$, e sia $\mathcal{S} = \emptyset$ l'insieme delle colonne da includere in $\mathbf{S}_i$. Si considerino le colonne di $\mathbf{T}$ numerate come $[c_1, \ldots, c_N]$, con le colonne non selezionate in $\mathbf{S}_{i-1}$ numerate per prime, in modo che vengano incluse per prime in $\mathbf{V}_i$.
L'algoritmo è simile a quello tradizionale per l'inversione di matrice ed è eseguito elaborando le colonne nell'ordine specificato da $c_1, \ldots, c_N$. A differenza dell'inversione tradizionale, nel momento in cui si trova una colonna di $\mathbf{T}$ linearmente dipendente dalle precedenti vengono annullate anche le corrispondenti riga e colonna nel lato destro di $\mathbf{M}$ e non si include il corrispondente indice in $\mathcal{S}$. Invece per le colonne correttamente invertite si inserisce l'indice in $\mathcal{S}$.
All'iterazione successiva è inoltre necessario controllare se $\mathbf{V}_{i+1}$ contiene tutte le colonne scelte tramite $\mathbf{S}_i$ e $\mathbf{S}_{i-1}$.

4.2.5 Costi computazionali

Nella sezione 8 di [13] si stima euristicamente che per N e n grandi, $N \ll n \ll 2^{N^2/2}$, il numero medio di iterazioni necessarie all'algoritmo per convergere sia circa $n/(N - 0.764499780)$. Il valore al denominatore è il valore atteso del rango di una matrice $N \times N$ a coefficienti casuali e equiprobabili in GF(2) e viene utilizzato per i vari $\mathbf{V}_i$, anche se solo il primo è scelto in modo casuale. Nel paper si supporta questa stima con risultati sperimentali.

In ogni iterazione, per determinare il nuovo $\mathbf{V}_{i+1}$ in accordo con le formule (4.9) e (4.10), è necessario calcolare $\mathbf{A}\mathbf{V}_i$, $\mathbf{V}_i^T\mathbf{A}\mathbf{V}_i$ e $\mathbf{V}_i^T\mathbf{A}^2\mathbf{V}_i = (\mathbf{A}\mathbf{V}_i)^T\mathbf{A}\mathbf{V}_i$ a partire da $\mathbf{V}_i$, oltre che a selezionare $\mathbf{S}_i$ e calcolare $\mathbf{W}_i^{inv}$. Infine tramite l'equazione (4.4), alla matrice $\mathbf{X} - \mathbf{Y}$ viene aggiunta la componente relativa a $\mathbf{V}_{i+1}$.
Queste operazioni richiedono:

- il calcolo di $\mathbf{A}\mathbf{V}_i$, ottenibile come già mostrato nella sottosezione 4.2.2 come $\mathbf{B}^T\mathbf{B}\mathbf{V}_i$. In particolare se $\mathbf{B}$ ha d elementi non nulli per colonna e se ogni riga di $\mathbf{V}_i$ è raccolta in una word di N bit del computer quest'operazione richiede $\text{O}(nd)$ operazioni;

- il prodotto interno di matrici $n \times N$ per calcolare $\mathbf{V}_i^T\mathbf{A}\mathbf{V}_i$ e $\mathbf{V}_i^T\mathbf{A}^2\mathbf{V}_i$. Dal momento che le righe sono costituite da un'unica word, è possibile calcolare ogni riga della matrice $N \times N$ risultante in $\text{O}(n)$, per una complessità totale di $\text{O}(nN)$;

- il calcolo di $\mathbf{S}_i$ e $\mathbf{W}_i^{inv}$ come proposto nella sottosezione 4.2.4, che richiede $O(N^2)$ operazioni (trascurabile rispetto a $O(nN)$ del punto precedente);
- alcune operazioni con matrici $N \times N$ per determinare i coefficienti $\mathbf{D}_i$, $\mathbf{E}_i$ e $\mathbf{F}_i$ (anche queste trascurabili).
- quattro prodotti tra matrici $n \times N$ e $N \times N$ e quattro somme di matrici $n \times N$ per il calcolo di $\mathbf{V}_{i+1}$, per una complessità totale di $O(nN)$ (il calcolo di $\mathbf{S}_i\mathbf{S}_i^T$ ha complessità trascurabile rispetto al resto).
- un prodotto interno e il prodotto di $\mathbf{V}_i$ per matrici $N \times N$ per aggiungere la componente i-esima a $\mathbf{X} - \mathbf{Y}$, per una complessità totale di $O(nN)$.

Nel complesso quindi ogni iterazione ha complessità $O(nd + nN)$ e, avendo O(n/N) iterazioni, l'algoritmo ha una complessità computazionale di $O(n^2d/N + n^2)$, contro $O(n^3)$ dell'algoritmo di Gauss. Quindi nel caso di matrici di fattorizzazione, dove $d \ll n$, si ha una complessità circa $O(n^2)$, molto migliore dell'eliminazione Gaussiana, e rimane dello stesso ordine di grandezza se $d \sim n$.

Dal punto di vista dello spazio di memoria utilizzato, sono necessarie le matrici $\mathbf{A}$, $\mathbf{V}_{i+1}$, $\mathbf{V}_i$, $\mathbf{V}_{i-1}$ e $\mathbf{V}_{i-2}$, alcune matrici $N \times N$ e la somma parziale $\mathbf{X} - \mathbf{Y}$. In particolare per $\mathbf{A}$ è sufficente memorizzare $\mathbf{B}$, con consumo di $O(nd)$ word anzichè $O(n^2)$. Per la fase finale è necessario memorizzare $\mathbf{V}_m$, $\mathbf{BV}_m$, $\mathbf{X} - \mathbf{Y}$ e $\mathbf{B}(\mathbf{X} - \mathbf{Y})$, per un totale di $O(n)$ word.

Queste complessità computazionali e spaziali fanno si che a parità di computer utilizzato con il metodo di Lanczos a blocchi si possano risolvere matrici molto maggiori che non con l'eliminazione Gaussiana. Inoltre in [12] si mostra come sia possibile parallelizzare l'algoritmo in modo non particolarmente complicato, utilizzando un insieme di computer dove uno funge da master e gli altri da slave. In questo modo è possibile risolvere matrici di dimensioni maggiori della memoria dei singoli computer (eccetto che per il computer principale che deve conoscere tutta la matrice), e con uno speed-up lineare rispetto al numero di computer usati.
Nel Capitolo 5 si mostrano alcune risoluzioni di matrici di fattorizzazione tramite l'algoritmo confrontate con l'algoritmo di Gauss e si citano i risultati ottenuti nei record di fattorizzazione da questo metodo.

Capitolo 5

Risultati Sperimentali

In questo capitolo si esporrano i risultati sperimentali di alcune fattorizzazioni. I candidati utilizzati negli esperimenti sono stati creati dalla moltiplicazione di due numeri casuali di dimensioni opportune, risultati pseudo-primi forti rispetto 30 basi casuali (e quindi assumibili come primi).
Più precisamente, un candidato di n cifre ha i fattori di $\lfloor n \rfloor$ e $\lceil n \rceil$ cifre. I candidati sono stati fattorizzati più volte, abilitando ogni volta una diversa funzionalità dell'algoritmo tra quelle presentate nella sezione della teoria, in modo da poterle confrontare direttamente.
Le prove sono state eseguite su un Intel Core2 Quad 2.83 Ghz, 12 Mb Cache L2, con 4 Gb di RAM e sistema operativo Linux Ubuntu 09.10 con kernel 2.6.31-15. I programmi utilizzati sono stati scritti in C e implementano anche tutta la parte aritmetica necessaria a gestire i numeri interi a precisione arbitraria.

5.1 Quadratic Sieve

Il Quadratic Sieve è stato implementato con tutte le caratteristiche descritte, eccezion fatta per la Double Large Prime Variation. Le varie migliorie (in particolare l'aggiunta del setaccio di numeri negativi, la Small Prime Variation, la Large Prime Variation e l'uso di un moltiplicatore) sono state abilitate singolarmente in modo da poter confrontare ciascuna variante con l'algoritmo di base e poterne apprezzare il contributo. Nelle Tabelle 5.1 e 5.2 sono confrontati i tempi dei vari esperimenti per candidati di 40, 45, 50 e 55 cifre decimali. Le voci delle tabelle sono rispettivamente:

B**:** *bound* della *factor base*;

$|FB|$**:** il numero di elementi presenti nella *factor base*;

*neg***:** l'uso o meno di numeri negativi nel setaccio;

*spv***:** limite usato nella Small Prime Variation;

*lpv***:** limite usato nella Large Prime Variation;

*mu***:** moltiplicatore usato;

compl/parz: relazioni complete dirette e relazioni complete ottenute dalle parziali;

interv: intervallo di setaccio esaminato;

t_{sieve}: tempo impiegato per completare il setaccio (nel formato $min : sec.decimi$).

Per ogni candidato si è preso come tempo di riferimento il migliore ottenuto al variare di B senza alcuna miglioria attiva (riga evidenziata in **grassetto**). Nell'ultima colonna è specificato per ogni altro tentativo la percentuale in più o in meno impiegata rispetto al tempo di riferimento.

5.1.1 Commenti

Dal momento che ogni numero presenta caratteristiche specifiche, dovute alle sue dimensioni e alla sua fattorizzazione, nel seguito si commenteranno perlopiù i risultati medi, sfruttando le particolarità di ciascun numero solo quando sarà opportuno.
L'utilizzo di un B scorretto può essere molto penalizzante, ad esempio nel primo candidato l'uso di $B = 10000$, circa il 33% minore di quello ottimo, dà un incremento del 20% nel tempo di setaccio, negli altri casi, quando il B è entro il $10 - 15\%$ da quello migliore trovato il tempo di setaccio non differisce per più del 5%. Quindi è sufficiente trovare un B in un intorno di quello ottimo.
Il setaccio di $f(X)$ in un intorno di $\sqrt{n}$ anzichè per $X \geq \sqrt{n}$ (e quindi l'introduzione di -1 nella *factor base*) riduce i tempi di setaccio tra il 13 e il 20%. Questo è dovuto alla riduzione di un fattore 2 delle dimensioni medie degli $f(x)$ considerati, con conseguente aumento della probabilità che questi siano B-smooth.
L'applicazione della Small Prime Variation (cioè il non setaccio per i numeri della *factor base* minori di una certa soglia, in questo caso 35) ha diminuito tra il 12 e il 34% i tempi di setaccio.
La Large Prime Variation riduce tra il 50 e il 60% i tempi ed è interessante vedere come intervanga soprattutto nella parte finale della raccolta di relazioni, quanto per il paradosso del compleanno si ha che molte parziali hanno lo stesso resto. Nella sottosezione 5.1.3 si mostrerà più in particolare quest'aspetto.
Il moltiplicatore può aver un'influenza molto grande, come per esempio nel candidato di 45 cifre, dove da solo riduce i tempi di setaccio più che tutte le altre tecniche messe assieme (si vedano penultima e terzultima riga della Tabella 5.1). Nel candidato di 50 cifre invece il moltiplicatore 1 è già la scelta ottima, non è quindi applicabile. Nella sottosezione 5.1.2 si mostrerà nel dettaglio il calcolo della previsione del contributo fornito dal moltiplicatore.
Nel complesso le varie tecniche utilizzate assieme possono ridurre il tempo di fattorizzazione fino a meno di un decimo del tempo originario; anche nel caso peggiore (in cui non si può usufruire del moltiplicatore) ci si può attendere una velocizzazione di quasi tre volte rispetto all'algoritmo di base.

Candidato di 40 cifre (129 bit)									
B	$\|FB\|$	neg	spv	lpv	mu	compl/parz	interv	t_{sieve}	%
10000	631	no	1	0	1		[0, 1.170.767.872]	0:12.7	+19.81%
15000	**912**	**no**	**1**	**0**	**1**		**[0, 473.792.512]**	**0:10.6**	
20000	1161	no	1	0	1		[0, 301.137.920]	0:11.2	+5.66%
15000	912	si	1	0	1		[-231.702.528, 231.702.528]	0:08.4	-20.75%
15000	912	no	35	0	1		[0, 481.329.152]	0:07.0	-33.96%
15000	912	no	1	$8B$	1	530/743	[0, 216.432.640]	0:04.7	-55.66%
15000	912	no	1	$16B$	1	479/453	[0, 183.042.048]	0:05.0	-52.83%
15000	912	no	1	$32B$	1	433/499	[0, 157.876.224]	0:05.1	-51.89%
15000	870	no	1	0	37		[0, 188.612.608]	0:05.7	-46.23%
15000	870	si	35	$8B$	37	627/376	[-73.138.176, 73.138.176]	0:01.2	-88.68%
Candidato di 45 cifre (149 bit)									
B	$\|FB\|$	neg	spv	lpv	mu	compl/parz	interv	t_{sieve}	%
30000	1608	no	1	0	1		[0, 1.4186.315.776]	2:28.5	+23.41%
40000	**2097**	**no**	**1**	**0**	**1**		**[0, 7.625.736.192]**	**1:57.9**	
50000	2558	no	1	0	1		[0, 5.138.350.080]	2:03.8	+5.00%
40000	2097	si	1	0	1		[-3.639.541.760, 3.639.541.760]	1:41.9	-13.57%
40000	2097	no	35	0	1		[0, 8.099.954.688]	1:43.7	-12.04%
40000	2097	no	1	$4B$	1	1180/937	[0, 3.395.092.480]	0:47.2	-59.97%
40000	2097	no	1	$8B$	1	1101/1165	[0, 3.020.914.688]	0:45.7	-61.24%
40000	2097	no	1	$16B$	1	1014/1138	[0, 2.587.754.496]	0:46.5	-60.56%
40000	2097	no	1	$32B$	1	1009/1305	[0, 2.555.215.872]	0:49.3	-58.18%
40000	2181	no	1	0	2		[0, 845.512.704]	0:44.2	-62.51%
40000	2097	si	35	$8B$	1	1291/826	[-2.075.426.816, 2.075.426.816]	0:45.8	-61.15%
40000	2181	si	35	$8B$	2	1573/824	[-322.437.120, 322.437.120]	0:09.6	-91.86%

Tabella 5.1: QS: candidati di 40 e 45 cifre.

Candidato di 50 cifre (165 bit)									
B	$\|FB\|$	neg	spv	lpv	mu	compl/parz	interv	t_{sieve}	%
50000	2530	no	1	0	1		[0, 12.682.067.968]	3:31.2	+1.64%
60000	**2973**	**no**	**1**	**0**	**1**		**[0, 9.526.312.960]**	**3:27.8**	
70000	3416	no	1	0	1		[0, 7.567.769.600]	3:32.3	+2.17%
60000	2973	si	1	0	1		[-4.469.063.680, 4.469.063.680]	2:53.4	-16.55%
60000	2973	no	35	0	1		[0, 11.834.851.328]	2:31.6	-27.04%
60000	2973	no	1	$8B$	1	1734/1382	[0, 4.473.225.216]	1:37.2	-53.22%
60000	2973	no	1	$16B$	1	1611/1228	[0, 3.936.911.360]	1:36.7	-53.46%
60000	2973	no	1	$32B$	1	1568/1495	[0, 3.742.072.832]	1:38.4	-52.65%
60000	2973	si	35	$16B$	1	1971/1022	[-3.155.132.416, 3.155.132.416]	1:20.0	-61.50%
Candidato di 55 cifre (182 bit)									
B	$\|FB\|$	neg	spv	lpv	mu	compl/parz	interv	t_{sieve}	
80000	3894	si	35	$32B$	5	2208/1877	[-25.068.896.256, 25.068.896.256]	9:12.6	

Tabella 5.2: QS: candidati di 50 e 55 cifre.

5.1.2 Contributo del moltiplicatore

Il polinomio $f(X)$ per un generico k e per $X = x$ grande vale circa:

$$f(x) = (x + \lceil\sqrt{kn}\rceil)^2 - kn \approx 2x\lceil\sqrt{kn}\rceil \approx e^{\frac{1}{2}\ln\, kn + \ln\, 2x}.$$

Nella sezione 3.6 si è stabilito che, tenendo conto dei contributi di fattorizzazione della *factor base* si può considerare la dimensione media di $f(x)$ come:

$$f(x) \approx e^{\frac{1}{2}\ln\, kn + \ln\, 2x - F(k,n)}. \tag{5.1}$$

Per il candidato di 45 cifre, il calcolo della funzione $F(n,k)$ presentata nella sezione 3.6 dà valore massimo per $k = 2$, $F(n,2) \approx 11.008$, mentre per $k = 1$ (cioè senza uso di moltiplicatore) $F(n,1) \approx 8.240$. Si vuole calcolare una stima del rapporto tra i tempi di esecuzione attesi della fase di setaccio al variare dei moltiplicatori. Nel caso dell'uso del moltiplicatore, tenendo presente che si usa $B = 40000$, e stimando come dimensione media per $x \approx B^2$, il valore medio degli N trattati è:

$$N \approx e^{\frac{1}{2}\ln\, n + 2\ln\, B + \ln\, 2 - 11.008} \approx e^{63.3797}$$

con conseguente u:

$$u = \frac{\ln N}{\ln B} \approx \frac{63.3797}{\ln 40000} \approx 5.981$$

In questo caso $u^u \approx 44262$, ci si aspetta quindi di trovare una relazione ogni circa 45000 valori setacciati. Rieseguendo il calcolo per $k = 1$ si ha $u \approx 6.242$ e $u^u \approx 92218$; date queste frequenze di occorrenza per le relazioni ci si aspetta quindi usando il moltiplicatore $k = 2$ di impiegare il 48% del tempo rispetto al setaccio usando $k = 1$. Sperimentalmente si è riscontrato che il setaccio usando il moltiplicatore ha impiegato impiegato il 37.5% rispetto al tempo senza moltiplicatore.
Le stime asintotiche differiscono di circa un ordine di grandezza dalla frequenza reale delle relazioni, ma il loro rapporto è utilizzabile per avere una prima stima sul rapporto tra i tempi di setaccio impiegati.

5.1.3 Andamento della ricerca di relazioni

La Tabella 5.3 mostra i dati relativi al numero di relazioni trovare al passare del tempo nella fattorizzazione del candidato di 55 cifre; in particolare per ogni tempo sono specificate quante relazioni complete sono state trovate in modo diretto, il numero di relazioni parziali trovate, il numero di complete ricombinate da queste e quante relazioni complete sono state trovate in totale.
La fattorizzazione è stata eseguita con $B = 80000$, moltiplicatore $k = 5$ e limite per la LPV di $32B$. Si noti che le parziali sono ricombinate solo periodicamente (dopo averne trovate $2B$ la prima volta e circa ogni 1000 le successive), quindi l'andamento relativo a queste è a gradini anzichè abbastanza continuo come per le dirette. Dai dati si vede chiaramente come all'aumentare delle parziali raccolte aumenti di molto il numero di complete ottenibili da queste, tanto che alla fine si ottengono più relazioni in modo indiretto che diretto.
In particolare a 6 minuti dall'inizio della setaccio si sono ottenute 1430 relazioni in modo

diretto e 926 da 8179 parziali. A fine setaccio, pur trovando solo altre 4463 parziali si sono ottenute altre 951 complete da queste, contro le 778 ottenute in modo diretto. Si noti come all'aumentare delle dimensioni degli $f(x)$ diminuisca la frequenza con cui si trovano relazioni B-smooth e parziali. Si tenga presente che già dopo i primi 30 secondi $x \approx 1.7 \cdot 10^9$ e a fine fattorizzazione $x \approx 25 \cdot 10^9$.
Nei grafici 5.1 e 5.2 è evidente come relazioni complete dirette e parziali aumentano in modo meno veloce all'aumentare di x, mentre le ricombinate aumentano sempre più velocemente con l'aumentare delle parziali, tanto che compensano la minor frequenza di ritrovamento delle dirette e le complete nel totale crescono quasi linearmente.

tempo (s.)	dirette	parziali	ricombinate	totali
0	0	0	0	0
30	332	1871	0	332
60	510	2743	0	510
90	652	3618	0	652
120	784	4383	0	784
150	898	5065	0	898
180	1005	5747	0	1005
210	1112	6327	0	1112
240	1233	6980	0	1233
270	1338	7585	0	1338
300	1430	8179	926	2356
330	1540	8810	926	2466
360	1627	9356	926	2553
390	1745	9889	1275	3020
420	1835	10387	1275	3110
450	1927	10902	1462	3389
480	2008	11425	1462	3470
510	2094	11936	1662	3756
540	2185	12447	1662	3847
552	2208	12642	1877	4085

Tabella 5.3: Andamento delle relazioni trovate nel setaccio per il candidato di 55 cifre.

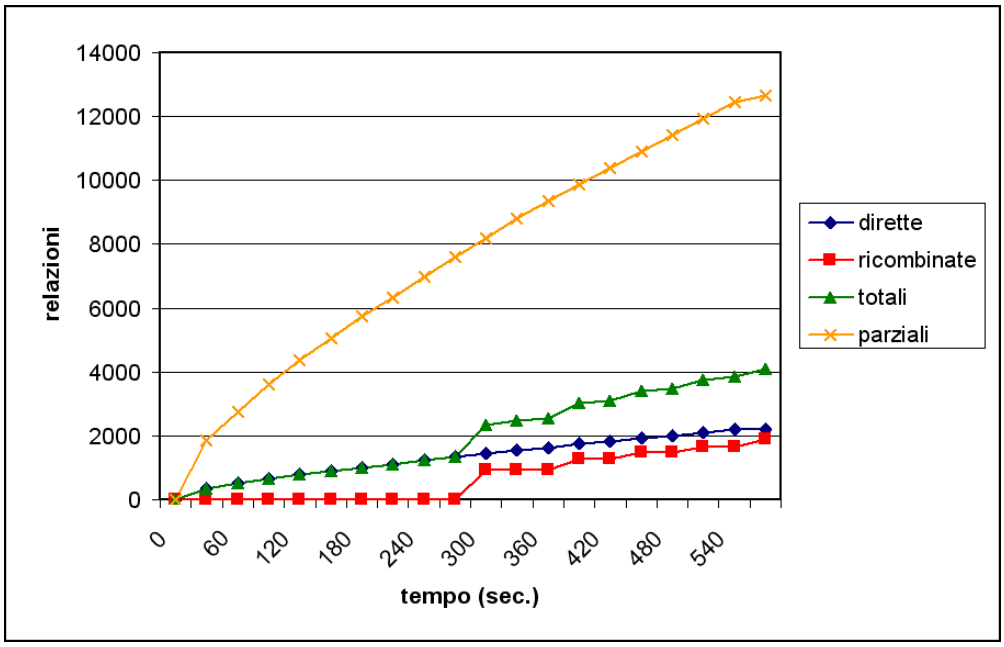

Figura 5.1: Grafico relativo ai dati della Tabella 5.3.

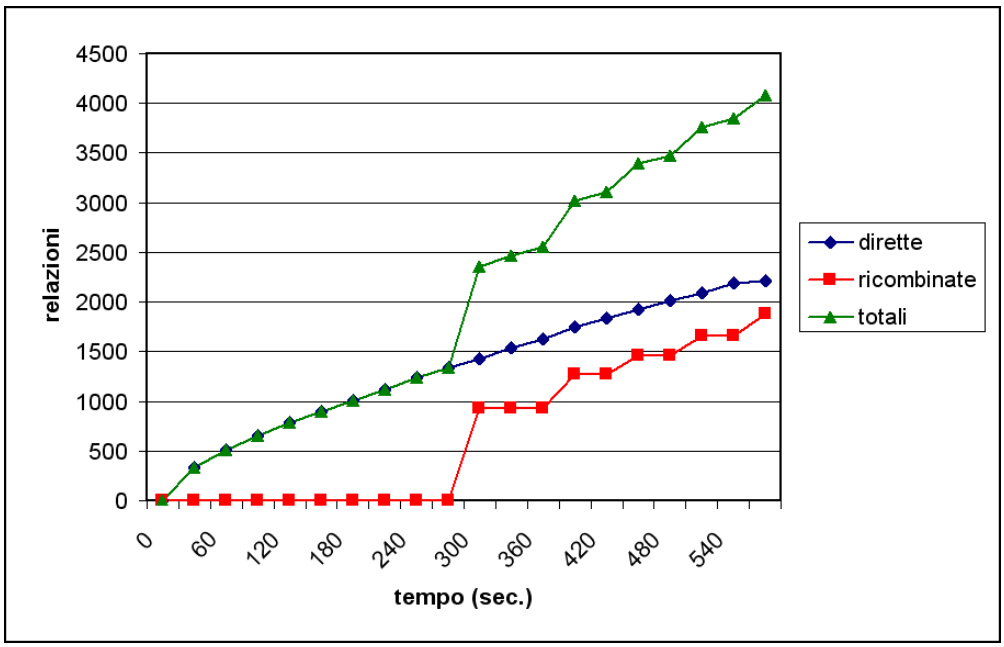

Figura 5.2: Grafico relativo ai dati della Tabella 5.3. Dettaglio dell'andamento di dirette e ricombinate.

5.2 Multiple Polynomials Quadratic Sieve

Per quanto riguarda il MPQS, gli esperimenti sono rivolti a scoprire quali sono i valori migliori di B ed M per questa implementazione. I candidati tra le 40 e le 55 cifre sono gli stessi del QS, in modo da poter fare poi un confronto dei tempi tra l'uso o meno di polinomi multipli. Il candidato di 65 cifre non è stato setacciato sistematicamente, ma solo per avere un riferimento cronometrico tra i candidati di 60 e 70 cifre.
Un M minore rende minori i residui degli $f(x)$, ma allo stesso tempo rende necessario creare molti polinomi, operazione molto costosa, specie all'aumentare delle dimensioni del candidato. Quindi al crescere di n si ha una crescita dell'M utilizzato ed è stato trovato in modo sperimentale quale è il valore che minimizza il tempo di setaccio. Negli esperimenti si è trovato $M \approx 10B$, che è quindi il tempo necessario ad ammortizzare la creazione di un polinomio. Per quanto riguarda il parametro B, comune ai due algoritmi, nel MPQS si può usare un valore molto minore rispetto al QS, in genere circa la metà. Infatti ora le valutazioni di $f(X)$ non crescono indefinitamente e un certo B mantiene le stesse prestazioni per tutta la durata del setaccio, senza correre il rischio di non essere più efficace quando i residui crescono. Per quanto riguarda LPV e moltiplicatore, valgono le stesse considerazioni fatte per il QS.
Le seguenti Tabelle riportano i tempi di setaccio al variare di B, della soglia lpv, dell'uso o meno del moltiplicatore e dell'M utilizzato. La voce *poly* specifica quanti polinomi sono stati generati nel corso del setaccio. A lato in alcuni esperimenti è presente l'indicazione delle dimensioni dei blocchi di setaccio. Dove non specificata è di 50000 elementi per array (la stessa usata anche nel QS).
Date le dimensioni consistenti delle *factor base* usate nei numeri con più di 50 cifre è conveniente sfruttare array più grandi (maggiori di B), in modo da ammortizzare effettivamente con il setaccio il processo di ritrovamento del primo elemento dell'array divisibile per un numero della *factor base*. L'uso di array grandi non è penalizzante nel caso in esame data la grande quantità di memoria cache L2 a disposizione.
Nella Tabella 5.4 sono riassunti i parametri da usare al variare delle dimensioni dei candidati per ottenere all'incirca le migliori prestazioni possibili con il programma presentato. Con quest'implementazione di MPQS non sono stati testati numeri maggiori di 70 cifre, che sono stati invece riservati al solo SIQS. Per un confronto tra prestazioni di MPQS e QS, si veda la sottosezione 5.3.2.

$\log(n)$	B	lpv	M	dimblocco
$\leq$45	20000	8-16	250k	250k
50	30000	32	500k	250k
60	50000	64	1M	250k
70	150000	128	4M	400k

Tabella 5.4: MPQS: tabella riassuntiva dei parametri ottimi trovati.

Candidato di 40 cifre (129 bit)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	
5000	343	0	1		500k	296	0:02.0	
7500	481	0	1		500k	151	0:01.3	
10000	631	0	1		500k	108	0:01.3	
15000	912	0	1		500k	65	0:01.4	
7500	481	0	1		1M	103	0:01.6	
7500	481	0	1		250k	266	0:01.5	
7500	481	$8B$	1	310/392	500k	85	0:01.2	
7500	481	$16B$	1	256/245	500k	69	0:01.1	
7500	481	$16B$	37	291/266	500k	69	0:00.4	
Candidato di 45 cifre (149 bit)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	
10000	609	0	1		500k	3233	0:20.5	
15000	858	0	1		500k	1793	0:12.7	
20000	1118	0	1		500k	1176	0:10.0	
25000	1381	0	1		500k	863	0:09.1	
30000	1608	0	1		500k	704	0:08.7	
35000	1842	0	1		500k	590	0:08.7	
40000	2097	0	1		500k	494	0:08.8	
30000	1608	0	1		250k	1051	0:08.0	
30000	1608	0	1		1M	422	0:09.1	
30000	1608	$16B$	1	846/783	250k	538	0:06.7	
30000	1608	0	2		250k	280	0:03.6	
30000	1608	$16B$	2	1024/832	250k	153	0:02.6	
Candidato di 50 cifre (165 bit)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	*dimblocco*
30000	1607	0	1		250k	2164	0:16.0	
40000	2093	0	1		250k	1662	0:12.8	
50000	2530	0	1		250k	1405	0:16.5	
30000	1607	0	1		500k	1367	0:12.5	
40000	2093	0	1		500k	1009	0:11.1	
50000	2530	0	1		500k	814	0:13.4	
30000	1607	0	1		1M	868	0:15.9	
40000	2093	0	1		1M	634	0:11.9	
50000	2530	0	1		1M	539	0:15.0	
40000	2093	0	1		500k	879	0:09.3	250k
40000	2093	$16B$	1	1314/1061	500k	540	0:8.8	
40000	2093	$32B$	1	1297/1164	500k	523	0:09.5	
40000	2093	$32B$	1	1168/1160	500k	435	0:07.8	250k

Tabella 5.5: MPQS: candidati di 40, 45 e 50 cifre.

Candidato di 55 cifre (182 bit)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	*dimblocco*
40000	2162	0	1		500k	6994	1:07.7	
45000	2416	0	1		500k	6027	1:01.1	
50000	2644	0	1		500k	5358	1:16.8	
45000	2416	0	1		250k	9129	1:03.3	
45000	2416	0	1		1M	3636	0:55.8	
45000	2416	0	1		2M	2093	1:08.8	
45000	2416	0	1		1M	3220	0:43.9	200k
45000	2416	$16B$	1	1283/1153	1M	1676	0:28.6	200k
45000	2416	$32B$	1	1169/1267	1M	1510	0:27.6	200k
45000	2416	$64B$	1	1130/1306	1M	1437	0:28.2	200k
45000	2342	0	5		1M	3220	0:40.9	200k
45000	2416	$32B$	5	976/1386	1M	1175	0:20.9	200k
Candidato di 60 cifre (198 bit) (dimblocco=150k/200k)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	
45000	2361	0	1		1M	10002	2:13.8	
50000	2598	0	1		1M	8877	2:00.0	
55000	2815	0	1		1M	7779	2:13.1	
60000	3031	0	1		1M	6962	2:14.1	
50000	2598	0	1		750k	11146	2:03.4	
50000	2598	0	1		1.5M	6295	1:49.2	
50000	2598	0	1		2M	5294	2:44.9	
50000	2598	$16B$	1	1330/1288	1M	4223	1:26.7	
50000	2598	$32B$	1	1208/1410	1M	3835	1:20.0	
50000	2598	$64B$	1	1206/1412	1M	3791	0:57.7	
50000	2598	$128B$	1	1216/1797	1M	3789	1:21.2	
Candidato di 65 cifre (216 bit) (dimblocco=300k)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	
100000	4838	$128B$	1	2202/2847	1.8M	12349	4:42.3	
Candidato di 70 cifre (232 bit) (dimblocco=300k/400k)								
B	$\|FB\|$	lpv	mu	compl/parz	M	poly	t_{sieve}	
200000	9164	$128B$	1	4767/4484	6M	19006	34:56.6	
200000	8986	$128B$	2	5031/4138	8M	12619	28:40.2	
200000	8986	$128B$	2	4985/4021	6M	16489	29:35.9	
200000	8986	$256B$	2	4841/4165	6M	15512	28:50.0	
200000	8986	$128B$	2	5116/4066	4M	21615	26:37.1	
200000	8986	$256B$	2	4993/4013	4M	20588	26:16.2	
175000	7944	$128B$	2	4148/3819	6M	17360	27:53.1	
150000	6911	$128B$	2	3449/3620	4M	24892	23:52.2	
125000	5868	$128B$	2	2642/3273	4M	27800	23:48.4	
150000	6911	$128B$	2	3589/3469	3M	31371	25:58.8	

Tabella 5.6: MPQS: candidati di 55, 60, 65 e 70 cifre.

5.2.1 Andamento della ricerca di relazioni (MPQS)

Come per il QS, si vedrà ora l'andamento della frequenza con cui si trovano relazioni complete dirette, parziali e complete ricombinate. Dal momento che ora le dimensioni di $f(x)$ rimangono circa costanti per tutta la durata del setaccio, ci si aspetta che le relazioni parziali e le complete dirette crescano linearmente. Le ricombinate dovrebbe cresce sempre più velocemente, man mano che aumentano il numero di relazioni parziali raccolte.
Nella Tabella 5.7 si mostra quante relazioni complete dirette, ricombinate e totali, e quante relazioni parziali si sono raccolte al passare dei minuti della fattorizzazione del candidato di 70 cifre con $B = 150000$, $M = 4000000$ e $k_{lpv} = 128B$.
Come si può vedere dai grafici 5.3 e 5.4 che la rappresentano, effettivamente le complete dirette e le parziali crescono linearmente nel corso di tutto il setaccio, mentre le ricombinate aumentano la frequenza con l'aumentare delle parziali raccolte. Il processo di ricombinazione delle parziali avviene dopo aver raccolto $2B$ parziali la prima volta, ogni $B/2$ nuove parziali nel seguito, dimezzando questa soglia ogni volta che la ricombinazione da più di 300 complete ricombinate, in modo che l'algoritmo sfrutti quasi al meglio le parziali raccolte. Si può notare come nel momento in cui il processo di ricombinazione inizia ad essere eseguito con una certa regolarità (circa dopo 10 minuti, con soglia di ricombinazione $B/4$), si ha una flessione delle retta del numero di relazioni raccolte. In particolare si passa da circa 135 complete e 1900 relazioni parziali al minuto a circa 100 complete e 1400 parziali (il rapporto tra i due rimane invariato).
Questa differenza di prestazioni non è comunque imputabile al processo di ricombinazione, eseguito circa una volta al minuto per un tempo inferiore al secondo, responsabile quindi al più di un rallentamento del setaccio di 1/60. Probabilmente la causa va ricercata nella presenza di un processore più veloce negli altri (come si può vedere anche nei grafici della sottosezione 5.3.1).

tempo (min.)	dirette	ricombinate	totali	parziali
0	0	0	0	0
1	131	0	131	1930
2	277	0	277	3812
3	435	0	435	5935
4	560	0	560	7775
5	700	0	700	9639
6	850	0	850	11416
7	983	365	1348	13292
8	1114	514	1628	15185
9	1255	698	1953	17033
10	1363	698	2061	18372
11	1451	931	2382	19702
12	1545	931	2476	20944
13	1648	1176	2824	22195
14	1736	1176	2912	23540
15	1837	1492	3329	24797
16	1902	1642	3544	26142
17	2012	1789	3801	27480
18	2100	1968	4068	28766
19	2213	2163	4376	30328
20	2299	2323	4622	31668
21	2381	2679	5060	33016
22	2472	2853	5325	34344
23	2591	3053	5644	35662
23,8	2642	3273	5915	36752

Tabella 5.7: Andamento delle relazioni trovate nel setaccio per il candidato di 70 cifre.

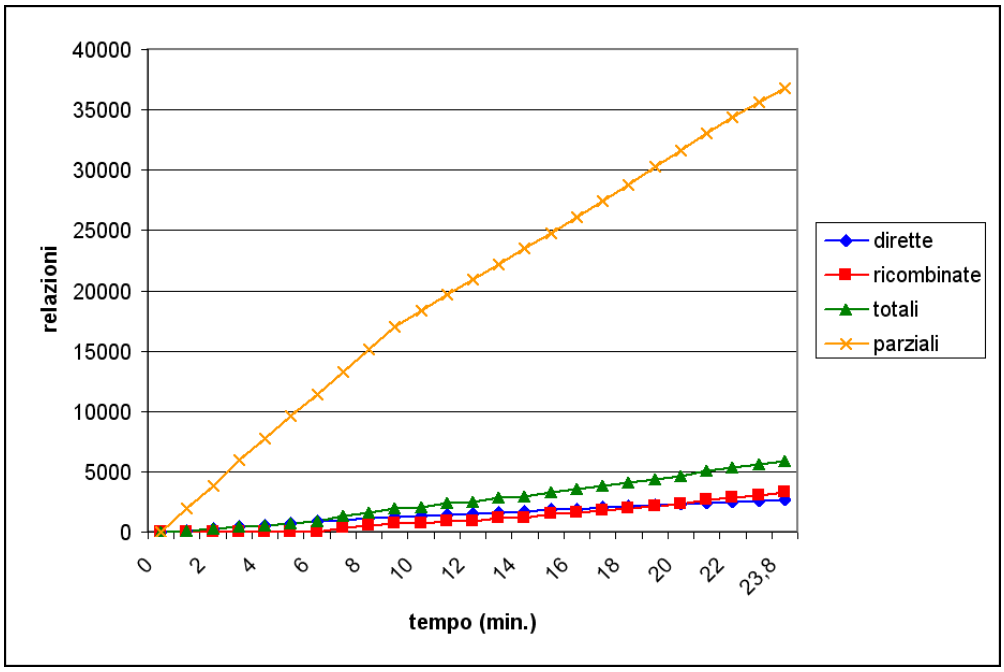

Figura 5.3: Grafico relativo ai dati della Tabella 5.7.

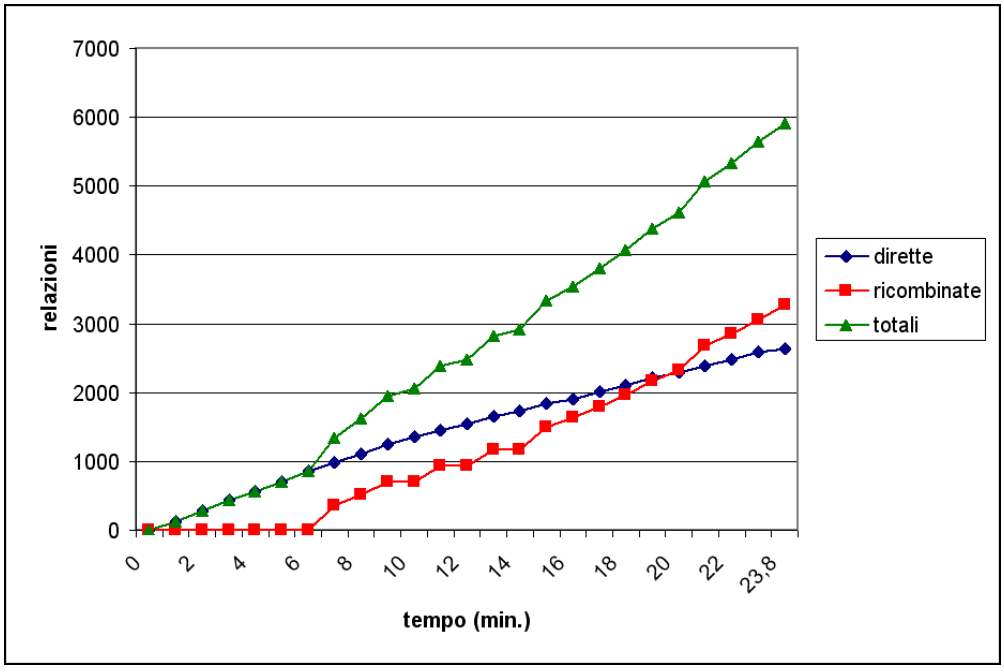

Figura 5.4: Grafico relativo ai dati della Tabella 5.7. Dettaglio dell'andamento di dirette e ricombinate.

5.3 Self Initializing Quadratic Sieve

In questa sezione si presenteranno i risultati del setaccio per gli stessi candidati usati nella sezione precedente, stavolta però tramite il SIQS. I candidati di 65, 75 e 80 cifre non sono stati testati in modo esaustivo. I parametri per questi sono stati estrapolati dalle analisi dei parametri dei precedenti candidati, senza fare una ricerca a tentativi completa.
Il principale vantaggio del SIQS rispetto al MPQS è il fatto di poter riutilizzare quasi integralmente l'inizializzazione di un polinomio per tutti 2^{k-1} polinomi con lo stesso a (composto di k fattori distinti della *factor base*). In questo modo è possibile abbassare il limite dell'M utilizzabile senza risentire dei troppi costi di creazione dei polinomi come avviene nel MPQS.
Negli esperimenti si è provato a utilizzare, tra gli altri, anche la configurazione risultata ottima nel MPQS. In questo modo è possibile vedere quanto tempo si risparmia nella creazione di polinomi. Ad esempio per il candidato di 60 cifre il MPQS impiega 57.7 secondi, usando 3791 polinomi. Il SIQS impiega 40 secondi usando circa lo stesso numero di polinomi, 3983. I coefficienti a usati dal SIQS per numeri di 60 cifre sono costituiti però di 7 fattori, che permettono di usare lo stesso a per $2^6 = 64$ polinomi. In questo caso quindi il SIQS ha dovuto inizializzare solo 63 polinomi circa. Questo mostra quanto pesante sia l'operazione di creazione di un polinomio; infatti si può stimare che il MPQS passi 20 secondi su 60 a crearne e questo peso cresce con l'aumentare delle dimensioni del candidato. Viceversa per candidati piccoli, al diminuire di k il SIQS tende a comportarsi esattamente come il MPQS.
Il numero di fattori di a, k_a, usato negli esperimenti è stato calcolato come $\lfloor \frac{\lceil \log_{10} n \rceil}{\log_{10} 2000} \rfloor$, in quanto si cercano tutti fattori attorno a 2000. Scegliere fattori più piccoli aumenterebbe il numero di polinomi con cui si può usare lo stesso a, ma ridurrebbe anche la probabilità che le valutazioni siano B-smooth, in quanto i fattori di k_a dividono per una sola radice le valutazioni anzichè per due come gli altri elementi della factor base. In particolare k_a vale 4 per candidati di 45 cifre, 5 per quelli di 50, e così via fino a 8 per quelli di 70 e 10 per quelli di 80 cifre.
Il SIQS sfrutta il basso costo di inizializzazione dei suoi polinomi per usare un M minore, trovando quindi relazioni più frequentemente e potendo inoltre sfruttare un B minore (così come il MPQS poteva fare nei confronti del QS). Per il SIQS vale $M \approx B - 2B$, circa un decimo del MPQS. Il numero di polinomi creati invece è in media dieci volte maggiore a quello del MPQS, fatto che lascia ipotizzare che in entrambi gli algoritmi si spende in realtà circa metà del tempo a creare polinomi e l'altra metà nel setaccio vero e proprio. Il vero vantaggio del SIQS non si ha quindi nel ridurre il costo di creazione dei polinomi, ma impiegando il basso costo per abbassare M e ridurre così il tempo atteso di esecuzione. Sperimentalmente si vede che il SIQS è circa due volte più veloce del MPQS; si veda la sottosezione 5.3.2 per dettagli.
Data la maggior velocità si è usato il SIQS anche per fattorizzare candidati di 75 e 80 cifre. Le impostazioni dei parametri per questi numeri sono state fatte stimando il tempo atteso sulla base del numero di relazioni raccolte in 5 minuti con diverse configurazioni (ipotizzate a partire dalle configurazioni ottime per numeri di 60 e 70 cifre).

Candidato di 40 cifre (129 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
7500	489	500k	$16B$	37	257/252	46	0:00.41
7500	489	50k	$16B$	37	302/282	260	0:00.24
7500	489	50k	$8B$	37	302/220	250	0:00.19
Candidato di 45 cifre (149 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
30000	1687	250k	$16B$	2	1036/840	158	0:02.3
20000	1168	50k	$16B$	2	723/590	755	0:01.1
15000	903	60k	$12B$	2	552/536	949	0:01.0
15000	903	50k	$16B$	2	489/490	897	0:00.9
Candidato di 50 cifre (165 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
40000	2093	500k	$32B$	1	1105/1110	453	0:06.4
40000	2093	200k	$32B$	1	1098/1132	900	0:05.1
40000	2093	150k	$32B$	1	1187/1087	1168	0:04.9
40000	2093	100k	$32B$	1	1247/866	1603	0:04.7
35000	1834	100k	$32B$	1	1029/972	1758	0:04.2
30000	1607	100k	$32B$	1	809/846	1937	0:03.8
30000	1607	60k	$32B$	1	890/818	3012	0:03.7
25000	1374	60k	$32B$	1	789/725	3562	0:03.4
20000	1107	60k	$32B$	1	509/621	4222	0:03.2
Candidato di 55 cifre (182 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
45000	2342	1M	$32B$	5	981/1409	1156	0:15.0
30000	1653	300k	$64B$	5	659/1059	4134	0:12.5
30000	1653	250k	$64B$	5	644/1079	4741	0:12.1
30000	1653	200k	$64B$	5	646/1093	5635	0:11.4
30000	1653	150k	$64B$	5	688/1086	7275	0:11.2
30000	1653	100k	$64B$	5	682/991	9294	0:13.2
40000	2113	200k	$64B$	5	936/1197	5635	0:10.6
40000	2113	150k	$64B$	5	887/1247	5271	0:10.1
40000	2113	100k	$64B$	5	966/1284	7323	0:09.8
40000	2113	100k	$32B$	5	993/1276	7822	0:09.8

Tabella 5.8: SIQS: candidati di 40, 45, 50 e 55 cifre.

Candidato di 60 cifre (198 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
50000	2598	1M	$64B$	1	1245/1373	3983	0:40.0
50000	2598	300k	$64B$	1	1208/1410	9158	0:27.2
50000	2598	250k	$64B$	1	1320/1389	12998	0:36.9
50000	2598	350k	$64B$	1	1256/1408	7987	0:35.1
40000	2113	300k	$64B$	1	1011/1123	12043	0:31.5
45000	2361	300k	$64B$	1	1092/1289	10329	0:28.8
55000	2815	300k	$64B$	1	1381/1529	8478	0:26.7
60000	3031	300k	$64B$	1	1567/1708	8059	0:27.1
Candidato di 65 cifre (216 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
100000	4838	300k	$128B$	1	2465/2398	53422	2:43.9
Candidato di 70 cifre (232 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
125000	5868	4M	$128B$	2	2988/2900	34059	26:59.5
200000	8986	300k	$100B$	2	6303/3320	260338	19:36.8
175000	7944	300k	$100B$	2	5153/2811	253199	17:19.7
150000	6911	300k	$100B$	2	4185/2800	254050	17:01.5
125000	5868	350k	$128B$	2	3183/2750	197017	16:18.3
125000	5868	300k	$128B$	2	3183/2705	254085	16:08.9
125000	5868	250k	$128B$	2	3423/2465	298851	16:24.4
Candidato di 75 cifre (249 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
200000	9082	300k	$128B$	3	4305/4797	1161856	1h 14:07.3
Candidato di 80 cifre (266 bit)							
B	$\|FB\|$	M	lpv	mu	compl/parz	#poly	t_{sieve}
300000	13165	300k	$128B$	2	9525/3705	3324584	4h 20:00.7
250000	11150	300k	$128B$	2	7696/3302	3297191	3h 57:31.4

Tabella 5.9: SIQS: candidati di 60, 65 e 70 usati anche per il MPQS e candidati di 75 e 80 cifre.

5.3.1 Andamento della ricerca di relazioni (SIQS)

Nel grafico 5.5 si presenta l'andamento delle relazioni trovate nella fattorizzazione del candidato di 80 cifre. Queste sostanzialmente confermano quanto mostrato nella relativa sezione del MPQS, dove si mostra come ora il ritrovamento di parziali e dirette è lineare, mentre si intuisce molto meglio l'andamento delle ricombinate, che crescono sempre più velocemente con l'aumentare delle parziali raccolte (fino al limite a crescere con la stessa velocità delle parziali quando tutti i resti minori della soglia di raccolta saranno stati raccolti).
Un segmento di retta, lungo circa 15 minuti sembra confermare che uno dei processori lavora ad una velocità leggermente superiore agli altri, cosicchè falsa leggermente l'andamento lineare durante il suo turno, come riscontrato nel grafico delle relazioni trovate con il MPQS.
Il fatto che siano state trovate solo un quarto delle relazioni tramite le parziali è segno che il parametro k_{lpv} o B non sono ottimi.

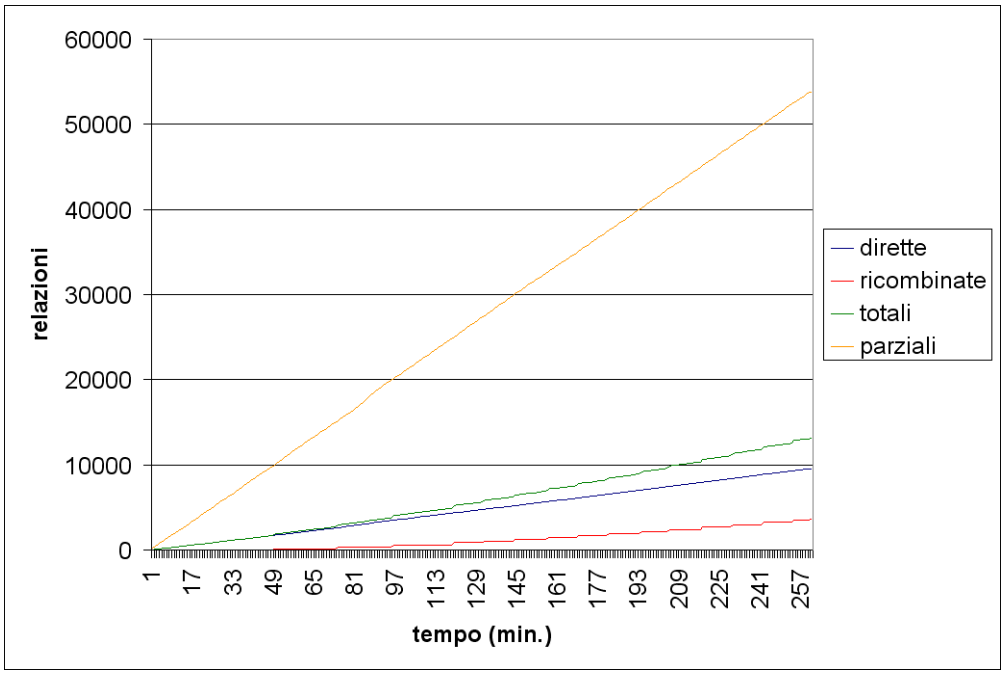

Figura 5.5: Grafico relativo al setaccio per il candidato di 80 cifre.

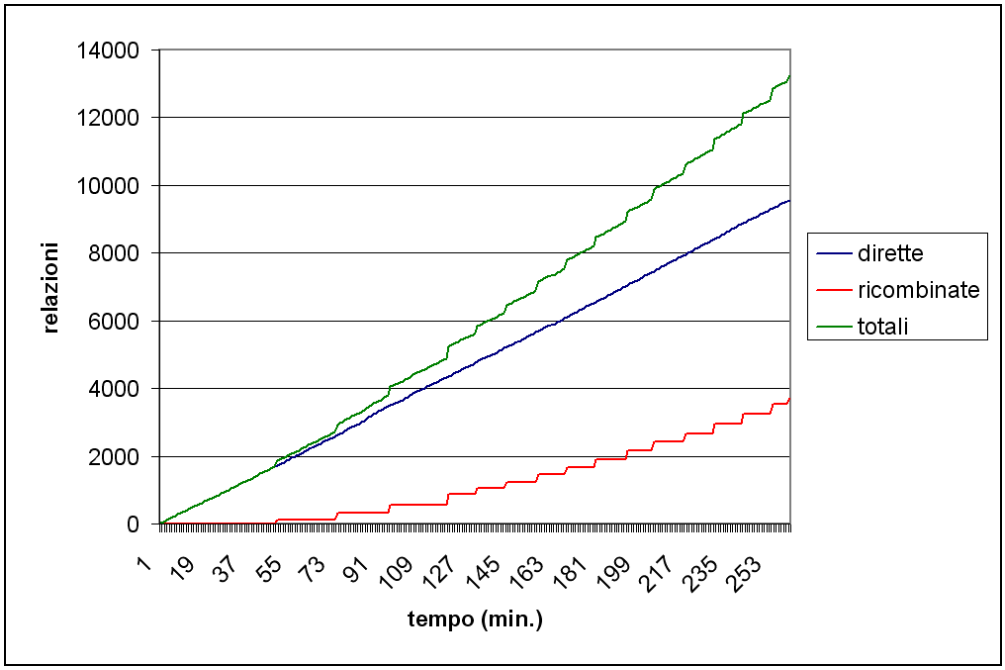

Figura 5.6: Grafico relativo al setaccio per il candidato di 80 cifre. Dettaglio dei relazioni complete dirette, ricombinate e totali.

5.3.2 Confronto di prestazioni tra QS, MPQS e SIQS

Nella Tabella 5.10 e nel grafico 5.7 si mostra un confronto dei tempi di setaccio dei vari candidati. Per ogni candidato si è considerata la prestazione migliore dell'algoritmo.

algoritmo	129	149	165	182	198	216	232
QS base	0:10.6	1:57.9	3:27.8				
QS migl	0:01.2	0:09.6	1:20.0	9:12.6			
MPQS	0:00.4	0:02.6	0:07.8	0:20.9	0:57.7	4:42.3	23:52.2
SIQS	0:00.2	0:00.9	0:03.2	0:09.8	0:26.7	2:43.9	16:08.9

Tabella 5.10: Tempi nelle fattorizzazioni dei vari candidati.

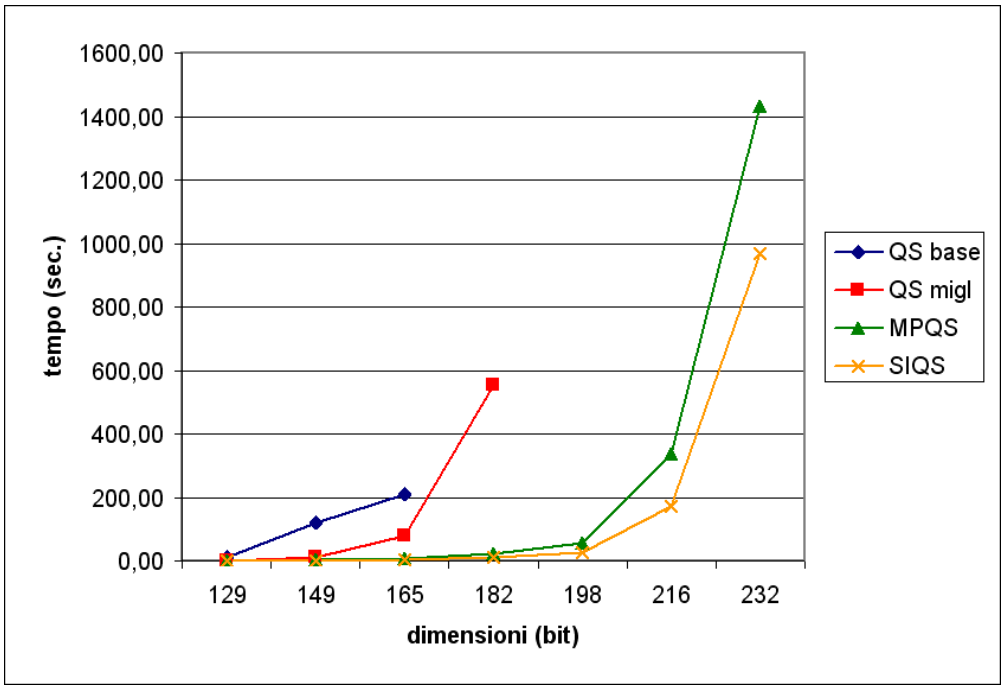

Figura 5.7: Grafico relativo ai tempi di setaccio per i candidati.

Nei grafici successivi si mostra un confronto (in un grafico a scala logaritmica) tra funzioni esponenziali, polinomiali e la $L(X)$, la complessità euristica del QS. Come si può immaginare, il grafico di ogni polinomio X^k è la versione riscalata di un fattore k del grafico del polinomio X. L'esponenziale cresce linearmente, mentre la $L(X)$ cresce con un tasso maggiore rispetto ai polinomi, ma non lineare. Infatti non esiste ϵ tale che $L(X) \sim e^{\epsilon X}$. Questo grafico da un'idea della differenza di qualità tra un algortimo con complessità esponenziale e gli algoritmi di fattorizzazioni illustrati in questa tesi.

Nei grafichi 5.9 e 5.10 si mostra un confronto tra andamento di $L(X)$ e i tempi di esecuzione del SIQS, sia in scala lineare che logaritmica. I tempi di esecuzione sono stati moltiplicati per un fattore $k = 3.8 \cdot 10^{12}$ e come si può vedere hanno approssimativamente

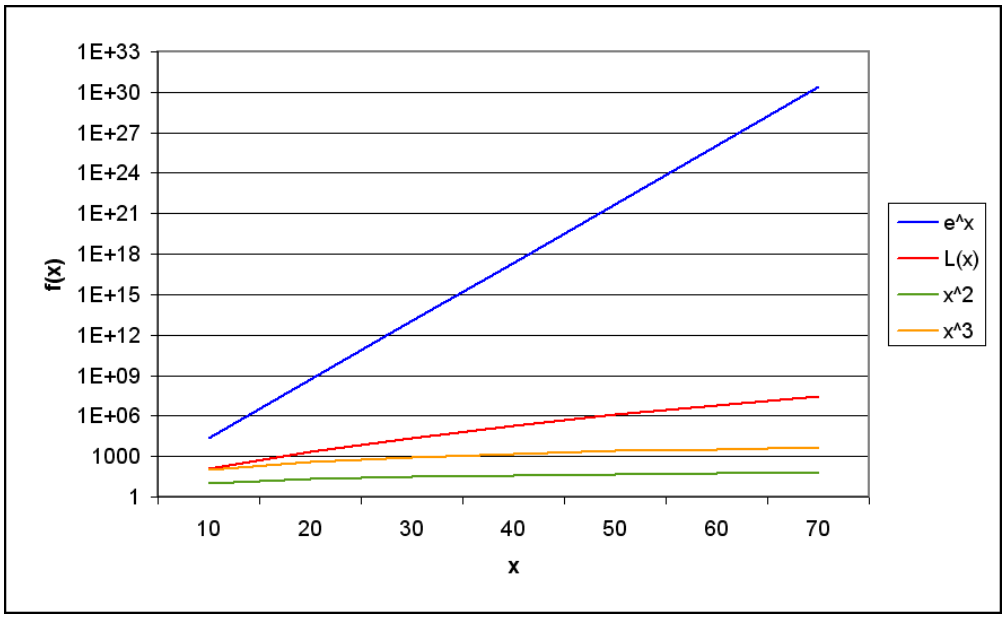

Figura 5.8: Confronto in scala logaritmica tra complessità lineare, quadratica, esponenziale ed $L(X)$.

l'andamento di $L(X)$. Il discostamento dal valore ideale per candidati piccoli (sotto le 60 cifre) è dovuto al fatto che le complessità sono state analizzate asintoticamente, quindi le approssimazioni sono sempre più imprecise man mano che si diminuiscono le dimensioni dei numeri trattati.

bit	t_{SIQS}	$3.8 \cdot 10^{12} \cdot t_{SIQS}$	$L(x)$
129	0.2	$7.60 \cdot 10^{11}$	$7.48 \cdot 10^{10}$
149	0.9	$3.42 \cdot 10^{12}$	$7.22 \cdot 10^{11}$
165	3.2	$1.22 \cdot 10^{13}$	$4.03 \cdot 10^{12}$
182	9.8	$3.72 \cdot 10^{13}$	$2.32 \cdot 10^{13}$
198	26.7	$1.01 \cdot 10^{14}$	$1.13 \cdot 10^{14}$
216	163.9	$6.23 \cdot 10^{14}$	$6.28 \cdot 10^{14}$
232	968.9	$3.68 \cdot 10^{15}$	$2.74 \cdot 10^{15}$
249	4447.3	$1.37 \cdot 10^{16}$	$1.25 \cdot 10^{16}$
266	14251.4	$5.42 \cdot 10^{16}$	$5.45 \cdot 10^{16}$

Tabella 5.11: Confronto tra i tempi di setaccio del SIQS e l'andamento di $L(X)$.

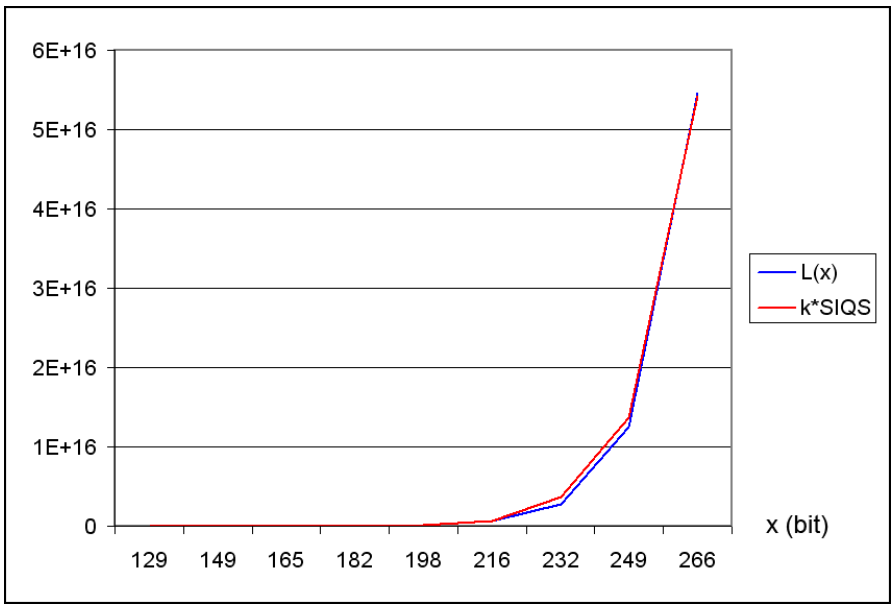

Figura 5.9: Confronto in scala lineare tra tempi misurati del SIQS e la funzione $L(X)$.

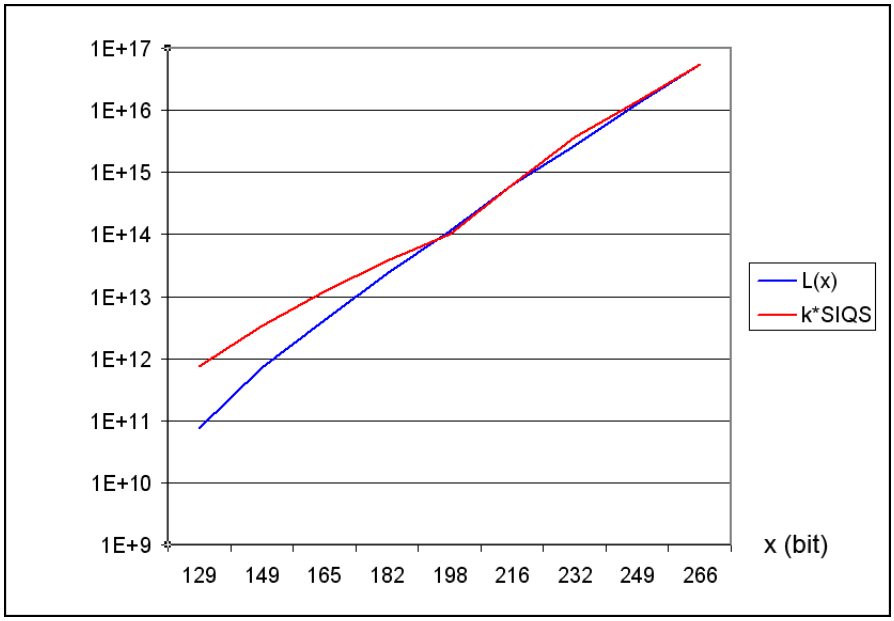

Figura 5.10: Confronto in scala logaritmica tra tempi misurati del SIQS e la funzione $L(X)$.

5.4 Eliminazione Gaussiana

In questa sezione si mostrerà la complessità nella ricerca di righe linearmente dipendenti nella matrice ricavata dalle relazioni trovate nella fase di setaccio al variare delle tecniche adottate.
In particolare si userà la tradizionale eliminazione Gaussiana, ciò che cambia nei vari esperimenti è il preprocessing applicato alle matrici. In particolare si discuterà dell'algoritmo per la diminuzione delle dimensioni delle matrici di sezione 4.1.1 e della riduzione di complessità computazionale invertendo l'ordine degli elementi dei vettori.
Nella Tabella 5.12, con M_{orig} si indica la matrice originale trovata dal processo di setaccio, mentre M_{rid} indica la matrice ridotta con il metodo di Pomerance. Inoltre il suffisso *inv* indica le matrici dove le componenti dei vettori rappresentano in ordine decrescente gli elementi della *factor base* (altrimenti sono considerati in ordine crescente).
La colonna *spurie* indica quante colonne con meno di due 1 sono state eliminate ad ogni iterazione dell'algoritmo di diminuzione della matrice; nella colonna $rid\%$ invece si specifica la percentuale di elementi per vettore e di vettori eliminata con questo processo.
È possibile vedere dai dati in tabella che all'aumentare di B aumenta sempre più la riduzione delle dimensioni della matrice dovuta all'algoritmo. L'implementazione testata lavora in modo efficente per B relativamente piccoli se confrontati con quelli trovati in letteratura, in genere si usa $B = 350000$ per candidati di 70 cifre e $B = 900000$ per quelli di 80. Con questi valori i vettori vengono ridotti anche a solo un terzo delle dimensioni originarie, con uno speedup di 27 volte usando la riduzione gaussiana sulla matrice ottenuta, piuttosto che su quella originale. Con le riduzioni di circa il 30% ottenute con le configurazioni di parametri utilizzate negli esperimenti si ottiene una velocizzazione di 3 volte.
Comunque bisogna notare che un altro fattore che permette di accelerare notevolmente la risoluzione della matrice è la semplice inversione degli elementi del vettore. Infatti in questo modo durante l'eliminazione Gaussiana si mantiene sparsa per molto più tempo la matrice. Nelle immagini presenti in questa sezione è mostrata sotto forma di immagine la matrice ottenuta per il candidato di 40 cifre, fattorizzato con $B = 5000$, che corrisponde ad usare una *factor base* di 369 elementi. Nelle immagini, ogni pixel è un elemento, in particolare ogni pixel nero rappresenta un 1 ed ogni pixel bianco rappresenta uno 0. Nell'Immagine 5.11 si vede la matrice ottenuta tramite SIQS, mentre nell'Immagine 5.13 quella ottenuta tramite MPQS.
Nella matrice relativa al SIQS si vedono molti trattini tra le colonne 150 e 350; questi sono dovuti al fatto che data una famiglia di polinomi con un certo a, da questa si ottengono molte relazioni consecutive tutte contenenti tra gli altri anche i fattori di a. Dove i trattini sono segmentati, si è in presenza di relazioni ottenute tramite LPV. In quest'ultime ci sono trattini più brevi, dovuti al fatto che la prima relazione con un certo resto è unita tramite XOR a tutte quelle con lo stesso resto. Nella matrice ottenuta tramite MPQS sono presenti sono trattini ottenuti tramite LPV.

Candidato di 50 cifre (165 bit)								
B	M_{orig}	t_{orig}	$t_{orig-inv}$	M_{rid}	rid %	spurie	t_{rid}	$t_{rid-inv}$
30000	1627x1607	0:01.04	0:00.18	1497x1477	8%	102 22 4 2	0:00.8	0:00.17
40000	2113x2093	0:02.26	0:00.4	1892x1872	10.5%	167 41 11 1 1	0:01.62	0:00.29
50000	2550x2530	0:03.97	0:00.8	2218x2198	13%	255 65 11 1	0:02.6	0:00.43
60000	2993x2973	0:06.62	0:00.86	2484x2464	17%	366 96 33 11 3	0:03.69	0:00.77
80000	3873x3853	0:16.13	0:01.58	3000x2980	22.6%	605 175 59 25 7 1 1	0:06.69	0:00.83
Candidato di 60 cifre (198 bit)								
B	M_{orig}	t_{orig}	$t_{orig-inv}$	M_{rid}	rid %	spurie	t_{rid}	$t_{rid-inv}$
50000	2618x2598	0:04.33	0:00.94	2484x2464	5.1%	117 13 4	0:03.7	0:00.83
60000	3051x3031	0:07.16	0:01.38	2771x2751	9.2%	226 46 6 2	0:05.18	0:01.07
80000	3916x3896	0:016.8	0:02.27	3406x3386	13%	389 86 26 8 1	0:10.39	0:01.65
100000	4812x4792	0:35.3	0:04.12	3960x3940	17.7%	662 146 33 9 2	0:17.56	0:02.39
150000	6982x6962	2:13.2	0:11.2	4789x4769	31.4%	1560 431 133 52 14 2 1	0:48.82	0:03.41
Candidato di 65 cifre (216 bit)								
B	M_{orig}	t_{orig}	$t_{orig-inv}$	M_{rid}	rid %	spurie	t_{rid}	$t_{rid-inv}$
80000	3959x3939	0:20.1	0:03.4	3647x3627	7.9%	256 40 13 3	0:13.2	0:02.8
100000	4858x4838	0:36.8	0:05.7	4306x4286	11.4%	431 90 20 6 3 2	0:23.6	0:04.2
120000	5752x5732	1:08.8	0:08.6	4769x4749	17.1%	750 168 45 11 6 3	0:34.6	0:05.4
Candidato di 70 cifre (232 bit)								
B	M_{orig}	t_{orig}	$t_{orig-inv}$	M_{rid}	rid %	spurie	t_{rid}	$t_{rid-inv}$
125000	5888x5868	1:49.7	0:10.3	5044x5024	14.4%	656 136 40 10 2	0:42.5	0:06.6
150000	6931x6911	2:20.6	0:20.3	5634x5614	18.7%	967 249 62 17 2	1:04.2	0:12.5
200000	9006x8986	6:18.3	0:32.8	6216x6196	31%	1951 544 177 69 38 11	2:10.6	0:14.5
250000	11020x11000		1:03.4	6974x6954	36.7%	2809 807 276 102 32 16 4	3:42.3	0:18.6
Candidato di 80 cifre (266 bit)								
250000	11173x11150			7239x7219	35.2%			≈25.0
300000	13215x13165			7960x7940	39.8%			≈30.0

Tabella 5.12: Tempi di risoluzione delle matrici per vari candidati.

I grafici 5.12 e 5.14 rappresentano la distribuzione di 1 nelle colonne. Come ci si può attendere gli elementi minori della *factor base* hanno una maggior quantità di 1, fatto che indica che dividono più spesso gli $f(x)$. Inoltre nel SIQS ci sono dei picchi per elementi tra il 2000 e il 3000, per il fatto che vanno a formare gli a e dividono spesso con grado 1 gli $f(x)$.

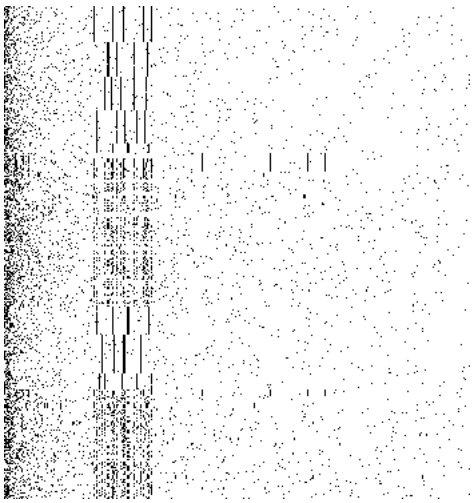

Figura 5.11: Matrice per il candidato di 40 cifre ottenuta tramite SIQS.

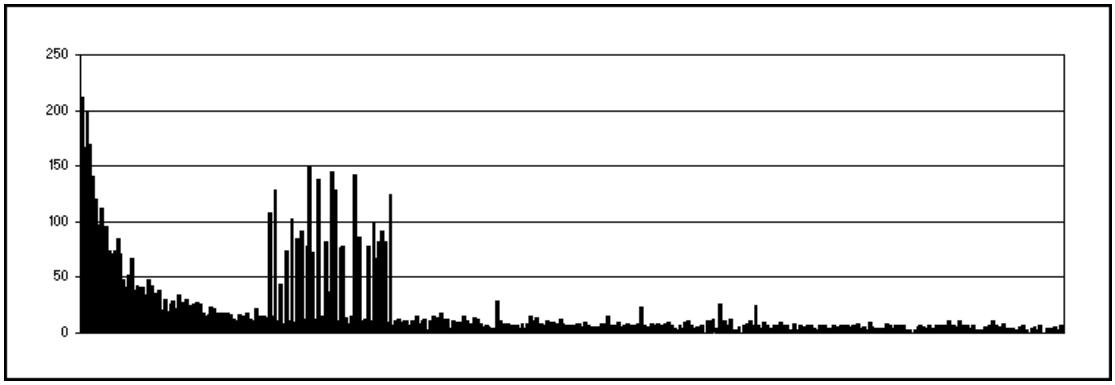

Figura 5.12: Numero di 1 nelle colonne della matrice ottenuta tramite SIQS.

La matrice in esame, di dimensioni 369×349, è molto piccola rispetto a quelle con decine di migliaia di elementi ottenuta per fattorizzazioni di numeri grandi, altrimenti sarebbe possibile apprezzare come molti numeri nella parte terminale della *factor base* in realtà non raccolgano alcun 1.

Nell'Immagine 5.15 si mostra la matrice ottenuta tramite SIQS una volta ordinata per l'eliminazione Gaussiana e nell'Immagine 5.16 come si presenta dopo la risoluzione. Se si invertono i vettori, in modo da avere le colonne corrispondenti agli elementi della *factor base* maggiori all'inizio, si ottengono le matrici nelle Immagini 5.17 e 5.18.

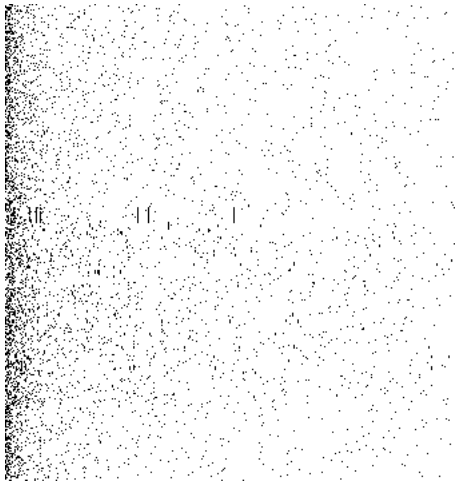

Figura 5.13: Matrice per il candidato di 40 cifre ottenuta tramite MPQS.

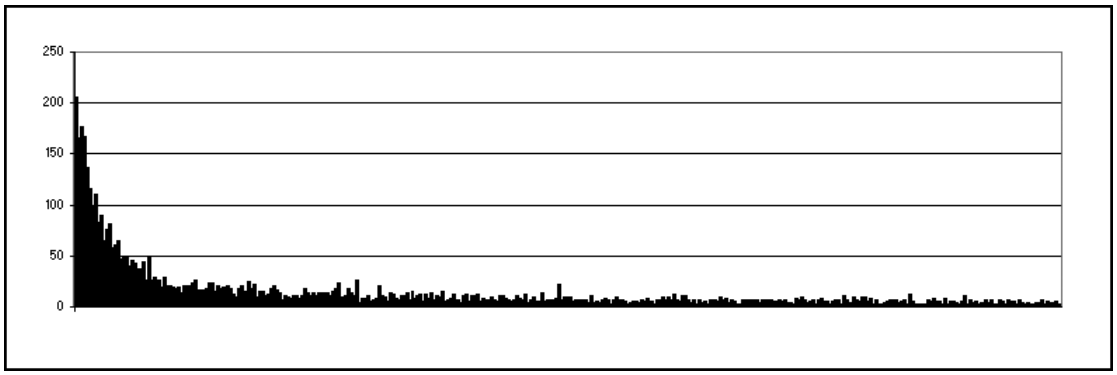

Figura 5.14: Numero di 1 nelle colonne della matrice ottenuta tramite MPQS.

Nelle matrici con vettori non invertiti, nel processo di triangolazione, fin dalle prime fasi ogni riga dovrà essere sottratta a molte righe. In questo esempio, con una matrice di lato maggiore 369, la prima colonna contiene 210 elementi non nulli e quindi la prima riga sarà sottratta alle successive 209.
Con vettori invertiti, nelle prime fasi le righe dovranno essere sottratte in media ad una sola altra riga (infatti le prime colonne risultano molto sparse, ma essendo sopravvissute al processo di riduzione hanno almeno 2 uno per colonna). In questo modo la matrice rimane sparsa per molto tempo (cioè ogni colonna elaborata continua ad avere pochi 1 per molto tempo). Dalla matrice risolta si vede che solo da due terzi della matrice le colonne cominciano ad essere abbastanza dense. Questo significa che la prima parte della matrice (quando bisogna lavorare con vettori di lunghezza quasi completa) richiede poche sottrazioni per riga e l'ultima parte (quando ormai i vettori hanno la prima metà nulla e quindi trascurabile) è da risolvere con la stessa complessità del metodo a vettori non invertiti.

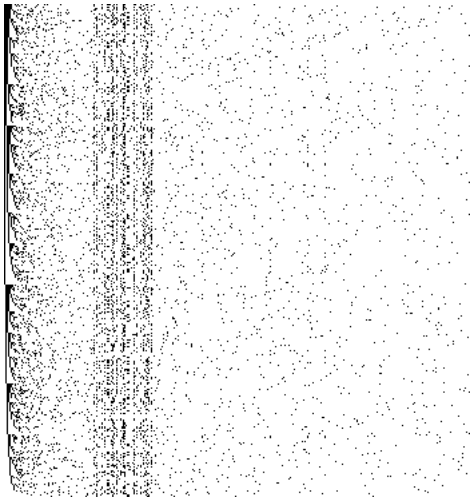

Figura 5.15: Matrice ordinata prima dell'eliminazione Gaussiana.

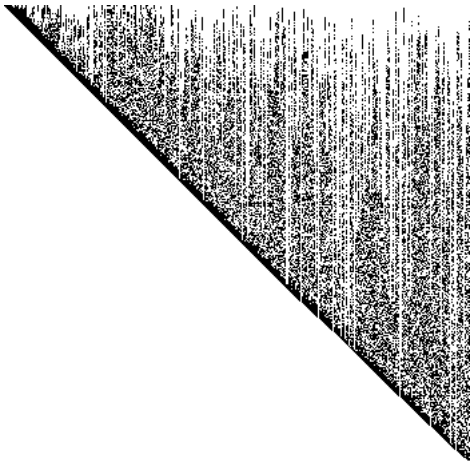

Figura 5.16: Matrice dopo l'eliminazione Gaussiana.

Come si può vedere nella Tabella 5.12, per matrici di una certa importanza questo fa risparmiare anche un fattore 15 nel tempo di risoluzione (che presumibilmente aumenta con l'aumentare delle dimensioni della matrice).
Si noti che alcune colonne della matrice completa sono bianche. Queste sono le colonne tolte dall'algoritmo di riduzione delle dimensioni della matrice proposto da Pomerance. In Figura 5.19 si mostra la matrice ridotta tramite questo e in Figura 5.20 la stessa una volta risolta.

Questi dati mostrano come, nonostante le dimensioni enormi delle matrici considerate, con tecniche che sfruttano la struttura sparsa è possibile ridurre di molto sia le dimensioni che la complessità di risoluzione. Come esempio si cita Arjen Lenstra, che nel

Figura 5.17: Matrice invertita e ordinata prima dell'eliminazione Gaussiana.

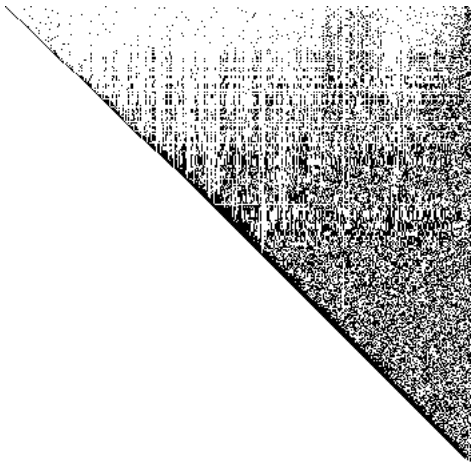

Figura 5.18: Matrice invertita dopo l'eliminazione Gaussiana.

1994 ha usato questi metodi per risolvere la matrice creata con il SIQS per fattorizzare il numero RSA-129. Essa era originariamente 524339×569466 ed è stata ridotta a una matrice 188614×188160, risolta in 45 ore tramite eliminazione Gaussiana. Per i successivi record di fattorizzazione si sono adottate tecniche più avanzate dell'eliminazione Gaussiana, ma la fase di riduzione rimane ancora molto importante.

5.5 Algoritmo di Lanczos a blocchi

Per ultimi, proponiamo i risultati della risoluzione delle matrici di fattorizzazione tramite il metodo di Lanczos a blocchi. Gli esperimenti sono stati eseguiti su matrici ottenute

Figura 5.19: Matrice ridotta e invertita, ordinata prima dell'eliminazione Gaussiana.

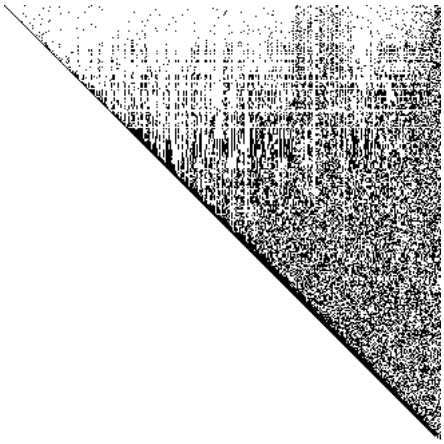

Figura 5.20: Matrice ridotta e invertita dopo l'eliminazione Gaussiana.

dagli stessi candidati usati per l'eliminazione Gaussiana, con lo stesso valore di B e utilizzando $N = 64$.
Il metodo di Lanczos non modifica la matrice, permettendo di memorizzarla in modo compatto, sfruttandone la sparsità. Per ogni riga è sufficiente salvare una lista con gli elementi non nulli. Invece nell'eliminazione Gaussiana man mano che si procede con la riduzione la matrice diventa sempre più densa e quindi questa va memorizzata completamente. È quindi necessario assegnare un bit ad ogni elemento, anche se nullo. Nella Tabella 5.13 si considerano alcuni valori tipici dell'algoritmo di Lanczos:

- numero di cicli impiegati nella realizzazione ($cicli_r$);

- numero di cicli previsto dalla teoria ($cicli_t = n/N - 0.764499780$);

- numero di dimensioni dei $\mathbf{W}_i$ creati durante l'esecuzione. In tutti i casi i valori compresi tra 60 e 64 sono i $\mathbf{W}_i$ creati per $i < m$ e i valori minori sono le dimensioni di $\mathbf{V}_m$;

- il numero medio di dimensioni dei $\mathbf{W}_i$ nella realizzazione E[dim] (da confrontare con N-0.764499780 = 63.23550022, il rango atteso di matrici casuali $N \times N$ a coefficienti binari equiprobabili). Si noti come quando sono richiesti molti cicli il valore atteso del grado tenda a quello teorico.

Nella Tabella 5.14 si mette a confronto lo spazio per salvare la matrice nel metodo di Lanczos ($byte_{lanczos}$) e nell'eliminazione Gaussiana ($byte_{gauss}$). I valori si rifanno solo allo spazio occupato dalla matrice, anche se in realtà il metodo di Lanczos richiede la memorizzazione di alcune altre matrici $n \times N$ e quello di Gauss di una matrice $n \times n$ per tener traccia di come vengono combinate le righe della matrice di fattorizzazione. Nonostante questo i valori sono comunque significativi in quanto gli ordini di grandezza sono gli stessi. In particolare si ha O(n^2) per Gauss e O(nd) in Lanczos. In Tabella si riporta anche il numero di elementi non nulli $elem$ della matrice e il numero medio di elementi per riga d, nonchè un confronto tra i tempi di risoluzione della matrice tramite l'algoritmo di Lanczos ($t_{lanczos}$) e quello di Gauss (t_{gauss}).

Candidato di 50 cifre (165 bit)				
B	dim matrix	$cicli_r/cicli_t$	$dim\ \mathbf{W}_i$	E[dim]
30000	1438x1418	23/22.74	17/1 63/11 64/11	63.5
40000	1865x1845	30/29.49	9/1 60/1 61/1 62/2 63/13 64/12	63.17
60000	2389x2369	38/37.78	24/1 62/5 63/16 64/16	63.30
80000	2867x2847	45/45.33	57/1 61/1 62/3 63/21 64/19	63.32
Candidato di 60 cifre (198 bit)				
B	dim matrix	$cicli_r/cicli_t$	$dim\ \mathbf{W}_i$	E[dim]
60000	2818x2798	44/44.56	15/1 61/1 62/2 63/27 64/13	63.21
80000	3424x3404	54/54.14	43/1 62/8 63/18 64/27	63.36
100000	4036x4016	64/63.82	21/1 61/1 62/5 63/27 64/30	63.36
150000	4980x4960	79/78.75	13/1 62/8 63/33 64/37	63.37
Candidato di 70 cifre (232 bit)				
B	dim matrix	$cicli_r/cicli_t$	$dim\ \mathbf{W}_i$	E[dim]
150000	5779x5759	92/91.39	2/1 61/1 62/13 63/42 64/35	63.22
200000	6786x6766	107/107.31	54/1 61/2 62/15 63/40 64/49	63.28
250000	7543x7523	119/119.28	57/1 61/3 62/14 63/53 64/48	63.24

Tabella 5.13: Parametri del metodo di Lanczos a blocchi nelle varie esecuzioni.

Già questi dati mostrano come il metodo di Lanczos abbia requisiti temporali e spaziali molto inferiori all'eliminazione Gaussiana, nonostante nell'implementazione non siano state implementate tutte le operazioni in modo ottimo (rispetto a quanto proposto in letteratura). Proponiamo inoltre dei dati di risultati sperimentali forniti in [13] per matrici di dimensione considerevole. Nel Giugno del 1994, nel corso della fattorizzazione di $(12^{151} - 1)/11$ (162 cifre decimali) tramite *Special Number Field Sieve*, presso l'Am-

sterdam Academic Computing Center SARA, su un Cray C90 si è risolta una matrice 828.077 × 833.017 con 26.886.496 elementi non nulli con il metodo di Lanczos, usando $N = 64$. Questo ha richiesto 330 Mb di memoria e 3.4 ore di calcolo. Nello stesso periodo e nello stesso laboratorio è stata risolta una matrice 1.284.719 × 1.294.861 con 38.928.220 elementi non nulli, nata dalla fattorizzazione di un fattore di 105 cifre di $3^{367} - 1$ tramite *General Number Field Sieve*. Le risorse richieste ammontarono a 480 Mb e 7.3 ore.

Se fosse stata utilizzata la riduzione Gaussiana strutturata, stimando la riduzione a un terzo delle righe e colonne nella fase di prepocessing, sarebbero stati necessari rispettivamente 9 e 22 Gb di memoria per memorizzare le matrici dense e un tempo di calcolo dell'ordine di una settimana.

Nell'Agosto 1999, sempre presso il SARA, durante la fattorizzazione di RSA-155, si è fattorizzata una matrice 6.699.191 × 6.711.336, con 417.132.631 elementi non nulli in 224 ore di calcolo, usando 2 Gb di un Cray C916. Una simile matrice densa, risolta tramite eliminazione Gaussiana, richiederebbe qualcosa come 500 Gb di memoria e mesi di calcolo.

L'attuale record di risoluzione di matrice spetta all'algoritmo di Wiedemann, utilizzato nella fattorizzazione del RSA-768 per risolvere una matrice 192.796.550 × 192.795.550 con 27.797.115.920 elementi non nulli, avvenuta in modo distribuito presso più centri di ricerca, richiedendo 119 giorni di calcolo (si veda [15]).

Candidato di 50 cifre (165 bit)							
B	dim matrix	$byte_{lanczos}$	$byte_{gauss}$	$elem$	d	$t_{lanczos}$ [msec.]	t_{gauss} [msec.]
30000	1438x1418	151.220	256.143	34.929	24.29	100	170
40000	1865x1845	190.908	431.747	43.997	23.59	170	290
60000	2389x2369	264.292	709.533	61.295	25.65	270	770
80000	2867x2847	327.112	1.022.802	76.044	26.52	380	830
Candidato di 60 cifre (198 bit)							
B	dim matrix	$byte_{lanczos}$	$byte_{gauss}$	$elem$	d	$t_{lanczos}$ [msec.]	t_{gauss} [msec.]
60000	2818x2798	325.804	988.061	75.815	26.9	540	1.070
80000	3424x3404	399.380	1.459.908	92.997	27.16	770	1.650
100000	4036x4016	485.892	2.029.603	113.401	28.1	1.080	2.390
150000	4980x4960	661.632	3.091.957	155.448	31.21	1.630	3.410
Candidato di 70 cifre (232 bit)							
B	dim matrix	$byte_{lanczos}$	$byte_{gauss}$	$elem$	d	$t_{lanczos}$ [sec.]	t_{gauss} [sec.]
150000	5779x5759	830.892	4.165.214	196.165	33.94	2,23	12,5
200000	6786x6766	1.037.372	5.745.197	245.771	36.22	3,03	14,5
250000	7543x7523	1.228.904	7.099.848	292.140	38.73	3,74	18,6

Tabella 5.14: Confronto tra complessità spaziali e temporali dei metodi di Lanczos e di Gauss.

Appendice A

Complessità del Crivello per numeri B-smooth

La complessità computazionale del crivello di Eratostene adattato per la ricerca di numeri B-smooth è, come già detto nel capitolo relativo al Quadratic Sieve, proporzionale a:

$$\sum_{p \leq B} \frac{1}{p} \sim O(\log \log(B))$$

Determinare il valore di questa sommatoria richiede l'introduzione di alcuni concetti di teoria dei numeri più generali, che permettono di intuire anche risultati raffinati e affascinanti riguardanti i numeri primi, come ad esempio il famoso Teorema dei Numeri Primi ($\pi(x) \sim x/\log(x)$).
Nel seguito saranno introdotte alcune definizioni e teoremi che permettono di risolvere la sommatoria, saranno inoltre abbozzate le dimostrazioni di alcuni teoremi importanti nel campo dei numeri primi. Tutti i concetti sono stati ripresi da [4].

A.1 Funzioni di numeri primi

Per prima cosa si introdurranno alcune funzioni più o meno famose che coinvolgono i numeri primi; l'elenco non ha lo scopo di essere completo, ma solo di introdurre le funzioni che saranno usate in seguito.

$\pi(x)$: funzione che conta quanti numeri primi non superano il valore x. Formalmente si può indicare con $\pi(x) = \sum_{p \leq x} 1$.

$\Lambda(x)$:

$$\Lambda(x) = \begin{cases} 0 & \text{per } x \neq p^m \\ \log(p) & \text{per } x = p^m \end{cases} \tag{A.1}$$

$\vartheta(x)$:

$$\vartheta(x) = \sum_{p \leq x} \log(p) = \log \prod_{p \leq x} p \tag{A.2}$$

$\psi(x)$:

$$\begin{aligned}\psi(x) &= \sum_{p^m \le x} \log(p) \\ &= \sum_{n \le x} \Lambda(n) = \sum_{p \le x} \left\lfloor \frac{\log(x)}{\log(p)} \right\rfloor \log(p)\end{aligned} \tag{A.3}$$

Gli ultimi due passaggi per la funzione $\psi(x)$ derivano da com'è definita $\Lambda(n)$; ogni $p^m \le x$ da un contributo $\log(p)$ a $\psi(x)$, quindi per ogni p questa considera $\lfloor \log_p(x) \rfloor$ volte il contributo di $\log(p)$.
Si noti che $\vartheta(x)$, a differenza di $\psi(x)$ conta un contributo $\log(p)$ solo per i primi p e non anche per le loro potenze.

A.1.1 Relazioni tra $\psi(x)$ e $\vartheta(x)$

Innanzitutto si noti che $p^2 \le x$, $p^3 \le x$, ... sono equivalenti a $p \le x^{1/2}$, $p \le x^{1/3}$, ..., quindi vale:

$$\begin{aligned}\psi(x) &= \sum_{p^m \le x} \log(p) \\ &= \sum_{p \le x} \log(p) + \sum_{p^2 \le x} \log(p) + \sum_{p^3 \le x} \log(p) + \dots \\ &= \sum_{p \le x} \log(p) + \sum_{p \le x^{\frac{1}{2}}} \log(p) + \sum_{p \le x^{\frac{1}{3}}} \log(p) + \dots \\ &= \vartheta(x) + \vartheta(x^{\frac{1}{2}}) + \vartheta(x^{\frac{1}{3}}) + \dots\end{aligned}$$

e quindi:

$$\psi(x) = \sum_m \vartheta(x^{\frac{1}{m}}) \tag{A.4}$$

La serie si interrompe per $x^{1/m} < 2$ cioè per:

$$m > \frac{\log(x)}{\log 2}$$

Dalla definizione di $\vartheta(x)$, poichè ogni termine è al più $\log(x)$ e poichè ci sono al più x termini, si riesce a ricavare un upper bound $\vartheta(x) < x \log(x)$ per $x \ge 2$. Quindi vale anche:

$$\vartheta(x^{\frac{1}{m}}) < x^{\frac{1}{m}} \log(x) \le x^{\frac{1}{2}} \log(x) \quad \text{per } m \ge 2$$

La sommatoria (A.4) ha al più $O(\log(x))$ termini, tutti eccetto il primo maggiorati da $x^{\frac{1}{2}} \log(x)$, quindi:

$$\sum_{m \ge 2} \vartheta(x^{\frac{1}{m}}) = O(x^{\frac{1}{2}} \log^2(x)) \tag{A.5}$$

ovvero, combinando quest'ultima con la (A.4):

$$\psi(x) = \vartheta(x) + O(x^{\frac{1}{2}} \log^2(x)) \tag{A.6}$$

Questa relazione ci permetterà di semplificare la dimostrazione del prossimo teorema.

Teorema A.1 (Complessità di $\psi(x)$ e $\vartheta(x)$)**.** *Le funzioni $\psi(x)$ e $\vartheta(x)$ sono di ordine x:*

$$A_1 x < \vartheta(x) < A_2 x, \quad A_3 x < \psi(x) < A_4 x \qquad (x \geq 2) \tag{A.7}$$

Grazie alla (A.6) sarà sufficiente mostrare che $\vartheta(x) < Ax$ e $\psi(x) > Bx$ per $x \geq 2$; verranno in seguito mostrati dei risultati più precisi che soddisfano le due condizioni.

Teorema A.2. $\vartheta(n) < 2n\log(2) \qquad \forall n \geq 1$

Dimostrazione. Innanzitutto si consideri il coefficiente binomiale:

$$M = \binom{2m+1}{m} = \frac{(2m+1)!}{m!(m+1)!} = \frac{(2m+1)(2m)\dots(m+2)}{m!}$$

Questo è noto essere intero; inoltre, poichè incorre due volte nell'espansione binomiale di $(1+1)^{2m+1}$ si ha che $2M < 2^{2m+1}$ e $M < 2^{2m}$.
Se consideriamo i primi $m+1 < p \leq 2m+1$; si ha che questi dividono tutti il numeratore di M ma non il denominatore:

$$\left(\prod_{m+1<p\leq 2m+1} p \right) \Bigg| M$$

quindi vale:

$$\begin{aligned} \vartheta(2m+1) - \vartheta(m+1) &= \sum_{p\leq 2m+1} \log(p) - \sum_{p\leq m+1} \log(p) \\ &= \sum_{m+1<p\leq 2m+1} \log(p) \leq \log M < 2m\log 2 \end{aligned}$$

dove l'ultimo passaggio è dovuto alla precedente considerazione che $M < 2^{2m}$. Con queste considerazioni è possibile dimostrare per induzione il teorema. I casi base per $n = 1$ e $n = 2$ sono banali ($\vartheta(1) = 0 < 2\log(2)$ e $\vartheta(2) = \log 2 < 4\log(2)$).
Ora sia la proposizione vera $\forall n \leq n_0 - 1$, allora se n_0 è pari si ha $\vartheta(n_0) = \vartheta(n_0 - 1) < 2(n_0 - 1)\log 2 < 2n_0 \log 2$.
Se n_0 è dispari, $n_0 = 2m + 1$, si ha:

$$\begin{aligned} \vartheta(n_0) = \vartheta(2m+1) &= \vartheta(2m+1) - \vartheta(m+1) + \vartheta(m+1) \\ &< 2m\log 2 + 2(m+1)\log 2 \\ &= 2(2m+1)\log 2 = 2n_0 \log 2 \end{aligned}$$

dove si sfrutta l'ipotesi induttiva per $m + 1 < n_0$. □

Per quanto riguarda la relazione $\psi(x) > Bx$, prima è opportuno introdurre il seguente lemma.

Lemma A.1.

$$n! = \prod_p p^{j(n,p)} \qquad \text{con} \quad j(n,p) = \sum_{m\geq 1} \left\lfloor \frac{n}{p^m} \right\rfloor .$$

Senza dare una dimostrazione rigorosa, si può osservare che tra 1 e n ci sono $\lfloor n/p \rfloor$ multipli di p, $\lfloor n/p^2 \rfloor$ multipli di p^2 e così via. Quindi la somma $j(n,p)$ non fa altro che considerare i contributi di ogni potenza di p ed è la funzione *floor* a preoccuparsi di azzerare le potenze che superano n.

Teorema A.3.

$$\psi(x) \geq \frac{1}{4} x \, \log(2)$$

Dimostrazione. Sia:

$$N = \frac{(2n)!}{(n!)^2} = \prod_{p \leq 2} p^{k_p}$$

dove per il lemma precedente:

$$k_p = j(2n,p) - 2j(n,p) = \sum_{m \geq 1} \left(\left\lfloor \frac{2n}{p^m} \right\rfloor - 2 \left\lfloor \frac{n}{p^m} \right\rfloor \right).$$

Ogni termine della sommatoria è uguale a 1 o 0 a seconda che $\lfloor 2n/p^m \rfloor$ sia pari o dispari, in particolare per $p^m > 2n$ è sempre nullo, quindi maggiorando ogni termine con 1 e fermando la somma per $p^m \leq 2n$ si ha:

$$k_p \leq \left\lfloor \frac{\log(2n)}{\log(p)} \right\rfloor$$

e ancora:

$$\log N = \sum_{p \leq 2n} k_p \log(p) \leq \sum_{p \leq 2n} \left\lfloor \frac{\log(2n)}{\log(p)} \right\rfloor \log(p) = \psi(2n)$$

Vale anche:

$$N = \frac{(2n)!}{(n!)^2} = \frac{n+1}{1} \cdot \frac{n+2}{2} \cdots \frac{2n}{n} \geq 2^n$$

e quindi:

$$\psi(2n) \geq n \log 2.$$

Per $x \geq 2$, ponendo $n = \lfloor x/2 \rfloor \geq 1$, si ha:

$$\psi(x) \geq \psi(2n) \geq n \log 2 \geq \frac{x}{4} \log 2$$

che dimostra il teorema. □

Alla luce del Teorema A.1, la (A.6) assume il valore di:

$$\psi(x) \sim \vartheta(x). \tag{A.8}$$

A.1.2 Risultati notevoli

Nella sottosezione precedente si è sostanzialmente dimostrato che $\psi(x) = \Theta(x)$ e $\vartheta(x) = \Theta(x)$, dove $\Theta(x)$ è l'usuale notazione asintotica per il limite superiore ed inferiore di una funzione. Questo fatto permette di dimostrare facilmente dei risultati molto importanti sulla distribuzione dei numeri primi.

Postulato di Bertrand

Sfruttando il fatto che esistono delle costanti fisse A e B tali che $Ax < \vartheta(x) < Bx$ per ogni $x \geq 2$ (vedi Teorema A.1) si può ricavare anche che:

$$\vartheta(Bx/A) > A(Bx/A) = Bx > \vartheta(x).$$

Quindi esiste una costante $C = max(2, B/A)$ per cui $\forall x > 1$ sia ha che esiste un primo p tale che $x < p \leq Cx$ (in quanto tra x e Cx la funzione $\vartheta(x)$ cambia valore). Il postulato di Bertrand raffina questo risultato, in particolare vale:

Teorema A.4 (Postulato di Bertrand)**.** *Se $n \geq 1$, esiste almeno un primo p tale che $n < p \leq 2n$; equivalentemente, se p_r è l' r-esimo primo, allora $p_{r+1} < 2p_r$.*

Comportamento asintotico di $\pi(x)$

Sfruttando nuovamente il Teorema A.1 è possibile vedere come:

$$\vartheta(x) = \sum_{p \leq x} \log(p) \leq \log(x) \sum_{p \leq x} 1 = \log(x)\pi(x)$$

e quindi:

$$\pi(x) \geq \frac{\vartheta(x)}{\log(x)} > \frac{Ax}{\log(x)}$$

Inoltre, se $0 < \delta < 1$,

$$\begin{aligned} \vartheta(x) &= \sum_{p \leq x} \log(p) \geq \sum_{x^{1-\delta} < p \leq x} \log(p) \geq \sum_{x^{1-\delta} < p \leq x} \log(x^{1-\delta}) \\ &= (1-\delta)\log(x) \sum_{x^{1-\delta} < p \leq x} 1 = (1-\delta)\log(x) \left(\pi(x) - \pi(x^{1-\delta})\right) \\ &\geq (1-\delta)\log(x) \left(\pi(x) - x^{1-\delta}\right). \end{aligned}$$

Da questa si può ricavare

$$\pi(x) \leq x^{(1-\delta)} + \frac{\vartheta(x)}{(1-\delta)\log(x)} < \frac{Bx}{\log(x)},$$

dove nell'ultimo passaggio $x^{1-\delta}$ può essere semplificato in quanto all'infinito è $o(x)$. Unendo le due disuguaglianze trovate si ha $Ax/\log(x) < \pi(x) < Bx/\log(x)$, dove A e B sono le stesse costanti valide per $\vartheta(x)$ nel Teorema A.1. Questo dimostra che $\pi(x) = \Theta(x/\log(x))$.

A.2 Due trasformazioni formali

Questa sezione tratta due trasformazioni semplici da capire ma che permetteranno di ottenere risultati molto significativi.

Teorema A.5. *Sia c_1, c_2, ... una sequenza di numeri, $C(t) = \sum_{n \le t} c_n$ e $f(t)$ una funzione di t qualsiasi. Allora*

$$\sum_{n \le x} c_n f(n) = \sum_{n \le x-1} C(n)\left(f(n) - f(n+1)\right) + C(x) f(\lfloor x \rfloor). \tag{A.9}$$

Inoltre, se $c_j = 0$ per $j < n_1$ e $f(t)$ ha derivata continua per $t \ge n_1$, allora:

$$\sum_{n \le x} c_n f(n) = C(x) f(x) - \int_{n_1}^{x} C(t) f'(t) dt. \tag{A.10}$$

Dimostrazione. Per quanto riguarda la (A.9), ponendo $N = \lfloor x \rfloor$, vale:

$$\begin{aligned}
\sum_{n \le x} c_n f(n) &= c_1 f(1) + c_2 f(2) + \cdots + c_N f(N) = \\
&= C(1) f(1) + \left(C(2) - C(1)\right) f(2) + \cdots + \\
&\quad + \left(C(N) - C(N-1)\right) f(N) \\
&= C(1)\left(f(1) - f(2)\right) + \cdots + C(N-1)\left(f(N-1) - f(N)\right) + \\
&\quad + C(N) f(N).
\end{aligned}$$

Dato che $C(x) = C(\lfloor x \rfloor)$, questa prova la (A.9). Per dedurre la (A.10) si osservi che $C(t) = C(n)$ per $n \le t < n+1$ e quindi:

$$C(n)\left(f(n) - f(n+1)\right) = -C(n)\left(f(n+1) - f(n)\right) = -\int_{n}^{n+1} C(t) f'(t) dt.$$

Quindi usando quest'ultima relazione, la (A.9) e il fatto che $C(t) = 0$ per $t < n_1$, si ottiene:

$$\begin{aligned}
\sum_{n \le x} c_n f(n) &= \sum_{n < x-1} \left(-\int_{n}^{n+1} C(t) f'(t) dt \right) + C(N) f(N) \\
&= C(N) f(N) - \int_{n_1}^{N} C(t) f'(t) dt
\end{aligned}$$

ed estendendo l'estremo superiore di integrazione da N a x ($-\int_N^x C(t) f'(t) dt = -C(x) f(x) + C(N) f(N)$) si ottiene infine

$$\sum_{n \le x} c_n f(n) = C(x) f(x) - \int_{n_1}^{x} C(t) f'(t) dt.$$

□

A.2.1 Sommatorie notevoli e raffinamento della stima di $\pi(x)$

Le trasformazioni introdotte nella sottosezione precedente sono molto potenti e permettono di trovare stime precise per molte sommatorie, che permetteranno poi di mostrare la convergenza di $\sum_{p \le x} 1/p$. Il primo passo è dimostrare la seguente.

Teorema A.6.

$$\sum_{n\leq x}\frac{\Lambda(n)}{n}=\log(x)+O(1).$$

Dimostrazione. Innanzitutto usiamo la (A.10) con $c_n = 1$ ($C(n) = n$) e $f(t) = \log(t)$ per calcolare:

$$\sum_{n\leq x}\log(n)=x\log(x)-\int_1^x t(\log(t))'dt=x\log(x)-(x-1)=x\log(x)+O(x).$$

Inoltre usando il lemma (A.1):

$$\begin{aligned}\sum_{n\leq x}\log(n) &= \log(n!)=\sum_{p\leq x}j(\lfloor x\rfloor,p)\log(p)\\ &= \sum_{p^m\leq x}\left\lfloor\frac{x}{p^m}\right\rfloor\log(p)=\sum_{p^m\leq x}\left\lfloor\frac{x}{n}\right\rfloor\Lambda(n)\end{aligned}$$

Rimuovendo il $floor$ si commette un errore di al più:

$$\sum_{n\leq x}\Lambda(n)=\psi(x)=O(x),$$

quindi:

$$\sum_{n\leq x}\frac{x}{n}\Lambda(n)=\sum_{n\leq x}\log(n)+O(x)=x\log(x)+O(x).$$

Dividendo per x si ottiene finalmente:

$$\sum_{n\leq x}\frac{\Lambda(n)}{n}=\log(x)+O(1).$$

□

Teorema A.7.

$$\sum_{p\leq x}\frac{\log(p)}{p}=\log(x)+O(1).$$

Dimostrazione. Se consideriamo:

$$\begin{aligned}\sum_{n\leq x}\frac{\Lambda(n)}{n} - \sum_{p\leq x}\frac{\log(p)}{p} &= \sum_{m\geq 2}\sum_{p^m\leq x}\frac{\log(p)}{p^m}\\ &< \sum_p\left(\frac{1}{p^2}+\frac{1}{p^3}+\cdots\right)\log(p)=\sum_p\frac{\log(p)}{p(p-1)}\\ &< \sum_{n\geq 2}\frac{\log(n)}{n(n-1)}=A.\end{aligned}$$

Quindi poichè la sommatoria considerata e la sommatoria del teorema precedente differiscono per una costante, è valida la proposizione del teorema. □

Prima di utilizzare questi risultati per determinare la complessità del crivello per numeri B-smooth, vale la pena sfruttarli per raffinare la stima di $\pi(x)$ precedentemente ottenuta.
Usando la (A.10) con $f(t) = 1/t$ e $c_n = \Lambda(n)$ ($C(x) = \psi(x)$), si ha:

$$\sum_{n \leq x} \frac{\Lambda(n)}{n} = \frac{\psi(n)}{n} + \int_2^x \frac{\psi(t)}{t^2} dt$$

e quindi combinando i Teoremi A.6 e A.1 si ottiene:

$$\int_2^x \frac{\psi(t)}{t^2} dt = \log(x) + O(1). \tag{A.11}$$

Da questa è possibile dedurre:

$$\underline{\lim}\{\psi(x)/x\} \leq 1, \qquad \overline{\lim}\{\psi(x)/x\} \geq 1.$$

Infatti, se $\underline{\lim}\{\psi(x)/x\} = 1+\delta$ con $\delta > 0$, si ha che $\psi(x) > (1+\frac{1}{2}\delta)x$ per ogni x maggiore di qualche x_0. Quindi:

$$\int_2^x \frac{\psi(t)}{t^2} dt > \int_2^{x_0} \frac{\psi(t)}{t^2} dt + \int_{x_0}^x \frac{(1+\frac{1}{2}\delta)}{t} dt > \left(1 + \frac{1}{2}\delta\right) \log(x) - A$$

in contraddizione con la (A.11). Similmente, se $\overline{\lim}\{\psi(x)/x\} = 1-\delta$ con $\delta > 0$ si ottiene:

$$\int_2^x \frac{\psi(t)}{t^2} dt < \int_2^{x_0} \frac{\psi(t)}{t^2} dt + \int_{x_0}^x \frac{1-\delta}{t} dt < (1-\delta) \log(x) - B.$$

Ora, sfruttanto questi limiti asintotici per $\psi(x)$ (che raffinano la precedente stima $\psi(x) = \Theta(x)$ in $\psi(x) \sim x$), ricordando che $\psi(x) \sim \vartheta(x)$ e le considerazioni sul comportamento asintotico della $\pi(x)$ nella sottosezione A.1.2 (cioè che le costanti A e B per cui $Ax < \vartheta(x) < Bx$ valgono anche per $Ax/\log(x) < \pi(x) < Bx/\log(x)$) si può dedurre che se $\pi(x)/(x/\log(x))$ ammette un limite all'infinito, questo è 1 ($\pi(x) \sim x/\log(x)$). Purtroppo l'esistenza di questo limite è particolarmente difficile da dimostrare; nonostante questo si citerà ora senza dare ulteriori dimostrazioni il seguente importante teorema:

Teorema A.8. (The Prime Number Theorem). *Il numero di primi non superiori a x è asintotico a $x/\log x$:*

$$\pi(x) \sim \frac{x}{log\ x}.$$

A.3 Soluzione di $\sum_{p \leq x} 1/p$

Dopo questo piccolo preambolo, è finalmente possibile affrontare la stima della sommatoria di nostro interesse. Si consideri la (A.10) con $c_n = \log(n)/n$ se n è primo e $c_n = 0$ per n non primo, in modo che

$$C(x) = \sum_{p \leq x} \frac{\log(p)}{p} = \log(x) + \tau(x),$$

dove $\tau(x) = O(1)$ per il Teorema A.7. Sia inoltre $f(t) = 1/\log(t)$. Con questi valori la (A.10) diventa:

$$\begin{aligned}\sum_{p\leq x}\frac{1}{p} &= \frac{C(x)}{\log(x)} + \int_2^x \frac{C(t)}{t\log^2(t)}dt \\ &= 1 + \frac{\tau(x)}{\log(x)} + \int_2^x \frac{dt}{t\log(t)} + \int_2^x \frac{\tau(x)}{t\log^2(t)}dt. \qquad \text{(A.12)}\end{aligned}$$

Ricordando che:

$$\begin{aligned}(\log\,\log(x))' &= \frac{1}{x\log(x)} \\ \left(\frac{1}{\log(x)}\right)' &= -\frac{1}{x\log^2(x)}\end{aligned}$$

la (A.12) diventa:

$$\begin{aligned}\sum_{p\leq x}\frac{1}{p} &= 1 + \frac{\tau(x)}{\log(x)} + \log\,\log(x) - \log\,\log(2) \\ &\quad + \int_2^\infty \frac{\tau(x)dt}{t\log^2(t)} - \int_x^\infty \frac{\tau(x)dt}{t\log^2(t)}\end{aligned}$$

Raccogliendo ora i termini

$$\begin{aligned}B_1 &= 1 - \log\,\log 2 + \int_2^\infty \frac{\tau(x)dt}{t\log^2(t)} = O(1) \\ E(x) &= \frac{\tau(x)}{\log(x)} - \int_x^\infty \frac{\tau(x)dt}{t\log^2(t)} = O\left(\frac{1}{\log(x)}\right)\end{aligned}$$

si ha

$$\sum_{p\leq x}\frac{1}{p} = \log\,\log(x) + B_1 + o(1)$$

che permette di concludere quanto desiderato: $\sum_{p\leq x}\frac{1}{p} \sim \log\,\log(x)$.

Bibliografia

[1] C. Pomerance. *Analysis and comparison of some integer factoring algorithms*. In H. Lenstra, Jr. and R. Tijdeman, editors, *Computational methods in number theory, Part I*, volume 154 of *Math. Centre Tracts*, pages 89-139, Math. Centrum, 1982

[2] R. Crandall, C. Pomerance. *Prime Numbers, a Computational Perspective*, II edition, Springer, 2005

[3] N. Lauritzen. *Concrete Abstract Algebra, From Number to Gröbner Bases*, Cambridge University Press, 2003

[4] G. H. Hardy, E. M. Wright: *An introduction to the theory of number*, Fifth edition, Clarendon Press, Oxford, 1979

[5] C. Pomerance. *Smooth Numbers and the Quadratic Sieve*, Algorithmic Number Theory, MSRI Publications, Voluma 44, 2008.

[6] C. Pomerance. *A Tale of Two Sieves*, Notices of the Amer. Math. Soc., December 1996.

[7] C. Pomerance. *The Quadratic Sieve Factoring Algorithm*, T. Beth, N. Cot e I. Ingemarsson (Eds.): Advances in Cryptoloy - Eurocrypt '84, LNCS 209, pp. 169-182, 1985

[8] A. Granville. *Smooth numbers: computational number theory and beyond*, Algorithmic Number Theory, MSRI Publications, Volume 44, 2008

[9] E. R. Candfield, P Erdös, C. Pomerance, *On a problem of Oppenheim concerning factorisatio numerorum*, J. Number Theory 17 (1983), 1-28

[10] C. Pomerance, J. W. Smith, *Reduction of Huge, Sparse Matrices over Finite Fields Via Created Catastrophes*, Experimental Math. 1 (1992), pp. 90-94

[11] C. Pomerance, J. W. Smith, R. Tuler, *A Pipeline Architecture For Factoring Large Integers with the Quadratic Sieve Algorithm*, SIAM J. Comput. 17 (1988), pp. 387-403.

[12] S. Contini. *Factoring Integers with the Self-Initializing Quadratic Sieve*, Master Thesis, U. Georgia, 1997

[13] P. L. Montgomery, *A Block Lanczos Algorithm for Finding Dependencies over GF(2)*, Advances in cryptology, Eurocrypt '95, Lecture Notes in Comput. Sci. 921 (1995), pp. 106-120.

[14] J. K. Cullum, R. A. Willoughby, *Lanczos algorithms for large symmetric eigenvalue computations. Vol. I Theory*, Birkhäuser, Boston, 1985.

[15] https://documents.epfl.ch/users/l/le/lenstra/public/papers/rsa768.txt

Elenco delle tabelle

Elenco delle figure