Project 1——大整数分解

$$N = \prod p_i^{a_i}$$

Overview

- Application -- RSA
- Algorithms

Application -- RSA

由Rivest, Shamir, Adleman于1977年提出,

目前应用最广泛的公钥密码算法

- 随机产生大素数p, q　　n = pq , Φ(n) = (p-1)(q-1)
- 随机选取整数e, 满足1<e<Φ(n) 且 gcd(e, Φ(n)) = 1
- 存在整数d, u 满足de + uΦ(n) = 1
- 仅保留n, e, d 清除其余数字

Application -- RSA

Alice

n, e, d

解密：

$m = m^{ed+u\Phi(n)}$

$\equiv m^{ed} \equiv c^{d} \bmod n$

若 $\gcd(m,n) = 1$

则 $m^{\Phi(n)} \equiv 1 \bmod n$

n, e

c

Bob

m

加密：

$c = m^{e} \bmod n$

Algorithms

- 试除法　naïve!
- 蒙特卡罗方法 Pollard: rho
- 椭圆曲线算法
- 平方同余算法：二次筛法、数域筛法
- 量子算法：Shor算法

Algorithms

Fermat: $N = x^2 - y^2$

Kraitchik: $x^2 \equiv y^2 \pmod{N}$

Lehmer & Powers & Dixon: Continued Fraction

Pomerance: Quadratic Sieve

MPQS, PPMPQS, SIQS, PPSIQS

Related Work -- YAFU

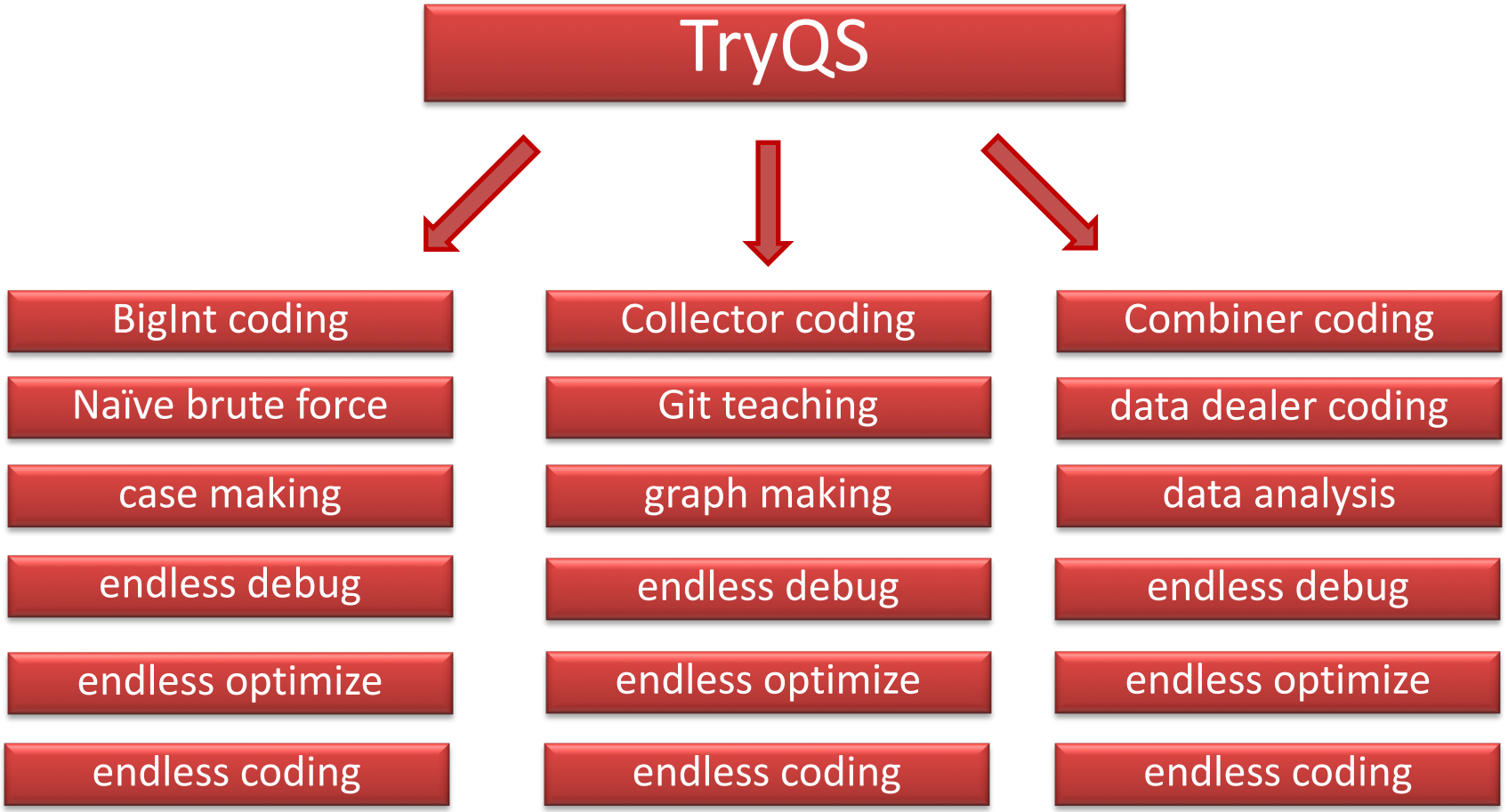
utility	factoring	arithmetic
rsa	factor	shift
size	siqs	nroot
primes	smallmpqs	modexp
nextprime	nfs	sqrt
rand	squfof	lg2
randb	pm1	log
isprime	pp1	ln
ispow	rho	gcd
issquare	trial	jacobi
sieverange	ecm	modinv
testrange	fermat	fib
bpsw	snfs	luc
aprcl		llt

```
**************************
Starting factorization of 114783454800748948121967173609627354252449742494021575262239 7
using pretesting plan: normal
no tune info: using qs/gnfs crossover of 95 digits
**************************
rho: x^2 + 3, starting 1000 iterations on C61
rho: x^2 + 2, starting 1000 iterations on C61
rho: x^2 + 1, starting 1000 iterations on C61
pm1: starting B1 = 150K, B2 = gmp-ecm default on C61
current ECM pretesting depth: 0.00
scheduled 30 curves at B1=2000 toward target pretesting depth of 18.77
Finished 30 curves using Lenstra ECM method on C61 input, B1=2K, B2=gmp-ecm default
current ECM pretesting depth: 15.18
scheduled 55 curves at B1=11000 toward target pretesting depth of 18.77
Finished 55 curves using Lenstra ECM method on C61 input, B1=11K, B2=gmp-ecm default
final ECM pretested depth: 18.89
scheduler: switching to sieve method
starting SIQS on c61: 114783454800748948121967173609627354252449742494021575262239 7
random seeds: 1412284569, 2947314842
==== sieve params ====
n = 62 digits, 204 bits
factor base: 3824 primes (max prime = 78781)
single large prime cutoff: 4726860 (60 * pmax)
allocating 2 large prime slices of factor base
buckets hold 2048 elements
using 32k sieve core
sieve interval: 4 blocks of size 32768
polynomial A has ~ 7 factors
using multiplier of 13
using SPV correction of 20 bits, starting at offset 29
using SSE2 for trial division and x64 sieve scanning
using SSE4.1 enabled 32k sieve core
using SSE2 for resieving 13-16 bit primes
trial factoring cutoff at 69 bits
```

e.g. factor(rsa(200))

6.6453 seconds

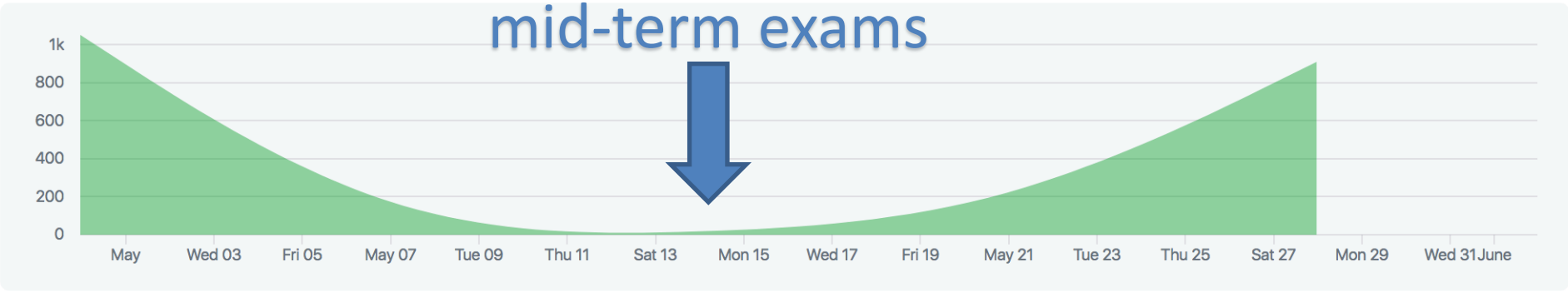
Division of our project



Apr 30, 2017 – Jun 2, 2017

Contributions to master, excluding merge commits

Contributions: **Additions**

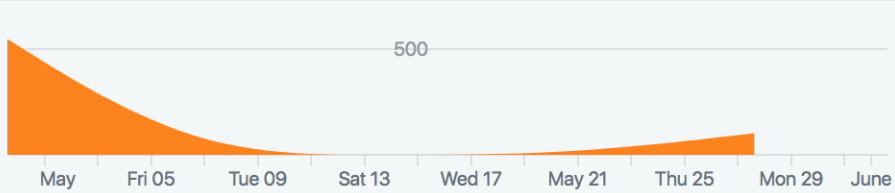


ShenQianli #1

33 commits / 798 ++ / 551 --

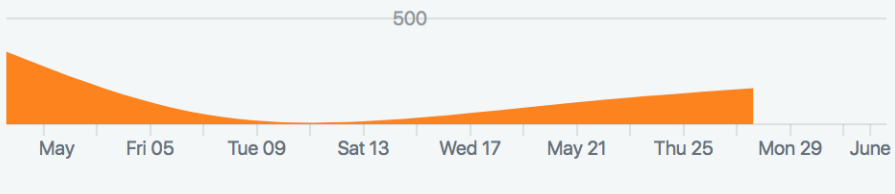
Ashlippers #2

6 commits / 650 ++ / 389 --



GuYoupeng #3

8 commits / 620 ++ / 179 --



large_integer.h

large_integer.h 〉 No Selection

```
#ifndef LARGE_INTEGER_H
#define LARGE_INTEGER_H

#include <iostream>
#include <math.h>
#define DIGIT_MAX 1000
//const int MAX = 3000;

using namespace std;

class BigInt {
    bool valid, negative;
    unsigned long long num[DIGIT_MAX];

    int compare(const BigInt&) const;
    int kcompare(const BigInt&, int) const;
public:
    BigInt();
    BigInt(const char*);
    BigInt(int);
    BigInt(long);

    bool operator == (const BigInt &) const;
    bool operator != (const BigInt &) const;
    bool operator < (const BigInt &) const;

    BigInt operator - () const;
    BigInt operator +(const BigInt&);
    BigInt operator -(const BigInt&);
    BigInt operator *(const BigInt&);
    BigInt operator /(const BigInt&);
    BigInt operator %(const BigInt&);
    BigInt bigsqrt();

    friend ostream &operator << (ostream&, const BigInt&);
};

BigInt biggcd(BigInt, BigInt);
#endif
```

collector(伪代码)

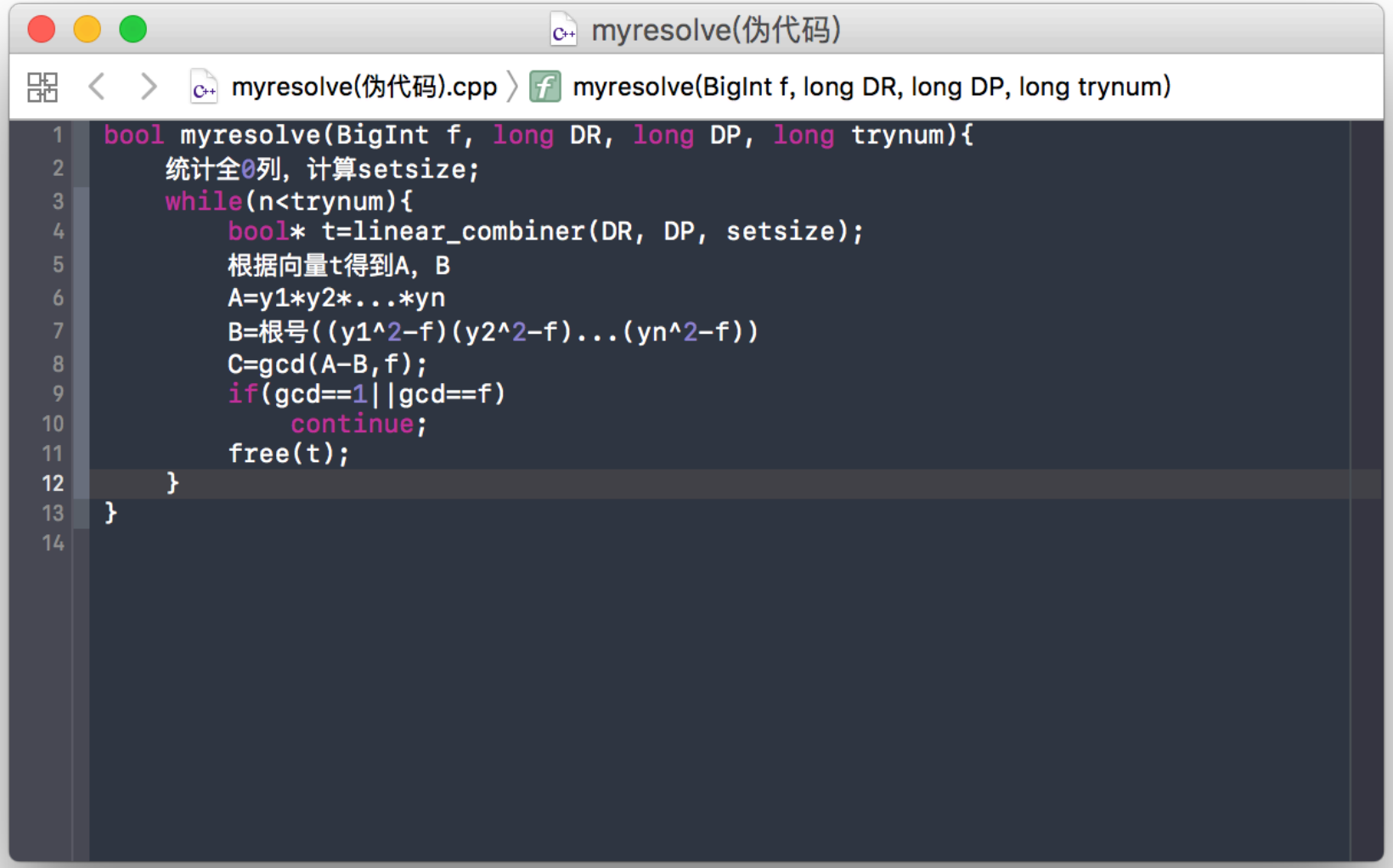
collector(伪代码).cpp 〉 No Selection

```
1  void gen(BigInt A, int &ccnt, int num_prime, int &pcnt) {
2      M = A * A - f;
3      根据规定规模的质数表分解M，在bool temp[]中记录
4      if (remain == 1)
5          将temp塞入m[][]
6      else //partial
7          if(pr[]中有与当前向量相同remain的向量)
8              将这两个向量组合塞入m[][]
9          else
10             将该remain存入pr[]，并存储temp[]
11 }
12 void collect(BigInt f, int num_relation, int num_prime, float alpha) {
13     while (ccnt <= alpha*cursetsize && ccnt <= num_relation) {
14         gen(A, ccnt, num_prime, pcnt);
15         A=A+1;
16     }
17 }
18
```

linear_combiner(伪代码) — Edited

linear_combiner(伪代码).cpp 〉 linear_combiner(long DR, long DP, long setsize)

```
/*
 randomly choose setsize+1 vectors and use some of them to get a linear combiniation
 the func will return an address of a bool-type array
 for we will use the function calloc, don't foget to free the array then
 */
bool* linear_combiner(long DR, long DP, long setsize)
{
    //printf("linear combining...\n");
    bool *t=(bool *)calloc(DR, sizeof(bool));
    bool a[][]={0};
    bool b[][]={0};
    set <int> s;
    while(s.size()<setsize){
        s.insert(随机数);
    }
    for(s中的每个数){
        将m[][]的该行塞入a;
    }
    b=I(setsize);
    对a进行高斯消元，b作相应的行变换;
    在a中得到全0行，根据b中相应行得到向量t;
    return t;
}
```

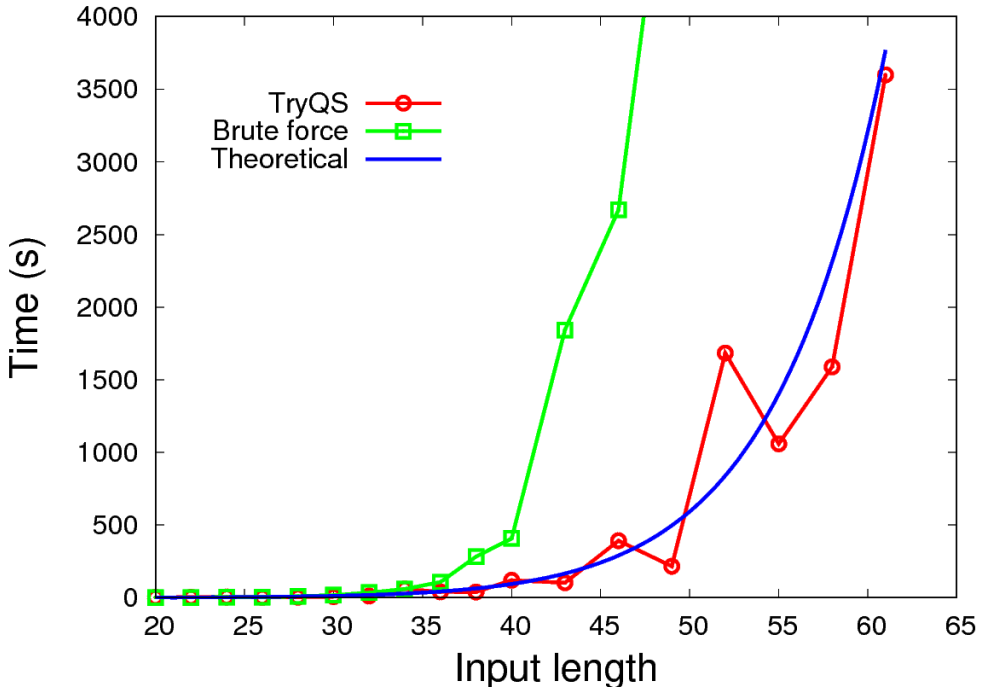

myresolve(伪代码)
myresolve(伪代码).cpp
myresolve(BigInt f, long DR, long DP, long trynum)
bool myresolve(BigInt f, long DR, long DP, long trynum){
统计全0列，计算setsize;
while(n<trynum){
bool* t=linear_combiner(DR, DP, setsize);
根据向量t得到A, B
A=y1*y2*...*yn
B=根号((y1^2-f)(y2^2-f)...(yn^2-f))
C=gcd(A-B,f);
if(gcd==1||gcd==f)
continue;
free(t);
}
}

Part IV - Evaluation

- 测试内容：
  - 分解不同长度的大整数需要的时间
  - 与其他算法的比较
  - 与理论渐进复杂度的比较
  - 调整QS算法中的参数对于运行时间的影响
- 测试环境：
  - Intel(R) Core(TM) i7-7700K CPU @ 4.80GHz
  - Dual Channel DDR4-3200 SDRAM
  - Ubuntu 14.04.5 LTS on Windows 10

Part IV - Evaluation

- Time consumption – Length of input large integer.



- Theoretical asymptotic complexity: $e^{(1+o(1))\sqrt{\ln n \ln \ln n}}$

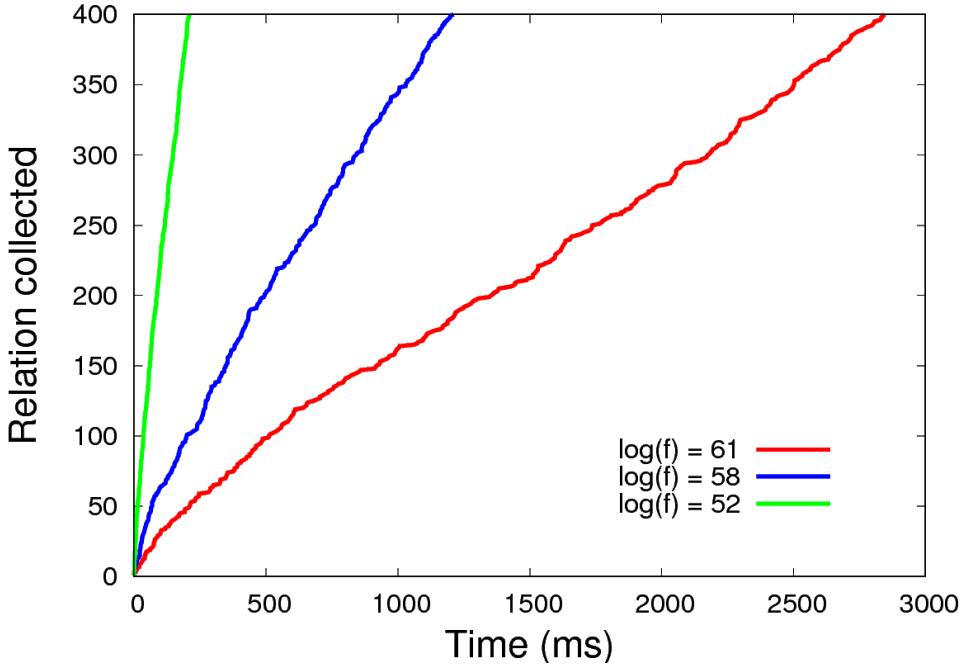
Part IV - Evaluation

- Time consumption breakdown
  - Collect – 收集能被因子基完全分解的向量，直到向量数量大于因子基维数
  - Combine – 用高斯消元求上述向量的线性组合
- Collect阶段占用绝大部分时间

f = 28847761134023128 1	RSA-129
collect time = 1570437 ms	collect time = 5000 year
combine time = 18296 ms	combine time = 21 hour

Part IV - Evaluation

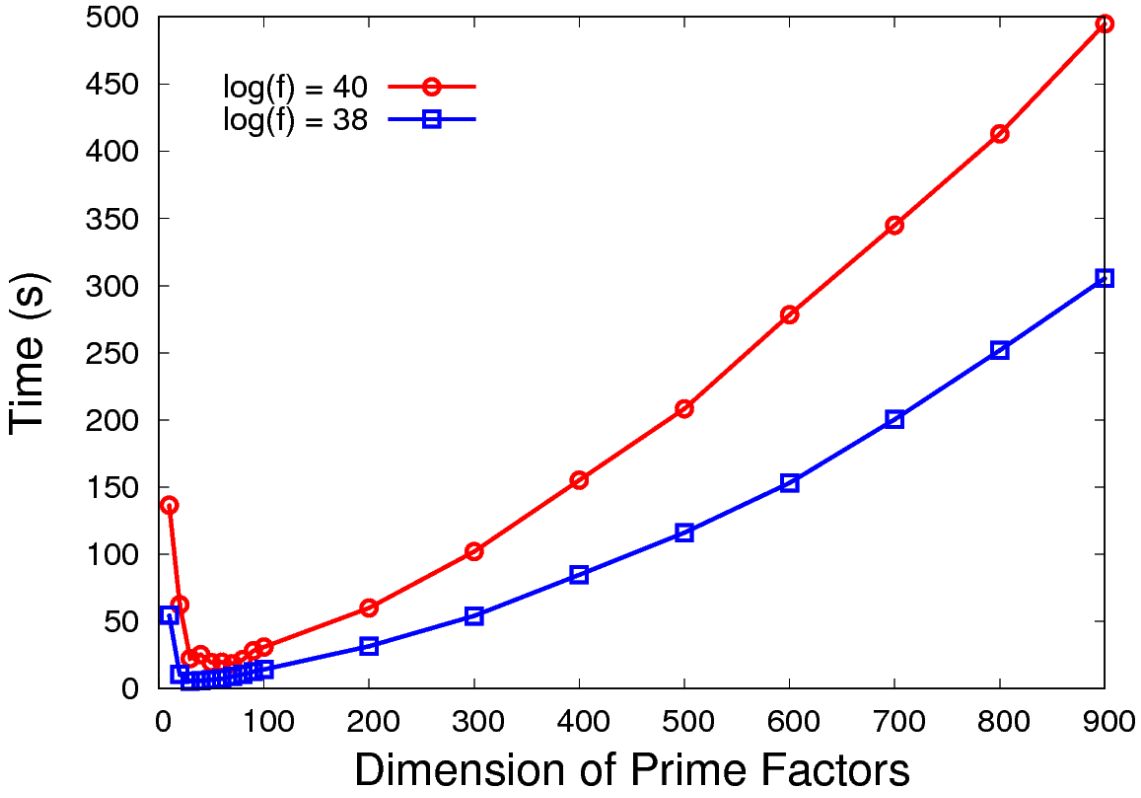
Time consumption – Collect



- 52 bit BigInt – 7112 rels/sec
- 58 bit BigInt – 3027 rels/sec
- 61 bit BigInt – 522 rels/sec

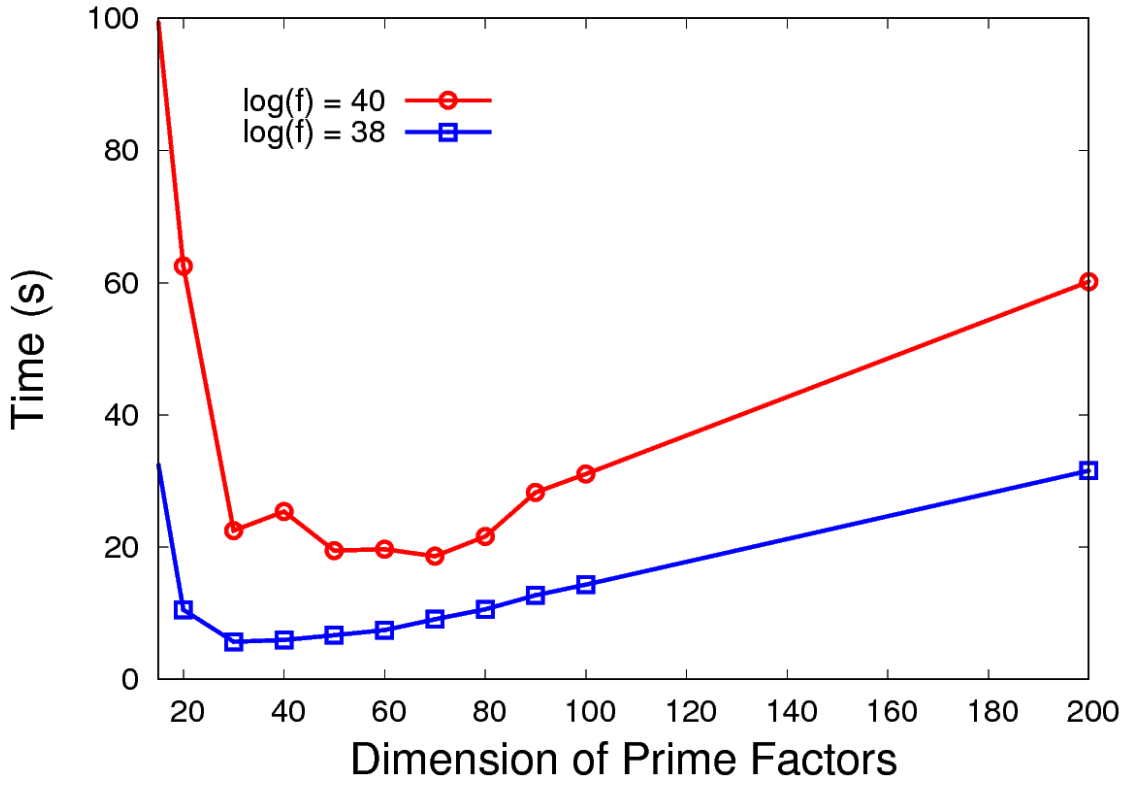
Part IV - Evaluation

- 如何选择因子基的维数？



Part IV - Evaluation

- 如何选择因子基的维数？



Part IV - Evaluation

- 总结：
  - 二次筛法能够对包含大质数的整数进行分解，在特定范围内时间代价远少于朴素暴力方法。
  - Collect过程消耗时间与待分解数大小与所需向量数量直接相关
  - 对特定大小的待分解数，设置恰当的因子基维数能显著加快筛法求解速度

Part V - Discussion

- Future improvement
  - Collect过程采用分布式计算
  - 例如：MapReduce框架下的Collect

    mapper 接收一个二次筛中的数，根据因子基分解，

    Emit(remain, relation)

    reducer 接收所有remain相同的键值对，并产生若干完全分解的向量

    输出结果为收集到的完全分解向量

Part V - Discussion

- Future improvement
  - 单一计算机条件下，Collect可以进行多线程并发
  - 可以使用更高性能的高精度运算库

    例如：The GNU Multiple Precision Arithmetic Library
  - 实现多种整数分解算法，对不同的输入自适应调整应用的算法

Part VI – Open Source

- Available at https://github.com/Ashlippers/TryQS
- Any stars and pull requests are welcome!

THANKS