Application of bio-inspired algorithm to the problem of integer factorisation

Roman V. Yampolskiy

Speed School of Engineering,
Computer Engineering and Computer Science,
University of Louisville,
Louisville, KY 40292, USA
E-mail: roman.yampolskiy@louisville.edu

Abstract: This paper describes an attempt at developing an evolutionary algorithm capable of solving non-trivial cases of integer factorisation, which are at the heart of security behind the modern public key cryptography systems. After reviewing previous work in integer factorisation and describing the developed genetic algorithm the paper addresses issues of convergence to a local maxima associated with the performance of genetic algorithms. Specifically, properties and statistical distribution of local maxima points associated with the integer factorisation problems are reviewed. Finally, performance of the developed system is analysed and recommendations are made for future research paths.

Keywords: evolutionary algorithm; integer factorisation; global maxima; genetic algorithm.

Reference to this paper should be made as follows: Yampolskiy, R.V. (2010) 'Application of bio-inspired algorithm to the problem of integer factorisation', *Int. J. Bio-Inspired Computation*, Vol. 2, No. 2, pp.115–123.

Biographical notes: Roman V. Yampolskiy received his PhD from the Department of Computer Science and Engineering at the University at Buffalo. There he was a recipient of a four year National Science Foundation IGERT fellowship. After graduating, he served as an Affiliate Academic at the University of London, College of London until accepting an Assistant Professor position at the University of Louisville in 2008. He had previously worked at the Laboratory for Applied Computing at the Rochester Institute of Technology and at the Center for Unified Biometrics and Sensors at the University at Buffalo. His main areas of interest are computer security, artificial intelligence, behavioural biometrics and intrusion detection. He is the author of over 40 publications including multiple books.

1 Introduction

Integer factorisation (IF) is a fundamental theoretical problem in mathematics and computer science. It is also a problem of great practical importance as security of public key cryptography (PKC), which is used in digital communications and e-commerce, is based on our inability to quickly factor large integers. As early as 1801 Gauss has stated (Knuth, 1981):

> "The problem of... resolving composite numbers into their prime factors, is one of the most important and useful in all of arithmetic... The dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated."

Factoring a positive integer N means finding positive integers p and q such that $N = p * q$, where both p and q are greater than 1. Such numbers p and q are called factors of N. Positive integers that can be factored are called composite numbers, others are known as prime numbers (Lenstra and Lenstra 1993). The most difficult cases of IF problem involve situations in which both p and q are prime and of similar size in terms of number of digits comprising them. The decision version of IF can be stated as: given an integer R and an integer S with $1 \leq S \leq R$, does R have a factor f with $1 < f < S$? This representation is useful because most well-studied complexity classes are defined as classes of decision problems. In combination with a binary search algorithm, a solution-function to a decision version of IF can solve the general case of IF in logarithmic number of queries (Adi Shamir, 2003). The difficulty of the decision version of the IF problem in terms of its membership in a specific complexity class remains an open question. IF is known to belong to both non-deterministic polynomial time (NP) and co-NP classes, because both 'yes' and 'no' decisions can be verified given the prime factors either via polynomial time primality test such as AKS (Brent, 1999) or via simple multiplication of divisors.

Table 1 Existing IF approaches

Type	*Algorithm/approach*	*Researcher(s) or publication with additional information*	*Year*	*Complexity*[1]	*Probabilistic versus deterministic*	*Record number factorised*[2]
Special	Trial division	(Bressoud, 1989)	-	$ne^{n/2}$	D	15D
	Pollard's rho	(Pollard, 1975)	1975	$n^{1/4}$ *polylog* (n)	P	F_8
	Elliptic curve	(Lenstra, 1987)	1987	$e^{(1+o(1))\sqrt{(\ln p \ln\ln p)}}$	D	67D
	Fermat's	(Mckee, 1999)	1630s	e^{n}	P	5D
	Euler's	(Mckee, 1996)	1770s	$N^{1/3+\varepsilon}$	P	7D
	Special number field sieve	(Lenstra and Lenstra, 1993)	1988	$e^{(1+o(1))\left(\frac{32}{9}\log n\right)^{1/3}(\log\log n)^{2/3}}$	D	313D
	Pollard's $p-1$	(Pollard, 1974)	1974	$B \times \log B \times \log^2 N$	P	
	Williams' $p+1$	(Williams, 1982)	1982		P	
	Dixon's	(Dixon, 1981)	1981	$\exp\left(2\sqrt{2}\sqrt{(\log n \log\log n)}\right)$	P	
General	Continued fraction	(Lehmer and Powers, 1931)	1931	$e\sqrt{(2\log n \log\log n)}$	P	
	Quadratic sieve	(Pomerance, 1996)	1981	$e\sqrt{(\log n \log\log n)}$	P	129D
	General number field sieve	(Elkenbracht-Huizing, 1997)	1990	$e^{(c+o(1))(\log n)^{1/3}(\log\log n)^{2/3}}$	D	200D
	Shanks' square forms	(Bressoud, 1989)	1970s	$\sqrt[4]{N}$	D	
	Shor's	(Shor, 1997)	1994	$e^{(\log N)^{1/3}(\log\log N)^{2/3}}$	D	2D
Alternative	DNA	(Chang et al., 2005)	2005	$(.5n)^3$	D	
	Neural network	(Meletiou et al., 2002)	2002	-	P	4D
	Genetic programming	(Chan, 2002)	2002	-	P	4D
	TWINKLE	(Shamir, 1999)	1999	Constant improvement of GNFS	D	512b[3]
	TWIRL	(Adi Shamir, 2003)	2003	Constant improvement of GNFS	D	1024b[4]
	Oracle	(Maurer and Rueppel, 1992)	1992	εn	P	
	Partial information	(Rivest and Shamir, 1986)	1986	$n/3$	P	

Notes: [1]N is the number we wish to factor, n is the number of digits in N, p is the smallest factor of N, e is 2.71828182845904523536…, ε is an arbitrarelly small positive real number, B is a smoothness bound, c – constant which depends on the complexity measure and on the variant of the algorithm.
[2]D – decimal digits, F – Fermat's number.
[3]Theoretical size, The Weizmann Institute Key Locating Engine (TWINKLE) hardware has never been constructed.
[4]Theoretical size, The Weizmann Institute Relation Locator (TWIRL) hardware has never been constructed.

Research in IF spans hundreds of years and demonstrates a great variety of approaches tried by mathematicians in their attempt to find a satisfactory solution to this fundamental problem. Table 1 summarises the most important achievements in the rich history of IF research. Overall attempted approaches can be classified into three somewhat overlapping categories. *Special* approaches are aimed at finding factorisations of number in a specific form as it is sometimes possible to take advantage of their unique structure in order to do much better compare to general-form numbers. *General* form algorithms are designed to handle any type of number and as a result of their generality such approaches are less powerful in comparison to algorithms designed to take advantage of special forms. Finally, a greatly diverse group of *alternative* approaches is comprised of such diverse and sometimes questionable methods as quantum and DNA computing, neural networks, genetic programming and even oracles.

With such a rich history it is not at all surprising that great breakthroughs have been made in the quest for IF. Table 2 presents a condensed list of important factorisations achieved in the last 20 years. It reports on record breaking attempts, algorithms used to produce them and gives credit to the scientists responsible for these great achievements.

Table 2 IF records by year and type

Number	*Decimal digits*	*Binary digits*	*Factoring algorithm*	*Type of number*	*Factored by researcher(s)*	*Date factored*
$10^{381}+1$	67	221	ECM	General	B. Dodson	Aug. 24, 2006
RSA100	100	330	MPQS	General	A.K. Lenstra	April 1, 1991
RSA110	110	364	MPQS	General	A.K. Lenstra, et al.	April 14, 1992
c116	116	383	MPQS	General	A.K. Lenstra et al.	1990
$c2^{481}+2^{241}+1$	118	390	QS	General	S. Contini et al.	June 15, 1996
P13171	119	395	GNFS	General	S. Contini et al.	Nov. 26, 1994
RSA120	120	397	MPQS	General	T. Denny et al.	June 9, 1993
$c7^{352}+1$	128	423	GNFS+FPGA	General	T. Shimoyama et al.	Sep. 4, 2006
RSA129	129	426	MPQS	General	A.K. Lenstra et al.	April 26, 1994
RSA130	130	430	GNFS	General	A.K. Lenstra et al.	April 10, 1996
RSA140	140	463	GNFS	General	H.J.J. te Riele et al.	Feb. 2, 1999
RSA150	150	496	GNFS	General	K. Aoki et al.	April 16, 2004
RSA155	155	512	GNFS	General	H.J.J. te Riele et al.	Aug. 22, 1999
$c2^{953}+1$	158	523	GNFS	General	F. Bahr et al.	January 2002
RSA160	160	530	GNFS	General	J. Franke et al.	April 1, 2003
$12^{151}-1$	163	542	SNFS	Special	H. Boender et al.	July 1993
$c2^{1826}+1$	164	543	GNFS	General	K. Aoki et al.	Dec. 19, 2003
RSA576	174	576	GNFS	General	J. Franke et al.	Dec. 3, 2003
$c11^{281}+1$	176	583	GNFS	General	K. Aoki et al.	May 2005
$32633^{41}-1$	186	616	SNFS	Special	S. Cavallar et al.	Sep. 15, 1998
RSA640	193	640	GNFS	General	J. Franke et al.	Nov. 2, 2005
RSA200	200	663	GNFS	General	J. Franke et al.	May 9, 2005
$(10^{211}-1)/9$	211	699	SNFS	C	S. Cavallar et al.	Apr. 8, 1999
$2^{773}+1$	233	774	SNFS	C	S. Cavallar et al.	Nov. 14, 2000
$2^{809}-1$	244	809	SNFS	M	J. Franke et al.	Jan. 3, 2003
$2^{1642}+1$	248	822	SNFS	C	K. Aoki et al.	Apr. 4, 2004
$6^{353}-1$	275	911	SNFS	C	K. Aoki et al.	Jan. 24, 2006
$2^{1039}-1$	313	1039	SNFS	M	K. Aoki et al.	May 21, 2007

Notes: MPQS – multiple polynomial quadratic sieve (a variation of quadratic sieve (QS), GNFS – general number field sieve, FPGA – field programmable gate array, ECM – elliptical curve method, SNFS – special number field sieve. RSA – RSA security challenge number. C – Cunningham number. c – Cunningham number co-factor. P – partition number. M – Mersenne number.

Source: Brent (2000), Marain (2002) and RSA (2009)

Table 3 Fermat number factorisations by year

Fermat number	*Size of factors*	*Factoring algorithm utilised*	*Year*	*Individual factors found by researcher(s)*
F_5	3,7	Trial division	1732	Leonhard Euler
F_6	6,14	Manual computation	1880	Fortuné Landry
F_7	17,22	CFRAC	1970	M.A. Morrison and J. Brillhart
F_8	16,62	Pollard's rho	1980	R.P. Brent and J.M. Pollard
F_9	7,	Trial division	1903	A.E. Western
	49,99	SNFS	1990	Arjen K. Lenstra et al.
F_{10}	8,	Trial division	1953	John L. Selfridge
	10,	Trial division	1962	John Brillhart
	40,252	ECM	1995	Richard P. Brent
F_{11}	6,6,	Trial division	1899	Allan J. C. Cunningham
	21,22,564	ECM	1988	Richard P. Brent
F_{12} – partially factored	6,	Trial division	1877	I. M. Pervushin and E. Lucas
	8,8,	Trial division	1903	A.E. Western
	12,	Trial division	1974	
	16,	Pollard's $p-1$	1986	
	1187	Proven to be composite	n/a	
				J.C. Hallyburton and J. Brillhart, Robert Baillie, n/a

Notes: F_{0-4}(3, 5, 17, 257, 65537) are prime numbers as discovered by Pierre de Fermat. CFRAC – continued fraction factorisation method, ECM – elliptical curve method, SNFS – special number field sieve.

Source: Brent (1999)

Table 3 also reports on record factorisations but only in the context of special form numbers known as Fermat's numbers. It summarises factoring breakthroughs made from early 1700s till today and lists utilised factorisation algorithm for every record as well as the specific algorithm used to factor the number in question.

The presented achievements in IF have a direct and immediate influence on security systems based on cryptographic encryption which relies on assumed difficulty of factoring. As our ability to factor large integers increases so does the minimum size of the keys believed to be safe against factorisation attacks both in symmetric and asymmetric cryptosystems. Table 4 outlines what is believed to be a secure keys size in bits for symmetric and asymmetric systems with respect to available hardware and algorithmic resource in are given time period, including projections up to the year 2050 (Lenstra and Verheul, 2001).

Table 4 Suggested secure symmetric and asymmetric key sizes by year

Year	*Key size in bits (symmetric)*	*Key size in bits (asymmetric)*
1985	59	488
1990	63	622
1995	66	777
2000	70	952
2005	74	1149
2010	78	1369
2015	82	1613
2020	86	1881
2025	89	2174
2030	93	2493
2035	97	2840
2040	101	3214
2045	105	3616
2050	109	4047

Source: Lenstra and Verheul (2001)

2 Genetic algorithm

Inspired by evolution, genetic algorithms constitute a powerful set of optimisation tools that have demonstrated good performance on a wide variety of problems including some classical NP-complete problems such as the travelling salesperson problem (TSP) and multiple sequence alignment (MSA). GAs search the solution space using a simulated 'Darwinian' evolution that favours survival of the fittest individuals. Survival of such population members is ensured by the fact that fitter individuals get a higher chance at reproduction and survive to the next generation in larger numbers (Goldberg, 1989).

GAs have been shown to solve linear and nonlinear problems by exploring all regions of the state space and exponentially exploiting promising areas through standard genetic operators eventually converging populations of candidate solutions to a single global optimum. However, some optimisation problems contain numerous local optima which are difficult to distinguish from the global maximum and therefore result in suboptimal solutions. As a consequence, several population diversity mechanisms have been proposed to delay or counteract the convergence of the population by maintaining a diverse population of members throughout its search.

The proposed GA is generational (Yampolskiy et al., 2004):

1 a population of N possible solutions is created

2 the fitness value of each individual is determined

3 repeat the following steps N/2 times to create the next generation
   a choose two parents using tournament selection
   b with probability p_c, crossover the parents to create two children, otherwise simply pass parents to the next generation
   c with probability p_m for each child, mutate that child
   d place the two new children into the next generation

4 repeat new generation creation until a satisfactory solution is found or the search time is exhausted.

2.1 Solution representation

Potential solutions representing successive approximations to factors p and q are represented as a single string of digits holding values for numbers comprising both p and q with potential leading zeroes for each. Sample string representing makeup of an individual member of the genetic pool follows:

$$\left[p_1\ p_2\ p_3\ p_4 p_{|N|/2}\ q_1\ q_2\ q_3\ q_4 \cdots q_{|N|/2} \right] \quad (1)$$

Since we are interested in solving the most difficult cases of IF in which size of p and q in terms of number of digits is essentially the same, without the loss of generality we are assuming that both p and q will be equal in size to 1/2 size of N. This assumption can be trivially overcome should the need arise to factor numbers in different (less challenging) format.

2.2 Creation of the initial population

Initial population necessary to begin the genetic search can be generated in a number of ways. For example, after the size of the initial population is decided on, the necessary number of individuals can be produced in a random fashion. Pseudo-random number generator can be used to decide the value of every element in the string representing each potential solution. Alternatively, a number of individuals in the population can be seeded with pre-computed high-fitness strings obtained from prior runs of the genetic algorithm.

2.3 Fitness function

The most important part of any evolutionary algorithm and certainly most difficult to design is the fitness function. A good fitness function provides progressive rewards to partial solutions while limiting the value of strings which are converging towards local maxima or do not evolve towards an acceptable solution. Our proposed fitness evaluations schema measures the degree of similarity between the number being factored and the product of evolving factors p and q in terms of placement and value of constituting numbers as well as overall properties of the numbers such as size and parity.

2.4 Crossover operation

A fundamental property of any evolutionary system is the ability to exchange genetic material between superior individuals in the population. In our case it is necessary in order to pass the good properties of the parent solutions on to their offspring. We investigated a number of crossover methods which have the necessary property of preserving potential partial solutions while allowing for significant genetic diversity to remain in the solution pool. In general crossover operation is accomplished by exchanging subsets of varying size of numbers between individuals selected to be "parents" based on their superior fitness.

2.5 Mutation operation

Occasional genetic mutations are necessary to provide genetic diversity to an otherwise overly homogeneous population produced by the selection of the fittest individuals and allow for the broader exploration of search space to take place. By applying the mutation operation a certain number of times we can achieve any degree of genetic diversity we desire. In the case of binary representation of solution strings mutation can be achieved by simply flipping a random bit in the chromosomal representation of p or q. In the case of solutions represented in non-binary bases, mutation operation transfers a chosen digit to another digit legitimate under the selected encoding.

2.6 Proof of concept

To date no experimental results on application of genetic algorithms towards IF problem have been published, though some works utilising computational intelligence approaches has appeared (Chan, 2002), (Meletiou et al., 2002). This can be explained by difficulty evolution inspired algorithms face in solving problems in which the only measure of fitness is a binary – correct/incorrect result, since there is no possibility for the algorithm to converge on a solution via hill climbing. While it is easy to assume that solutions to IF problem only exhibit such binary information (a number is a factor or is not a factor), it is actually not the case as can be seen from the following trivial example, which demonstrates a series of partial solutions with gradual increase in fitness value of factor-approximating numbers.

In the example in Table 5 given a number $N = 4885944577$, it is obvious to see that partial information about the numbers comprising p and q is indeed leaked by the relationship between number we are given to factor, and product of potential candidates to values of p and q. In fact as the size of N increases the degree of inter-influence of digits of N located at a distance from each other further diminishes allowing for a more independent evaluation of partial solutions.

Table 5 Partial solutions to a sample factorisation problem with increasing fitness

Potential solution	*Approximation of N*	*Fitness: number of matching digits*
$p = 14531$, $q = 73341$	1065718071	1
$p = 54511$, $q = 43607$	2377061177	2
$p = 84621$, $q = 43637$	3692606577	3
$p = 84621$, $q = 41637$	3523364577	4
$p = 84621$, $q = 51637$	4369574577	5
$p = 94621$, $q = 51637$	4885944577	10

3 Test data

Test data was readily available in order to properly evaluate performance of our algorithm. In addition to being able to quickly generate test numbers of any length, a third party set of challenge numbers was also available. Known as RSA numbers (after the Rivest, Shamir and Adleman public encryption algorithm) they provided an independent third party evaluation of our genetic algorithm.

RSA numbers are difficult to-factor composite numbers having exactly two prime factors (aka, semiprimes), similar to the modulus of an RSA key pair. The RSA challenge numbers were generated by the RSA Laboratories, to learn about the actual difficulty of factoring large numbers of the type used in RSA keys. There are 54 RSA number ranging in size from 100 decimal digits (330 binary) to 617 decimal digits (2048 binary). The first RSA numbers generated, from RSA-100 to RSA-500, were labeled according to their number of decimal digits; later, beginning with RSA-576, binary digits are counted instead. An exception to the rule is RSA-617, which was created prior to the change in the numbering scheme (Brent, 2000). The RSA challenge numbers were generated using a secure process that guarantees that the factors of each number cannot be obtained by any method other than factoring the published value. No one, including RSA Laboratories, knows the factors of any of the challenge numbers.

The generation took place on a Compaq laptop PC with no network connection of any kind. The process proceeded as follows (retrieved 5 February 2009):

- First, 30,000 random bytes were generated using a ComScire QNG hardware random number generator, attached to the laptop's parallel port.

- The random bytes were used as the seed values for the B_GenerateKeyPair function, in version 4.0 of the RSA BSAFE library. The private portion of the generated key pair was discarded. The public portion was exported, in DER format to a disk file.
- The moduli were extracted from the files and converted to decimal representation for posting on the web.
- The laptop's hard drive was destroyed.

4 Experimental results

Described genetic algorithm was programmed in Java, as a part of general framework, capable of being adapted to evolve solutions to numerous optimisation problems in addition to IF. We have begun our experiments while still developing the framework and the algorithm did remarkably well factoring small integers as the part of the debugging process. Once the algorithm was completed we decided to conduct an experiment with the smallest of the RSA challenge numbers but the algorithm failed to converge on a solution regardless of the population size, crossover type, mutation rate or the number of generations evolved. In fact, an improvement in fitness level of the best individual in each generation showed very quick immediate progress only to end up in a local maxima point and be unable to escape it even in the long run.

Our analysis shows that the difficulty comes from local maxima points pervasive in the IF domain. The rest of this section attempts to address the nature of our finding. A function f defined for real numbers is said to have a local maximum at the point y, if there exists some $\varepsilon > 0$, such that $f(y) \geq f(x)$ when $|x - y| < \varepsilon$. A function has a global maximum point at y, if $f(y) \geq f(x)$ for all x. Any global maximum point is also a local maximum point; however the opposite is not true, a local maximum point doesn't have to be a global maximum point (see Figure 1). In semiprime numbers such as those presented by RSA challenge (numbers ending in 1, 3, 7 or 9) the local maxima points come from numbers which are also a product of at least two integers and which match the number to be factored in terms of its constituting digits to a certain degree. For example, we might be interested in factoring number $N = 15194323 = 3889 * 3907$, but we might run into a local maxima point represented by a different number matching almost all of N's digits except one: $15794323 = 3733 * 4231$.

Such local maxima are frequent in the IF domain; in fact, we were able to show that given any semiprime number N with n decimal digits there are exactly $2 * 10^{n-1}$ unique pairs of numbers p_i and q_i up to n digits each, which if multiplied, have a product matching all n digits of N precisely. Same relationship also holds true for the prime numbers which obviously don't have a global maxima point (see Figure 2). For example for $N = 71$ ($n = 2$), that number is $2 * 10^{2-1} = 2 * 10^1 = 20$, or to list explicitly: (01 * 71 = 71), (03 * 57 = 171), (07 * 53 = 371), (09 * 19 = 171), (11 * 61 = 671), (13 * 67 = 871), (17 * 63 = 1071), (21 * 51 = 1071), (23 * 77 = 1771), (27 * 73 = 1971), (29 * 99 = 2871), (31 * 41 = 1271), (33 * 87 = 2871), (37 * 83 = 3071), (39 * 89 = 3471), (43 * 97 = 4171), (47 * 93 = 4371), (49 * 79 = 3871), (69 * 59 = 4071), (81 * 91 = 7371). If plotted local maxima p/q-pairs form a pattern, different for each specific IF problem. Some examples are given in the figure below. It is easy to see that even those highest of all the local maxima points already comprise 20% of the total search space and that percentage does not even include smaller local maxima points.

The number of local maxima points associated with IF will completely overwhelm any hill climbing algorithm making it unlikely for any such algorithm to reliably find solutions to non-trivial cases of IF. Realising that our approach without fundamental modifications is unlikely to successfully factor a hundred-plus digit RSA numbers we have generated a sequence of progressively larger semiprimes (beginning with $N = 4$) in order to determine overall capability of our approach. The best result achieved by our algorithm was a factorisation of a 12D semiprime (103694293567 = 143509 * 722563). This result took a little over 6 hours on a Intel 2 core 1.86 GHz processor with 2 GB of RAM and was achieved with a population consisting of 500 individuals, two point crossover, mutation rate of 0.3% and genome represented via decimal digits.

Additional examples of integers factored by our algorithm along with complete information about the specific run and evolving fitness values follow. Runs 1 and 2 are successfully factorisations of a 6D and 8D numbers. 3rd run represents an incomplete attempt to factor RSA100 challenge number which is two large for our algorithm to handle.

Run 1 : Factoring : 213443
Fitness value of 9.0, 829 * 357 = 295953
Fitness value of 10.0, 589 * 357 = 210273
Fitness value of 11.0, 569 * 757 = 430733
Fitness value of 12.0, 587 * 364 = 213668
Fitness value of 13.0, 281 * 761 = 213841
Fitness value of 14.0, 407 * 549 = 223443
Solution : 463 * 461 = 213443

Run 2 : Factoring : 15194323
Fitness value of 10.0, 9257 * 5768 = 53394376
Fitness value of 12.0, 3680 * 4304 = 15838720
Fitness value of 13.0, 2821 * 3663 = 10333323
Fitness value of 14.0, 4081 * 3683 = 15030323
Fitness value of 16.0, 5951 * 4973 = 29594323
Fitness value of 17.0, 8151 * 4373 = 35644323
Fitness value of 18.0, 3351 * 4773 = 15994323
Fitness value of 19.0, 4399 * 4477 = 19694323
Solution : 3889 * 3907 = 15194323

Run 3 : Factoring : RSA100 =
1522605027922533360535618378132637429718068114961380688657908494580122963258952897654000350692006139
Fitness value of 47.0, 385362477423618583322117591203660621092129552542 73 *
887918262619728492521722692327998071662875943598 49 =
342170381432813757462452376810210772919498229604156992046980147443858143563158497211188812445688 4777
Fitness value of 59.0, 96378627536567198869282246326871973487323674615512*
756142639572337896726905882203838863276542008016 43 =
728759898238591317275616187119711873682706918406685669755103289762500712024496862913650444020288 6216
Fitness value of 62.0, 48199401972071282251803726368545070325588916526303 *
707085665151382057764653072380063377945463367814 13 =
340811062033208586919771633715123578968537354429653710556989199561182437112805088624308305376006139
Fitness value of 63.0, 21080432533027225787136926665545897372079112976303 *
676829749576632775709684979466264984784621368314 13 =
142678638722959197643544481892025730878869983281047789405192544393589549475833620433366232017500 6139
Fitness value of 67.0, 45112401168824468808230926665549874366675110976303 *
676687769566832775408684979466264684784235768314 13 =
305270101267360097103266524899389556464907424654724631980526864994542976623700010810418656166900 6139
Fitness value of 68.0, 25743321584521229357380737164870882613268481976303 *
586857146226571762488584209706174184864419568314 13 =
151076522394850361807992607414453395781627863232732181424126443313066599218026209750296537403200 6139
Fitness value of 70.0, 25813763273880186803388247655718727008128401976303 *
589851346977071762085597006721114184838566175683141 3 =
152262830376454939967805788056343112588193407470169713452791763800603941569051148531794444039200 6139
Fitness value of 76.0, 25813743413280242019730531661406881034808401976303 *
589854406470274662788716229216914184861418568314 13 =
152263502998163791470792035272056701619706995777386333165733885258081178695695079060116635069200 6139
Fitness value of 78.0, 25813263419520986803388247655718887008128401976303 *
589854407470274662788716229216914184861418568314 13 =
152260671991956676450589364706305735671306867379813027659810556260552940964874989481706751069200 6139
Fitness value of 82.0, 25813253189380486182388462535798583024108401976303 *
589854456250254848324377959363961364861418568314 13 =
152260624240721834157954425275860881286974241804370142658720375432734242341595135498504725069200 6139
Fitness value of 87.0, 25813253119499621153488161635698083525108401976303 *
589852246410706882676443919364102884828418568314 13 =
152260053397050386539817837910498936141871875458016382811011219748313822724967553455716035069200 6139
…

While obviously not a practical approach to factor integers of the size required in modern cryptographic applications our approach did outperformed algorithms based on genetic programming (Finkel, 2003; Chan, 2002) and neural networks (Jansen and Nakayama, 2005; Meletiou et al., 2002; Laskari et al., 2006) as well as the best results reported so far for the Shor's algorithm (Shor, 1997). We will continue with our experiments and hope that the next version of our algorithm will be competitive with other approaches as well. Next version of our algorithm will be based on distributed voting between members of the genetic pool with respect to specific digits making up the solution. For example if 45% of all top ranked potential solution have digit '8' in location 2 eight will be selected as a part of the final solution in the said location for this particular iteration of the genetic algorithm.

The proposed genetic algorithm has outperformed genetic programming approach because GP attempts to come up with a universal solution for all instances of IF instead of trying to find an optimal solution for a specific problem which is easier as our algorithm demonstrates by producing better results. Neural Network approach has its own shortcomings namely its dependency on specific digits as first layer inputs and structure designed to accommodate

numbers comprised of a specific amount of digits. Such rigidness prevents NN from performing competitively with our GA approach.

Figure 1 Global maximum and local maxima

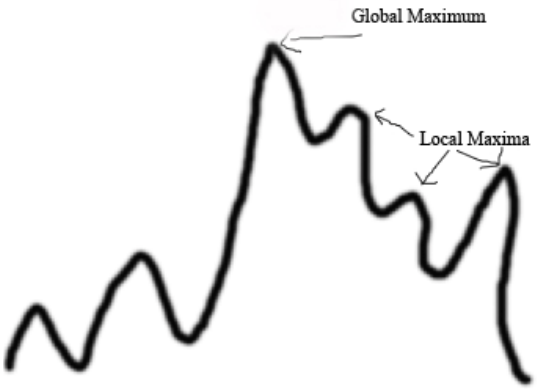
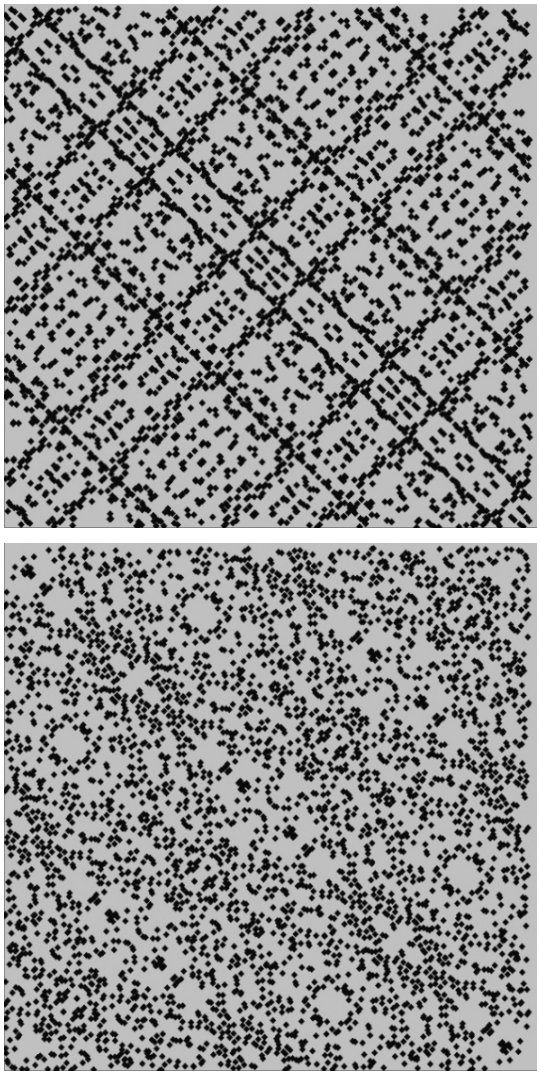


Figure 2 Top – local maxima distributions for 7841 (prime number) and bottom – 8883 (semiprime)



5 Conclusions and future work

We have developed a Java-based genetic algorithm framework easily adaptable to any optimisation problem and applied it to the problem of IF. We were able to demonstrate that the IF problem is not an all-or-nothing problem with only right/wrong answer and no partial solutions. Unfortunately, IF also has an extremely high number of local maxima points which we were able to show represent well over 20% of the total solution space. This property makes a straightforward hill climbing genetic algorithm incapable of solving a non-trivial instance of an IF problem.

In the future we propose to develop a mathematical basis for analysis and evaluation of different types of local maxima associated with specific IF problems and incorporation of such data into the fitness functions in order to avoid convergence of the genetic algorithm to a suboptimal solution. More specifically we suggest investigating the types of local maxima pattern distributions seen for different sub-types of IF challenge numbers as well as investigate statistically likely locations of global maximum within such patterns. Additionally, patterns in spatial distribution of local maxima points can be explored as a tool for image sub-sampling or data compression.

One of the challenges associated with successful application of genetic algorithms is very long execution times required to evolve acceptable solutions to real world problems. GAs can be very demanding in terms of computational load and memory requirements with fitness evaluation usually being the most expensive step. Numerous parallel genetic algorithms have been proposed and in a multitude of problem domains they demonstrate superior performance in comparison to serial GAs (Nowostawski and Poli, 1999). This happens both because of larger amount of computational resources being available and also because of higher degree of genetic diversity producible by multiple independent populations evolving simultaneously and only periodically sharing code of selected (not necessarily fittest) individuals. In the future experiments we will port our genetic algorithm framework to a parallel architecture along side with local maxima conscious fitness function.

References

Lenstra, A.K. and Lenstra, H.W. Jr (Ed.) (1993) *The Development of the Number Field Sieve*, Springer-Verlag, New York.

Adi Shamir, E.T. (2003) 'Factoring large numbers with the TWIRL device', *Crypto – The 23rd Annual International Cryptology Conference*, Santa Barbara, California, USA, pp.1–26.

Brent, R.P. (1999) 'Factorization of the tenth Fermat number', *Mathematics of Computation*, Vol. 68, pp.429–451.

Brent, R.P. (2000) 'Recent progress and prospects for integer factorisation algorithms', *Computing and Combinatorics: Sixth Annual International Computing and Combinatorics Conference*, Sydney, Australia, pp.3–22.

Bressoud, D.M. (1989) *Factorizations and Primality Testing*, Springer-Verlag, New York.

Chan, D.M. (2002) 'Automatic generation of prime factorization algorithms using genetic programming', in *Genetic Algorithms and Genetic Programming at Stanford*. pp.52–57. Stanford Bookstore. Stanford, California.

Chang, W-L., Guo, M. and Ho, M.S-H. (2005) 'Fast parallel molecular algorithms for DNA-based computation: factoring integers', *IEEE Transactions on Nanobioscience*, Vol. 4.

Dixon, J.D. (1981) 'Asymptotically fast factorization of integers', *Math. Comput.*, Vol. 36, pp.255–260.

Elkenbracht-Huizing, R-M. (1997) 'Factoring integers with the number field sieve', PhD Thesis, Leiden University.

Finkel, J.R. (2003) 'Using genetic programming to evolve an algorithm for factoring numbers', in Koza, J.R. (Ed.): *Genetic Algorithms and Genetic Programming at Stanford*, Stanford Bookstore, Stanford, California.

Goldberg, D.E. (1989) Genetic *Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Pub. Co.

Jansen, B. and Nakayama, K. (2005) 'Neural networks following a binary approach applied to the integer prime-factorization problem', *IEEE International Joint Conference on Neural Networks (IJCNN)*, 31 July, pp.2577–2582.

Knuth, D.E. (1981) *The Art of Computer Programming*, Addison-Wesley.

Laskari, E.C., Meletiou, G.C., Tasoulis, D.K. and Vrahatis, M.N. (2006) 'Studying the performance of artificial neural networks on problems related to cryptography', *Nonlinear Analysis: Real World Applications*, Vol. 7, pp.937–942.

Lehmer, D.H. and Powers, R.E. (1931) 'On factoring large numbers', *Bulletin of the American Mathematical Society*, Vol. 37, pp.770–776.

Lenstra, A.K. and Verheul, E.R. (2001) 'Selecting cryptographic key sizes', *Journal of Cryptology*, Vol. 14, pp.255–293.

Lenstra, H.W. (1987) 'Factoring integers with elliptic curves', *Annals of Mathematics*, Vol. 2, pp.649–673.

Marain, F. (2002) 'Thirty years of integer factorization', in F. Shyzak (Ed.): *Algorithms Seminar 2000–2001*, INRIA, pp.77–80.

Maurer, U. and Rueppel, R. (Eds.) (1992) *Factoring with an Oracle*, Springer-Verlag.

McKee, J. (1996) 'Turning Euler's factoring method into a factoring algorithm', *Bulletin of the London Mathematical Society*, Vol. 4, pp.351–355.

McKee, J. (1999) 'Speeding Fermat's factoring method', *Mathematics of Computation*, Vol. 68, pp.1729–1737.

Meletiou, G., Tasoulis, D.K. and Vrahatis, M.N. (2002) 'A first study of the neural network approach to the RSA cryptosystem', *IASTED 2002 Conference on Artificial Intelligence*, Banff, Canada, pp.483–488.

Nowostawski, M. and Poli, R. (1999) 'Parallel genetic algorithm taxonomy', *Third International Conference on Knowledge-Based Intelligent Information Engineering Systems*, Adelaide, SA, Australia, pp.88–92.

Pollard, J.M. (1974) 'Theorems of factorization and primality testing', *Proceedings of the Cambridge Philosophical Society*, Vol. 76, pp.521–528.

Pollard, J.M. (1975) 'A Monte Carlo method for factorization', *BIT Numerical Mathematics*, Vol. 15, pp.331–334.

Pomerance, C. (1996) 'A tale of two sieves', *Notices of the AMS*, Vol. 43, pp.1473–1485.

Rivest, R.L. and Shamir, A. (1986) 'Efficient factoring based on partial information', *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology-EUROCRYPT '85*, Springer-Verlag, Linz, Austria, pp.31–34.

RSA (2009) 'RSA numbers', *Wikipedia*, *The Free Encyclopedia*, 5 February, available at http://en.wikipedia.org/wiki/RSA_numbers.

Shamir, A. (1999) 'Factoring large numbers with the TWINKLE device (extended abstract)', *Workshop on Cryptographic Hardware and Embedded Systems (CHES '99)*, Worcester, Massachusetts, USA, pp.2–12.

Shor, P.W. (1997) 'Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer', *SIAM Journal Sci. Statist. Computing*, Vol. 26, pp.1484–1509.

Williams, H.C. (1982) 'A p + 1 method of factoring', *Math. Comp.*, Vol. 39, pp.225–234.

Yampolskiy, R., Anderson, P., Arney, J., Misic, V. and Clarke, T. (2004) 'Printer model integrating genetic algorithm for improvement of halftone patterns', *Western New York Image Processing Workshop (WNYIPW)*, IEEE Signal Processing Society, Rochester, NY.