Factoring as Optimization

Christopher J.C. Burges

August, 2002

Technical Report
MSR-TR-2002-83

The factoring of biprimes is proposed as a framework for exploring unconstrained optimization algorithms. A mapping from a given factoring problem to a positive degree four polynomial F is described. F has the properties that the global minima, which specify the factors, are at $F = 0$, and that all coefficients can be chosen to be of order unity. The factoring of biprimes is an attractive test bed because a large number of different F functions can be generated easily, in such a way that the complexity of the resulting minimization problem can be easily controlled. Representing the problem in this form also gives interesting perspectives on why factoring is hard.

This framework suggests new approaches for optimization algorithms aimed at solving combinatorically hard problems. In this paper a new unconstrained optimization algorithm, 'curvature inversion descent' (CID), is described. It has the property that local minima can often be deterministically escaped. CID is illustrated with an 8-dimensional minimization problem generated by factoring a small biprime. For that problem, gradient descent with restarts finds the solution 31% of the time; with CID, this increases to 88% of the time.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1 Introduction

Factoring large biprimes[1] is known to be a hard problem [5]. However since biprimes are easy to generate, this property can be inverted to give a means of generating unconstrained optimization problems whose complexity can be easily controlled. The factorization problem can be cast as the minimization of a degree four polynomial $F : \mathbf{x} \in \mathcal{R}^d \mapsto \mathcal{R}$ with the following properties: $F \geq 0$, and the lower bound $F = 0$ is met at the global minima; the value of $\mathbf{x}$ at a global minimum specifies the biprime's factors; the coefficients appearing in F can be chosen to be of order 1, thus ensuring the smoothness of F and of its derivatives; and there are at most two global minima (and the problem can often be formulated so that the global minimum is unique). Factoring is an interesting problem, and it is instructive to ask how the computational complexity is encapsulated in this formulation. The dimension d of the space over which F is defined increases logarithmically with the size of the biprime. The factoring of any composite number can be cast in this form, although then there will be as many distinct global minima as there are ordered pairs of factors.

The polynomial F is a sum of terms of two types, those arising from sum constraints, and those arising from binarizing constraints. These are referred to as sum terms and binarizing terms below. Sum terms are constructed by expressing the factors in terms of binary variables, computing a complete set of equations that those variables must satisfy using simple long multiplication, and by taking each such equation (whose right hand side is zero) and squaring it. The sum term equations are satisfiability constraints, but here, as opposed to SAT, the relations are algebraic rather than Boolean. The d binarizing terms each have the form $x^2(1-x)^2$, whose minimum is at zero when $x \in \mathcal{R}$ takes the value 0 or 1. Note that the coefficient of each sum and binarizing term in F must be positive, but beyond that, can be chosen arbitrarily; this property of F will be used in the algorithms described below.

Vectors are represented in bold font. The binary expansion of an integer I is denoted by $B(I)$ and the number of bits in that expansion is denoted by $N_I \equiv 1 + \lfloor \log_2(I) \rfloor$. Each term in F is indexed; the set of indices of sum terms will be denoted $\mathcal{S}$, and that of binarizing terms, $\mathcal{B}$. Throughout the paper it is assumed that the factors are odd.

2 Factoring as Optimization

The problem of factoring the biprime $T \equiv 91 = 13 \times 7$ will be used as a simple example throughout. To further simplify the description, for this example it is assumed that

[1] A biprime is a product of two primes.

both factors are known to have at most 4 bits. The 'longhand' binary multiplication is shown below, with the biprime written at the top. In Equations (1) through (12), the variables are binary: $\{x, y, z\} \in \{0, 1\}$.

$$\begin{array}{ccccccc}
1 & 0 & 1 & 1 & 0 & 1 & 1 \\
 & & & x_1 & x_2 & x_3 & 1 \\
 & & & y_1 & y_2 & y_3 & 1 \\
\hline
 & & & x_1 & x_2 & x_3 & 1 \\
 & & y_3x_1 & y_3x_2 & y_3x_3 & y_3 & 0 \\
 & y_2x_1 & y_2x_2 & y_2x_3 & y_2 & 0 & 0 \\
y_1x_1 & y_1x_2 & y_1x_3 & y_1 & 0 & 0 & 0
\end{array}$$

Adding the columns and introducing binary variables z_i to represent the 'carries' gives the following equations:

$$\begin{aligned}
x_3 + y_3 &= 1 && (1)\\
x_2 + y_3x_3 + y_2 &= 2z_1 && (2)\\
x_1 + y_3x_2 + y_2x_3 + y_1 + z_1 &= 1 + 2z_2 + 4z_3 && (3)\\
y_3x_1 + y_2x_2 + y_1x_3 + z_2 &= 1 + 2z_4 && (4)\\
y_2x_1 + y_1x_2 + z_3 + z_4 &= 2z_5 + 4z_6 && (5)\\
y_1x_1 + z_5 &= 1 && (6)\\
z_6 &= 0 && (7)
\end{aligned}$$

Note that these equations are not over the finite binary field; they are algebraic equations in Boolean variables over the field of reals. They will be referred to below as the 'factoring equations'; the general form of the above decomposition will be referred to as the 'factoring decomposition'. As will be described further below, solving these simultaneous equations amounts to factoring T.

For biprimes the factoring equations will have two solutions (if the factors are different), since swapping the factors gives a different solution. However the problem can sometimes be converted to one with a unique solution by eliminating sufficiently many variables so that the ordering becomes unique. For example[2], Eq. (1) gives $x_3 = \bar{y}_3$, so choosing the values $x_3 = 0$ and $y_3 = 1$ specifies the order and gives a problem with a unique solution. The above equations then reduce to

$$\begin{aligned}
x_2 + y_2 &= 2z_1 && (8)\\
x_1 + x_2 + y_1 + z_1 &= 1 + 2z_2 && (9)\\
x_1 + y_2x_2 + z_2 &= 1 + 2z_3 && (10)\\
y_2x_1 + y_1x_2 + z_3 &= 2z_4 && (11)\\
y_1x_1 + z_4 &= 1 && (12)
\end{aligned}$$

[2] $\bar{y} \equiv$ not y

where z_3 in (3) has been set to zero (since $y_3x_2 + y_2x_3$ is at most 1) and where the remaining z's have been renamed accordingly. Equations (8) - (12) can be mapped into an optimization problem as follows. Define the objective function:

$$\begin{aligned} F(T) &= \alpha_1(x_2 + y_2 - 2z_1)^2 + \alpha_2(x_1 + x_2 + y_1 + z_1 - 1 - 2z_2)^2 + \\ &\quad \alpha_3(x_1 + y_2x_2 + z_2 - 1 - 2z_3)^2 + \alpha_4(y_2x_1 + y_1x_2 + z_3 - 2z_4)^2 + \\ &\quad \alpha_5(y_1x_1 + z_4 - 1)^2 + \sum_{i=1}^{8} \alpha_{i+5} w_i^2 (1 - w_i)^2 \end{aligned} \tag{13}$$

where the w_i are shorthand for the aggregated $\{x, y, z\}$ variables, the w_i now take values in the reals, and $\alpha_i \in \mathcal{R}_+$. The terms in w_i are binarizing: they vanish only on the vertices of the unit cube, so F has a global minimum at $F = 0$ for all $\alpha_i \in \mathcal{R}_+$ when the w_i satisfy Eqns. (8) - (12) and are binary. Note that the $\alpha_i \in \mathcal{R}_+$ can be chosen arbitrarily: below they will be referred to as the free variables. The objective function generated in this way will be referred to as a 'factoring potential' for a given composite positive integer.

2.1 Comparison with SAT

This construction is in a sense a satisfiability problem, but it differs from the standard SAT. SAT equations are over a finite binary field (and are usually expressed in conjunctive or disjunctive normal form), whereas here the equations are over the reals (and even before mapping the variables to the reals, the equations take values in the integers, not in $\{0, 1\}$). Factoring has been used as a SAT problem generator [3, 2], but there, at each stage of the computation, new binary variables are introduced to represent partial sums, and so the number of variables used is larger, and the method remains Boolean throughout (for example, for a SAT instance generator, factoring 15 gives a problem with 35 variables, whereas the factoring decomposition above requires 4 variables[3]).

2.2 Formal Construction

Lemma 1: In any factoring of a positive integer T into two factors F_1 and F_2 with $F_1 \geq F_2$, then $N_{F_2} \leq \lceil \frac{N_T}{2} \rceil$.

Proof: Suppose otherwise, that is that $N_{F_2} > \lceil \frac{N_T}{2} \rceil \Rightarrow N_{F_2} \geq \lceil \frac{N_T}{2} \rceil + 1$. Then

$$F_2 \geq 2^{N_{F_2} - 1} \geq 2^{\lceil \frac{N_T}{2} \rceil}$$

so

$$F_2^2 \geq 2^{2\lceil \frac{N_T}{2} \rceil} \geq 2^{N_T} > T$$

[3]Or 3 after an elimination of the form $x_1 = \bar{y}_1$.

but this contradicts $T = F_1 F_2 \geq F_2^2$.
□

Denoting the elements of $B(T)$ by b_i, Lemma 1 allows the general decomposition shown in Table 1. Each column below the horizontal line in Table 1, together with carry variables from previous terms, sums to give an equation over binary variables, where the b_i, and any new carry variables z, appear on the right hand side.

			b_1	b_2	$\cdots$	$b_{N_T - \lceil \frac{N_T}{2} \rceil + 1}$	$\cdots$	$b_{N_T - 1}$	b_{N_T}
			x_1	x_2	$\cdots$	$x_{N_T - \lceil \frac{N_T}{2} \rceil + 1}$	$\cdots$	$x_{N_T - 1}$	1
						y_1	$\cdots$	$y_{\lceil \frac{N_T}{2} \rceil - 1}$	1
			x_1	x_2		$\cdots$		$x_{N_T - 1}$	1
		$y_{\lceil \frac{N_T}{2} \rceil - 1} x_1$	$y_{\lceil \frac{N_T}{2} \rceil - 1} x_2$			$\cdots$		$y_{\lceil \frac{N_T}{2} \rceil - 1}$	
						$\cdots$			
$y_1 x_1$	$\cdots$	$\cdots$			y_1				

Table 1: Long multiplication with variables. The product is written in binary form on top. The last row is shifted from the first by $\lceil \frac{N_T}{2} \rceil - 1$ columns. In the text, the columns below the horizontal line containing $\{x, y\}$ variables are called factoring columns.

Remark 2: In general the number of z variables appearing on the right hand size of a given factoring equation is equal to the number of bits required to specify the integer found by setting all left hand $\{x, y, z\}$ variables equal to one, subtracting the $\{0, 1\}$ constant from the right hand side, and if the resulting number is odd, subtracting one; the z variables are just the binary expansion of that number, omitting the lowest order bit. In some cases the z variables can be further eliminated by using the equations generated by the higher order bits of T (cf. Eq. (7)).

Lemma 3: The mapping between binary solutions of the factoring equations and (positive integer) solutions of the factoring problem is an isomorphism.

Proof: Let the factoring equations in the general case be ordered as exemplified by Eqs. (1) - (7). Then in the first equation that contains one or more z variables, those variables appear on the right hand side and are completely determined by the values of the $\{x, y\}$ variables on the left hand side, since the right hand variables (together with the constant, if present) form a binary representation of the number on the left hand side, and since the binary representation of any number is unique. Thus the z variables appearing in the next equation are also completely determined by the $\{x, y\}$ variables in that and in the previous equations, since those z variables on the left are determined by the previous equations, and those on the right are determined as above. Thus the z variables appearing in all equations are uniquely determined by the values taken by the $\{x, y\}$ variables. Hence solutions of the factoring equations are in 1-1

correspondence with the values of the $\{x, y\}$ variables alone, and by construction, the values of those variables are in 1-1 correspondence with the original positive integer factors; hence the map from binary solutions of the factoring equations to (positive integer) solutions of the factoring problem is bijective.
□

Remark 4: Lemma 3 does not hold if the variables are allowed to take values in all the reals. In the above example (Eqs. (8) - (12)), set $x_1 = x_2 = 1$. Then the solution set is then determined by the intersection of five hyperplanes in $\mathcal{R}^6$, and so is infinite (since at least one solution is known to exist).

Theorem 5: In the factoring decomposition, a sufficient number of x variables is $n_x = \lfloor \log_2(T) \rfloor$, a sufficient number of y variables is $n_y = \lceil \frac{1}{2}(\lfloor \log_2 T \rfloor + 1) \rceil - 1$, and the number of factoring equations is $n_e = n_x + n_y$. The equations containing the largest number of $\{x, y\}$ monomials contain $1 + n_y$ such terms, and there are $N_T - \lceil \frac{N_T}{2} \rceil + 1$ such equations. The number of z variables is bounded above as

$$n_z \leq n_y(1 + \lfloor \log_2(n_x + n_y) \rfloor) - 1 + \sum_{i=1}^{n_y - 1} \lfloor \log_2 i \rfloor + \sum_{i=n_y}^{n_x} \lfloor \log_2(n_y + i) \rfloor$$

and the maximum coefficient of any term appearing in the factoring equations is bounded above by $n_x + n_y$.

Proof: Since the minimal number of bits required to specify a positive integer I is $N_I \equiv 1 + \lfloor \log_2(I) \rfloor$, then from Table 1, $n_x = N_T - 1 = \lfloor \log_2(T) \rfloor$ is sufficient, and from Lemma 1, $n_y = \lceil \frac{N_T}{2} \rceil - 1 = \lceil \frac{1}{2}(\lfloor \log_2 T \rfloor + 1) \rceil - 1$ is sufficient. The number of factoring equations is equal to the number of factoring columns, $n_e = n_x + n_y$, since any carry variables appearing alone must vanish. It is useful to split the equations into three blocks. Labeling the columns in the table from the left, columns 1 through $\lceil \frac{N_T}{2} \rceil - 1$ will be called the left hand block, columns $\lceil \frac{N_T}{2} \rceil$ through N_T will be called the center block, and columns $N_T + 1$ through $N_T + \lceil \frac{N_T}{2} \rceil - 2$ will be called the right hand block. Then the equations containing the largest number of $\{x, y\}$ monomials appear in the center block, and from Table 1, all such equations contain $n_y + 1$ such terms, and there are $N_T - \lceil \frac{N_T}{2} \rceil + 1$ such equations. Now instead index the equations from the right hand column of Table 1 (underneath $b_{N_T - 1}$), starting at $i = 1$. Since each equation can contain at most one z term from each of the previous equations, and since the first equation contains no z terms on the left hand side, the number of z terms that can appear on the left hand side of the i'th equation is bounded above by $i - 1$. Thus since the number of $\{x, y\}$ monomials in the i'th equation of the right hand block is $i + 1$, the total number of left hand monomials in the i'th equation for the right hand block is bounded above by $2i$. Hence the number of z variables that appear on the right hand sides of equations in the right hand block is bounded above by $\sum_{i=1}^{n_y - 1} \lfloor \log_2 2i \rfloor$, using 0 for the constant appearing on the right hand side, here and below, and noticing that the z variables form the binary expansion (omitting the least significant bit)

of the largest even number less than or equal to the number of multinomials on the left hand side. For the center block, the total number of monomials appearing on the left hand side of each equation is bounded above by $n_y + i$ $(i = n_y, \cdots, n_x)$, and so the maximum number of z variables appearing on the right hand side of all equations in the center block is bounded above by $\sum_{i=n_y}^{n_x} \lfloor \log_2(n_y + i) \rfloor$. For the left hand block, the total number of monomials appearing on the left hand side of each equation is bounded above by $n_x + n_y$ $(i = n_x + 1, \cdots, n_x + n_y)$, and so the number of z variables appearing on the right hand side of all equations in the left hand block is bounded above by $\sum_{i=n_x+1}^{n_x+n_y} \lfloor \log_2(n_x + n_y) \rfloor$. Adding these three terms together and using $\lfloor 1 + s \rfloor = 1 + \lfloor s \rfloor$ $\forall s \in \mathcal{R}$ gives the result. Finally, using the bound $i - 1$ for the number of z variables that can appear on the left hand side of the i'th equation, the total number of monomials on the left hand side of that equation with the maximum number of such monomials is that equation in the center block with maximum index i, with an upper bound of $n_x + n_y$ such monomials (the same bound holds for subsequent equations). Hence the maximum coefficient of the z variable appearing on the right hand side is bounded above by $2^{N_{n_x+n_y}-1} = 2^{\lfloor \log_2(n_x+n_y) \rfloor} \leq n_x + n_y$, and since all $\{x, y\}$ monomials appear with coefficient one, this is also an upper bound on the maximum coefficient of any term appearing in the factoring equations.
□

Remark 6: The above bound on the number of variables in the factoring decomposition grows as $(\log T)(\log \log T)$.

Remark 7: If in fact $N_{F_1} > \lceil \frac{N_T}{2} \rceil$, then the solution to the factoring equations for a biprime, and to the corresponding minimization problem, is unique.

Remark 8: The size of the largest coefficient of a z variable can be reduced by any chosen power of 2 by introducing auxiliary z variables with auxiliary constraints[4].

Remark 9: The above theorem also leads to simple smoothness bounds on the factoring potential, in that bounds on the sizes of all derivatives can be derived from the above bounds on the coefficients of the $\{x, y, z\}$.

Remark 10: In solving a given factoring problem, rather than using the general formulation of Table 1, a more efficient method is to partition the problem by allowing n_x to vary from N_T to the number of bits required to specify the square root of T, and for each such n_x, attempt to solve the set of problems generated by each possible n_y. In that approach the most significant bit of x and y can both be taken to be 1 (which is not true of the simpler formulation above).

The factoring potential has the following interesting property:

[4]For example, $4z$ can be replaced by $2z_a$ and the constraint $z_a = 2z$. Note that z itself would still appear elsewhere in the equations.

Theorem 11: For any factoring potential, for every sum term $\alpha_i s^2$, s satisfies the following condition: if s contains a linear term (one or more of the $\{x, y, z\}$), then $\nabla s^2 = 0 \Rightarrow s = 0$; and if s does not contain a linear term, then $\nabla s^2 = 0$ implies that either $s = 0$, or that $s^2 = 1$ and every variable appearing in s vanishes, in which case $s = 0$ or $s = 1$. Furthermore for $T > 15$, the first condition (s contains a linear term) holds for all but at most two of the sum terms.

Proof: If s contains a linear term, then since the corresponding variable appears only once in s^2, the component of the gradient of s^2 which is the derivative with respect to that linear term will be some nonzero integer times s; hence vanishing of the gradient implies vanishing of s. If s does not contain a linear term, then since its variables appear as only quadratic products, and since each variable appears only once, the vanishing of the gradient of s^2 implies that either $s = 0$, or that every variable appearing in s vanishes, in which case s^2 takes the value of the constant that appears in it (0 or 1). Finally, referring to Table 1, and ordering the factoring equations as in Eq. (1) - (7), the last equation containing $\{x, y\}$ variables may not have a carry term, but for $T > 15$ the second to last will, since the previous equation has three x, y monomials; and for $T > 15$, for all the remaining equations, only the first will have no carry term.
□

Remark 12: Theorem 11 does not hold for the binarizing terms; $\nabla x^2(1-x)^2 = 0 \Rightarrow x \in \{0, \frac{1}{2}, 1\}$.

Remark 13: Theorem 11 shows that the sum terms have the property that, for a given sum term and for any problem of interest ($T > 15$), and for all but at most two of the sum terms, if its gradient vanishes, then that particular sum term is at a global solution ($s = 0$) for the sub-problem of minimizing s^2.

3 Why is Factoring Hard?

It is interesting to examine the difficulty of the factoring problem in terms of the difficulty of finding a global minimum in the factoring potential. Since by construction, the coefficients appearing in the factoring decomposition can all be chosen to be of order one, and since gradient descent algorithms do not themselves introduce numerical limitations which act as a barrier to finding the minima, the difficulty of finding a global minimum cannot arise from numerical limitations alone. Global minima can also be hard to find if the energy landscape is flat everywhere except close to the minima - a 'golf-course' potential. However the construction of such potentials requires the non-vanishing of many orders of derivatives (such as is the case for Gaussians, for example), and so cannot arise for fourth degree polynomials. However one definite source of difficulty is the number of local minima.

3.1 Multiple Minima

Gradient descent algorithms will quickly find local minima of these potentials. In fact since derivatives of the factoring potentials are third degree polynomials, a key component of typical gradient descent algorithms, namely performing a line search along a given search direction, can be done in a single step, since the roots of third degree, one dimensional polynomials can be written in closed form. Using the dimension d as a parameter characterizing the problem, the difficulty of finding a global minimum must arise, at least in part, from the fact that the number of local minima can grow combinatorically[5] in d. In fact, as the ratio of the smallest binarizing coefficient to the largest sum coefficient increases, the problem approaches a combinatoric problem, since then the binarizing terms will force all variables to take values closer and closer to 0 or 1. The following theorem explores this connection.

Theorem 14: Using $\log(\log(m))$ as an estimate for the number of distinct prime factors of the positive integer m [1], if the α_i statisfy $\alpha_i < 1\ \forall i \in \mathcal{S}$ and $\alpha_i > 16n\ \forall i \in \mathcal{B},\ n = 1, 2, \cdots, N_T - 1$, then the number of distinct minima in the factoring potential for an integer T is bounded below by $((\lfloor \log_2 T \rfloor \log 2)^{\log 2} - 2) \sum_{i=1}^{n} \binom{N_T - 1}{i}$.

Proof: Let T_i be the positive integer generated by negating[6] the i'th most significant bit of $B(T)$. Let k index the set of pairwise factors T_i, where all factors are assumed to be greater than one. Denote the ordered bits corresponding to a given factoring, $k = k_1$, along with the appropriate carry bits generated by the multiplication, by $w_{k_1}(T_i)$. Then $w_{k_1}(T_i)$ satisfies every factoring equation for T except for the one equation corresponding to the i'th bit of $B(T)$; furthermore every binarizing term vanishes at $w_{k_1}(T_i)$. The value of the non-vanishing term is just its coefficient α_i, since the squared term that α_i multiplies becomes unity. Suppose now that $w_{k_1}(T_i)$ is used as a starting point for solving the factoring equations for T, using any monotonic descent algorithm $\mathcal{A}$. Let $n = 1$, so that $\alpha_i < 1\ \forall i \in \mathcal{S}$ and $\alpha_i > 16\ \forall i \in \mathcal{B}$. Then $\mathcal{A}$ will either leave F equal to unity (if $w_{k_1}(T_i)$ happens to be a local minimum) or will lower its value. Suppose further that the same is done using $w_{k_2}(T_i)$ as a starting point, $k_1 \neq k_2$. Again F will be less than or equal to one at the resulting minimum. Label the two minima thus found $\mathbf{x}_{k_1}$ and $\mathbf{x}_{k_2}$ respectively. Now along any path joining $\mathbf{x}_{k_1}$ and $\mathbf{x}_{k_2}$, there must exist a point at which the value of F exceeds one, since at least one binarizing term must pass through its maximum as its argument passes from $w = b$ to $w = \bar{b}$, $b \in \{0, 1\}$, and since at the maximum ($w = \frac{1}{2}$), the binarizing term $\alpha_q w_q^2 (1 - w_q)^2 > 1$. Hence $\mathbf{x}_{k_1}$ and $\mathbf{x}_{k_2}$ cannot be the same point and the two minima are distinct. Thus for this choice of the α's, this method generates as many distinct local minima as there are different pairwise factors of T_i, and this is true for all $i = 1, \cdots, N_T$. The same argument extends to general n. Let $T_{i_1, \cdots, i_n}$ be the positive integer generated by negating bits $i_1, \cdots, i_n$ of $B(T)$. Then for a given

[5] An algorithm that solves the problem in time exponential in d does so in time polynomial in T, but this is still impractical for solving large problems.

[6] In the Boolean sense.

factoring of $T_{i_1,\cdots,i_n}$, all but n terms in $F(T)$ vanish, and those terms take the values $\alpha_{i_1},\cdots,\alpha_{i_n}$. Thus, again using the bit values of a given pair of factors of $T_{i_1,\cdots,i_n}$ as a starting point for $\mathcal{A}$, $F(T)$ at the resulting minimum is bounded above by n, and since again at least one binarizing term must pass through its maximum along any path joining any two minima found in this way, and since at that point $F > n$, those minima are again distinct. To obtain the bound, an estimate of the number of distinct pairwise factors of the $T_{i_1\cdots i_n}$ integers is needed. Consider only the subset of such numbers whose $B(T)$'th bit is equal to one. Since all such numbers are greater than or equal to $2^{\lfloor \log_2 T \rfloor}$, then using $\log(\log(m))$ as an estimate for the number of distinct prime factors of the positive integer m [1], a lower bound on the number of distinct prime factors of $T_{i_1\cdots i_n}$ is given by $\log(\lfloor \log_2 T \rfloor \log 2)$. Now if an integer has p distinct prime factors, then it has $2^p - 2$ ordered distinct pairwise factors (not including 1) if all its prime factors are distinct, and more than this otherwise. Then using the Hardy-Ramanujan estimate, the number of distinct pairwise factors of a given $T_{i_1\cdots i_n}$ is bounded below by $(\lfloor \log_2 T \rfloor \log 2)^{\log 2} - 2$. Since the most significant bit is fixed at 1, the number of distinct minima for a given n is then bounded below by $((\lfloor \log_2 T \rfloor \log 2)^{\log 2} - 2) \sum_{i=1}^{n} \binom{N_T - 1}{i}$.
□

Remark 15: The factors of $T_{i_1\cdots i_n}$ can have very different binary expansions from those of the factors of T, even though all but n corresponding terms in the factoring potential for both sets of binary variables are equal.

Remark 16: The above enumeration of distinct minima may not be exhaustive, beyond the simplifications made in the proof. Local minima that cannot be reached using a monotonic descent algorithm from a starting point on a cube vertex are not counted.

Remark 17: The point of the above theorem is to show that for some choices of values for the free variables, the number of solutions is combinatoric in N_T. However it does not address the situation where all free variables are of order unity.

3.2 The Structure of the Factoring Equations

An alternative view of how the difficulty of factoring is encapsulated in the above decomposition can be seen directly in the factoring equations themselves, where in this section, all variables are again assumed to be binary. The equations are 'maximally mixed', in that every possible product of an x variable with a y variable occurs, but each product occurs only once. Thus, even if a given product can be solved for, progress cannot be made by simply substituting the value for that product elsewhere. The factoring equations have a trapezoidal shape (cf. Eqs. (1)-(7)), which combines with maximal mixing to raise the difficulty of solution. For example, suppose backtracking is used, that is, find any solution to the first equation; using this solution, find any solution to the second; continue until an inconsistency is discovered; backtrack as

few equations as possible, choose a different solution, and then proceed as before until a new inconsistency is found; and so on. As each subsequent equation is visited, the number of possible solutions to that equation alone will increase until approximately half of the equations have been visited; and since each subsequent equation contains products that are not contained in the previous equations, typically a large fraction of the equations will have to be visited until an inconsistency is reached, and the backtracking will be combinatorically hard.

4 Algorithms

This section describes two basic ideas for algorithms to find the global minimum. The algorithms were tested on the simple factoring problem described in Section 2. Suppose that a local minimum has been found at $\mathbf{w}_{\min}$. The idea of the first algorithm is to adjust the free variables so that the objective function at $\mathbf{w}_{\min}$ is unchanged, but so that the L_2 norm of the gradient at $\mathbf{w}_{\min}$ is maximized. Since the value of the objective function F at the global minima is zero, and since $F > 0$ everywhere else, as long as such steps are possible, each such step is guaranteed to reduce the gap between the current and optimal values of the objective function. The idea of the second algorithm is instead to adjust the free variables so that the objective function at $\mathbf{w}_{\min}$ is unchanged, and so that the gradient still vanishes there, but so that the size of the curvature is minimized, in order that the local minimum is changed into either a saddle point or a local maximum. In both algorithms, the next local minimum is found using conjugate gradient descent with the one-step cubic line search described above. Again, as long as steps are possible, this algorithm also has the property of monotonically decreasing F.

4.1 Gradient Shifting

Suppose that a local minimum has been found. In general, although the overall gradient vanishes, some terms appearing in F will have nonvanishing gradients. In fact if all gradients individually vanish, then since the gradient of a given binarizing term $x^2(1-x)^2$ vanishes only at $x = 0, \frac{1}{2}, 1$, then a local minimum has been found with variables taking values in $\{0, \frac{1}{2}, 1\}$, and since there is no *a priori* reason to expect minima other than global minima to occur at these values, in most cases the individual gradients are not expected to all vanish. If this is the case, the coefficients α_i can be changed so that $F(\mathbf{w}_{\min})$ remains the same, but so that the overall gradient at $\mathbf{w}_{\min}$ no longer vanishes. There are several ways in which this can be implemented. The size of the resulting overall gradient could be maximized, subject to the constraints $L \leq \alpha_i \leq H \ \forall i$, where $L < 1 < H$ are positive constants. However this results in a non-convex optimization problem. A simpler method is to use the same bound constraints, but to find the terms with largest and second largest gradients - call the corresponding free variables α_{max1} and α_{max2} respectively. Then decrease α_{max1} and

increase α_{max2} until a bound constraint is met. Roughly speaking, the idea is to try to move the α's away from the lower bound as much as possible, while maintaining the constraints. The algorithm is given below. In this algorithm, all changes of the α variables (called 'moves' below) are performed in such a way that the objective function is kept constant (except for the move in step (4)), and so that the size of the resulting gradient is maximized.

Algorithm Gradient Shifting:

1. Initialize: set all $\alpha_i = 1$.
2. Find a local minimum, using conjugate gradient descent with exact cubic line search.
3. Loop through all terms. If any is zero, set its coefficient to H.
4. If $\alpha_{max1} = H$ and $\alpha_{max2} = H$, then set $\alpha_{max1} = \alpha_{max2} = 1$. (Note that this reduces the objective function.)
5. Else, if one of the α's equals L and the other equals H, the move direction is fixed: move either the lower to H or the higher to L, whichever move maintains all constraints (F constant and $L \leq \alpha_i \leq H$).
6. Else, if at least one of the α's is not at bound, choose the move that leaves $\alpha_{max1} + \alpha_{max2}$ as large as possible while maintaining the constraints.
7. Else, $\alpha_{max1} = L$ and $\alpha_{max2} = L$. Replace the second index with that corresponding to the term with largest gradient and for which the α is not equal to L. Repeat steps 3 through 6.
8. If $\alpha_i = L$ for all i for which the gradient is nonzero, then no further moves (that do not increase F but that do change the gradient) are possible, and if $F \neq 0$, then the algorithm has failed; exit.
9. Repeat from step 2, using the current point as starting point. If $F = 0$ at the resulting local minimum, then the global minimum has been found; exit.

This algorithm was implemented, using the potential in Eqs. (8) through (12). It was found to suffer from what will be called an 'Escher trap'[7]. Despite the attempt to keep the coefficients as large as possible, the algorithm described above, and several variants of it, always arrived at $\alpha_i = L \ \forall i$. Further examination showed that the $\mathbf{w}$ point arrived at was the same as the start point, despite the guaranteed monotonic descent in F. An intuitive explanation of what is happening is that at each step in which the α's are changed, the objective function itself has changed, but the minimum

[7]After a painting by M.C. Escher, which shows an endless, descending (or ascending), closed staircase.

has just moved a little, and the next round of gradient descent finds the new location of the "same" minimum; eventually all the α's become equal to L and the minimum has moved directly underneath the starting point (i.e. $\mathbf{w}$ is the same but F has been reduced by a factor of $\frac{1}{L}$). The search point has always moved downhill but has ended up where it started. In the next Section, an algorithm that changes the geometry of the energy landscape more drastically is considered.

4.2 Curvature Inversion Descent

The curvature inversion descent (CID) algorithm finds a local minimum as above, but then adjusts the α_i so that the objective function *and the gradient* are kept constant, but so that the curvature[8], projected along a suitable line, is minimized. If this minimization succeeds in making the projected curvature at $\mathbf{w}_{\min}$ negative, then restarting the gradient descent minimization on either side of $\mathbf{w}_{\min}$ along that line is guaranteed to find at least one minimum $\mathbf{w}'_{\min}$ such that $F_\alpha(\mathbf{w}'_{\min}) < F_\alpha(\mathbf{w}_{\min})$, where the dependence of F on the free variables is denoted by a subscript. One way to implement this is as follows:

Algorithm Curvature Inversion Descent:

1. Initialize: set all $\alpha_i = 1$; choose any start point.
2. Find $\mathbf{w}_{\min}$ using conjugate gradient descent with exact cubic line search.
3. Find that eigenvector $\mathbf{e}$ of the Hessian at $\mathbf{w}_{\min}$ that has the smallest eigenvalue.
4. Compute the one dimensional function $f_i(\mathbf{w}_{\min} + \mu\mathbf{e})$ for each term f_i in F. Compute the second derivative of each f_i with respect to μ. Denote the resulting vector of curvatures at $\mathbf{w}_{\min}$ (with dimension equal to the number of terms in F) by $\mathbf{c}$.
5. Solve the linear programming problem: find those α_i which leave the objective function $F(\mathbf{w}_{\min})$ constant, which also leave the gradient of F at $\mathbf{w}_{\min}$ zero, and which minimize $\mathbf{c} \cdot \alpha$.
6. Repeat from step 3 until the projected curvature of F along $\mathbf{e}$ (i.e. $\mathbf{c} \cdot \alpha$) converges.
7. If the projected curvature is negative, restart from step 2, using as start points the points $\mathbf{w}_{\min} + \epsilon\mathbf{e}$, $\mathbf{w}_{\min} - \epsilon\mathbf{e}$, for some $0 < \epsilon \ll 1$. This must result in one or two new minima. If this results in two minima, choose that minimum for which the lowest projected curvature (using steps 3 through 6) can be found.

[8]Here 'curvature projected along a line' is used to mean just the second derivatives of F along that line.

8. The algorithm stops when either the minimum is a global minimum, or is a minimum such that the projected curvature cannot be made negative by the above procedure.

The following Lemma supports this algorithm:

Lemma 18: Suppose a function $f : \mathcal{R}^d \to \mathcal{R}$ has Hessian $H(\mathbf{x})$ at point $\mathbf{x}$. Consider the set of lines through $\mathbf{x}$ and consider the set of one dimensional functions induced along each line by f. Then that line for which the induced function has minimum second derivative at $\mathbf{x}$ is $\mathbf{x} + \lambda\mathbf{e}$, where $\mathbf{e}$ is that eigenvector of $H(\mathbf{x})$ with smallest eigenvalue.

Proof: Let $\mathbf{n}$ denote a unit vector in $\mathcal{R}^d$. The one dimensional function induced by f along $\mathbf{n}$ is $g_{\mathbf{x},\mathbf{n}}(\mu) \equiv f(\mathbf{x}+\mu\mathbf{n})$. Taking second derivatives gives $\frac{\partial^2}{\partial\mu^2} g_{\mathbf{x},\mathbf{n}}(\mu) = \mathbf{n}'H(\mathbf{x})\mathbf{n}$. Now expanding $\mathbf{n}$ in terms of the eigenvectors $\mathbf{e}_i,\ i = 1, \cdots, d$ of H so that $\mathbf{n} = \sum_i (\mathbf{n}\cdot\mathbf{e}_i)\mathbf{e}_i$ gives $\mathbf{n}'H(\mathbf{x})\mathbf{n} = \sum_{i=1}^d \lambda_i (\mathbf{n}\cdot\mathbf{e}_i)^2 \equiv \sum_{i=1}^d \lambda_i a_i$, where $\sum_i a_i = 1,\ a_i \geq 0$. This is therefore a convex sum of the λ's, and is minimized by choosing $a_i = \delta_{ik}$, where k indexes the smallest eigenvalue of H.
□

Note that the constraints amount to $d+1$ linear constraints, but that there are $d+n_e$ free variables. A necessary condition for a successful move is that the projected curvature for at least one term (sum or binarizing) must be negative. There are many possible variations of this algorithm; however any algorithm for which the slope at the local minimum becomes nonzero may also suffer from the Escher trap.

5 Illustration of CID

CID was tested on the 8-dimensional problem described in Section 2, using as factoring potential the expression given in Eq. (13). The α_i were initialized to one, and $L = 0.1$, $H = 10$ were used. Using conjugate gradient descent with cubic line search, and starting from the 256 vertices of the unit cube, gave 5 different minima, of which one was the global minimum; ordering the variables as $x_0, x_1, \cdots, y_0, y_1, \cdots, z_0, z_1, \cdots$, the minima were found to lie at the points shown in Table 2.

The central row of Table 2 are the coordinates of the global minimum. The higher multiplicity suggests that, roughly speaking, the global minimum is wider than the other minima for this problem. Using CID, and again starting from the 256 vertices, the multiplicities become $[225, 31]$, with the first number corresponding to the global minimum. This example illustrates that CID can be used to escape local minima, but that it is not guaranteed to do so.

Multiplicities	Coordinates of Minima Locations							
58	0.91	-0.02	1.00	-0.04	0.43	0.66	0.25	0.08
31	0.68	-0.08	1.00	1.00	0.87	0.74	0.13	0.35
80	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
39	0.12	1.07	1.04	1.07	0.61	0.94	0.61	0.92
48	0.27	1.09	1.10	0.01	0.13	0.82	0.05	0.65

Table 2: Coordinates of local minima found with gradient descent. The number of times each minimum was encountered is given in the left column.

6 Discussion

Factoring potentials as test beds for unconstrained optimization algorithms have several appealing properties. They are low degree polynomials with coefficients of order unity, so the difficulty of finding the correct solution is a direct result of the number of local minima that occur, rather than a result of, for example, inherent precision limitations in the setup of the problem. Problems can be generated by choosing a biprime of a given size and computing the corresponding factoring potential, and for a very large number of biprimes, the correct solution is known beforehand, in contrast to other problems such as TSP, where the problem construction itself does not furnish the solution. The use of the natural numbers as a starting point gives a reasonable one-dimensional ordering of the difficulty of the problem; for example, results for different algorithms could be presented on plots of {computation time required} versus {size of biprime}. Finally, algorithms that solve the problem better than the best current algorithms [5] would be of significant practical interest.

Factoring equations with fewer variables could be derived by starting with a base $b > 2$. However, the binarizing terms would then have to be replaced with higher degree polynomials to ensure that at the solutions, the variables take values in $\{0, \cdots, b-1\}$. Progress can also be made by applying algebraic identities over the Booleans to eliminate some variables; details can be found in the Appendix.

The free variables present in this kind of optimization problem present a useful starting point for other algorithms, too. Suppose that values of the free parameters could be chosen so that the Hessian is positive semidefinite everywhere. Then the problem would be solved, since it has been made convex, and any descent algorithm will find a global minimum. This idea could be approximated by sampling the Hessian at several points, and then finding the values of the free parameters that makes the Hessians at those points as close to positive semidefinite as possible (in the Frobenius norm, for example). In this way the problem is made locally convex, and if the global minimum occurs in that set of points over which the function has been made convex, it must occur as the minimum of the function over that set of points.

The CID algorithm is interesting because it can escape local minima deterministically.

It is not restricted to the problem of solving for roots of factoring potentials; it can be applied to any unconstrained optimization problem that can be expressed as the simultaneous solution of a set of subproblems, in such a way that a potential with sufficiently many free variables can be derived. Although in this paper CID was applied to an 8 dimensional problem, there are no obvious barriers to applying it to higher dimensional problems, in contrast with algebraic methods such as homotopy and Gröbner bases methods, which are believed to be infeasible for quartics for $d \geq 8$ [4]. For the factoring problem a common drawback of gradient descent methods, the absence of a method to verify optimality [4], is not an issue. In future work CID should be compared against semidefinite programming methods (these are also not guaranteed to find the global minimum; see [4] and references therein). Finally, variations of CID such as ones that relax the constraints on the gradients may also prove fruitful.

7 Acknowledgements

Thanks to P. Simard for pointing out the SAT connection, and to D. Achlioptas for alerting me to reference [4]. Thanks to O. Downs for pointing out that TSP can also be cast as the minimization of a fourth degree polynomial.

8 Appendix: Algebraic Variable Elimination

For a given set of factoring equations, the search will be made easier if any variables can be eliminated. This appendix collects a few notes on this topic. Here all variables take values in $\{0, 1\}$. The appearance of binary variables in equations over the integers requires interestingly different techniques for procedures like variable elimination. Below, $\bar{}$ denotes 'not', $|$ denotes 'or', products denote 'and', $\otimes$ denotes the XOR operation, and $|_2$ at the end of an expression denotes modulo 2.

8.1 Elimination of Variables

Consider the following equation in n variables:

$$A_1 + A_2 + A_3 + \cdots + A_n = 1 \tag{14}$$

Simple elimination, for example, replacing A_1 everywhere by $1 - A_2 \cdots - A_n$, will not work, since doing so would allow the eliminated variable to take values outside the set $\{0, 1\}$. One method to eliminate one of the variables, but to ensure that all variables

remain in $\{0,1\}$, is to introduce auxiliary binary variables $x_1, \cdots, x_{n-1}$, defined by

$$\begin{aligned} A_1 &= 1 - x_1 \\ A_2 &= x_1(1 - x_2) \\ A_3 &= x_1 x_2 (1 - x_3) \\ &\cdots \\ A_n &= x_1 x_2 \cdots x_{n-1} \end{aligned}$$

The key property of the x's is that they are unconstrained. However, the A's have been replaced by products of x's, which brings its own complexity. This can also be accomplished if there are 'carry' variables on the right hand side: for example,

$$A + B + C = 1 + 2z \tag{15}$$

can be reduced from four to three variables with

$$\begin{aligned} A &= z + (1 - z)(1 - x_1) \\ B &= z + (1 - z)x_1(1 - x_2) \\ C &= z + (1 - z)x_1 x_2 \end{aligned}$$

However the equations for more carry variables quickly become complicated, since every combination of the z's must be accounted for. A constant on the right hand side, for example

$$A_1 + A_2 + A_3 = 2 \tag{16}$$

can be solved by setting

$$\begin{aligned} A_1 &= x_1 \\ A_2 &= \bar{x}_1 | x_2 \\ A_3 &= \bar{x}_1 | \bar{x}_2 \end{aligned}$$

Similarly

$$A_1 + A_2 + A_3 + A_4 = 2 \tag{17}$$

is solved by

$$\begin{aligned} A_1 &= x_1 \\ A_2 &= x_2 \\ A_3 &= \overline{x_1 x_2}(x_3 | \bar{x}_1 \bar{x}_2) \\ A_4 &= \overline{x_1 x_2}\ \overline{x_2 x_3}\ \overline{x_3 x_1} \end{aligned}$$

Again, things get complicated quickly.

8.2 Useful Identities

Some identities that are useful in performing algebraic operations on Boolean variables are collected here. They can be used to modestly reduce the number of variables appearing in a factoring potential.

$$\begin{aligned}
\overline{(A|B|\cdots|C)} &= \bar{A}\bar{B}\cdots\bar{C} \\
\overline{(AB\cdots C)} &= \bar{A}|\bar{B}|\cdots|\bar{C} \\
(A \otimes B) \otimes C &= A \otimes (B \otimes C) \equiv A \otimes B \otimes C \\
(A \otimes B \otimes \cdots \otimes C) &= (A + B + \cdots C)|_2 \\
A \otimes B &= A|B - AB \\
A \otimes B &= (A|B)\overline{AB} = (A|B)(\bar{A}|\bar{B}) \\
A|B &= A + B - AB \\
A + B &= z_1 + 2z_2 \;\Rightarrow\; z_1 = A \otimes B,\; z_2 = AB \\
A + B &= 2z \;\Rightarrow\; A = B = z \\
CA &= \bar{C}B = 0 \;\Rightarrow\; AB = 0 \\
A(B|C) &= (AB|AC)
\end{aligned}$$

Here are some 'contraction rules':

$$\begin{aligned}
\bar{A}(A|B|C|\cdots) &= \bar{A}(B|C|\cdots) \\
A(B|A) = A & \\
A(AB|C|\cdots) &= A(B|C|\cdots) \\
\bar{A}\;\overline{AB} = \bar{A} & \\
(A|C)(B|C) &= C|AB
\end{aligned}$$

Factor equations often result in quadratic identities of the form $AB = 0$. This can be replaced by an algebraic inequality:

$$AB = 0 \;\Leftrightarrow\; A + B < 2 \tag{18}$$

In fact systems of such equations can be succinctly captured with inequalities: for example,

$$AH = BH = CH = AJ = BJ = CJ = AK = BK = CK = 0$$

$$\Leftrightarrow\; (A|B|C) + (H|J|K) < 2$$

Finally, Boolean ORs can be expressed as algebraic sums by using the following Lemma:

Lemma 19: The Boolean expression $x = (y_1|y_2|y_3|\cdots|y_n)$ is equivalent to the algebraic expression

$$x = \sum_i y_i - \sum_{i<j} y_i y_j + \sum_{i<j<k} y_i y_j y_k - \cdots + (-1)^{n+1} y_1 y_2 \cdots y_n \tag{19}$$

Proof: Suppose m of the y_i are equal to 1, the rest 0, and suppose $m > 1$. Then $\sum y_i = m = \binom{m}{1}$, $\sum_{i<j} y_i y_j = \binom{m}{2}$, $\sum_{i<j<k} y_i y_j y_k = \binom{m}{3}$, etc. Hence the right hand side is $\binom{m}{1} - \binom{m}{2} + \binom{m}{3} - \cdots + (-1)^{m-1}\binom{m}{m} = 1$ (to see this, expand $(1-1)^m$). The cases $m = 0$ and $m = 1$ are self evident.
□

References

[1] G.H. Hardy and S. Ramanujan. The normal number of prime factors of a number n. *Quarterly J. Mathematics*, 48:76–92, 1920.

[2] Akira Horie and Osamu Watanabe. Hard Instance Generation for SAT. In *Proceedings of the 8th International Symposium on Algorithms and Computation*, pages 22–31. Lecture Notes in Computer Science 1350, 1996.

[3] See http://www8.pair.com/mnajtiv/sat/sat.html.

[4] P. Parrilo and B. Sturmfels. Minimizing polynomial functions. Technical report, Caltech, Pasadena, California, 2001.

[5] Hans Riesel. *Prime numbers and computer methods for factorization.* Birkhäuser, 2 edition, 1994.