Journal of Computational Mathematics, Vol.24, No.3, 2006, 357–364.

DISTURBED SPARSE LINEAR EQUATIONS OVER THE 0-1 FINITE FIELD *[1)]

Ya-xiang Yuan and Zhen-zhen Zheng
(*LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100080, China*)

Dedicated to the 70th birthday of Professor Lin Qun

Abstract

In this paper, disturbed sparse linear equations over the 0-1 finite field are considered. Due to the special structure of the problem, the standard alternating coordinate method can be implemented in such a way to yield a fast and efficient algorithm. Our alternating coordinate algorithm makes use of the sparsity of the coefficient matrix and the current residuals of the equations. Some hybrid techniques such as random restarts and genetic crossovers are also applied to improve our algorithm.

Mathematics subject classification: 65L05, 65F10.
Key words: Sparse linear equation, 0-1 finite field, Alternating direction method, Random restart, Genetic hybrids.

1. Introduction

In this paper, we study the problem of solving large sparse linear equations over $GF(2)$, the field with two elements. Let $F = \{0, 1\}$, the problem can be written as

$$\begin{array}{rcl} a_{11}x_1 \oplus a_{12}x_2 \oplus \ldots \oplus a_{1n}x_n & = & b_1, \\ a_{21}x_1 \oplus a_{22}x_2 \oplus \ldots \oplus a_{2n}x_n & = & b_2, \\ \vdots & \vdots & \vdots \\ a_{m1}x_1 \oplus a_{m2}x_2 \oplus \ldots \oplus a_{mn}x_n & = & b_m, \end{array} \tag{1.1}$$

where $\oplus$ is the "exclusive or" operator over the 0-1 field, namely

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0, \tag{1.2}$$

and where $A = (a_{ij}) \in F^{m\times n}$ is a given sparse matrix, $b = (b_i) \in F^m$ is a given vector and $x = (x_i) \in F^n$ is the variable that we need to calculate, with m and n being two positive integers. Such a problem have many applications, including classification problems [14] and integer factorization problems [5, 6].

However, solving large sparse linear systems over finite fields is not easy. The most common approach to this problem is to generalize or modify the standard numerical methods for linear equations defined in $\Re^n$, such as structured Gaussian elimination, conjugate gradient, and block Lanczos algorithm (see [13, 5]). Another famous approach is due to Wiedemann, whose method uses "coordinate recurrence" and the minimum polynomial of the sub-matrices of the coefficient matrix A(see [17, 6]).

* Received March 1, 2006.
[1)] This work is partially supported by Chinese NSF grant 10231060 and the CAS Knowledge Innovation Program.

For simplicity, we denote problem (1.1) by

$$\bigoplus_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, ..., m\ . \tag{1.3}$$

It is quite usual for many application problems to have $m >> n$ in which case the equations (1.3) have no solution. Therefore, it is natural for us to consider the least squares problem:

$$\min_{x\in F^n} \sum_{i=1}^{m} \left(\bigoplus_{j=1}^{n} a_{ij}x_j - b_i \right)^2 . \tag{1.4}$$

If the operator $\bigoplus$ is replaced by $\sum$ in the above problem and if some simple linear constraints are added, (1.4) turns into the standard quadratic assignment problem, which has been widely studied (see [3, 4, 15, 1, 18]). The amazing success of semi-definite programming relaxations to quadratic assignment problems(for example, see[20, 2]) naturally suggests us to explore the possibilities of applying relaxations to problem (1.4). Indeed, we tried interior-point gradient methods with diagonal-scalings [19] and trust-region interior-point algorithms [7] to solve the relaxation problem to (1.4). But unfortunately, our numerical experiments show that it is difficult and time-consuming for interior point methods to find good solutions of this particular problem.

Therefore, we investigated other possible approaches to solve this difficult problem. Coming into our minds was one very simple technique: the alternating coordinate direction search method. The alternating direction search method is one of the direct methods for nonlinear optimization problems. It tries to find a minimum of a nonlinear function defined in the n dimensional space by searching along n coordinate directions in turns. The classical alternating directions methods include the pattern search method by Hooke and Jeeves [12] and Rosenbrock's method [16]. Considering the binary and sparse properties of the problem, we noticed that the basic idea of alternating directions could be implemented here very efficiently. At a typical step when we search along a coordinate direction j, all we need to do is just to compare the objective function values at two points, which can be easily done by counting the non-zero elements of the current residual vector from those indices i such that a_{ij} are non-zero.

The numerical results for the alternating coordinate search method are very encouraging, particularly for the zero residual problems, namely problems that have an exact solution. Actually our simple algorithm also works very well for disturbed problems when the sparsity is over 97 %. The disturbed systems are generated by randomly changing b, where b is the right hand side vector of a zero-residual problem. In order to improve our algorithm, we also consider random restarts and genetic hybrid techniques.

This paper is organized as follows. In the following section we present an alternating direction algorithm for sparse linear equations over finite field of two elements. Numerical results of our algorithm are given in Section 3. In Section 4 we discuss random and genetic hybrids improvement techniques and give some more computational results. Finally, some concluding remarks are given in Section 5.

2. The Alternating Direction Algorithm

Define the residual vector $r(x) \in F^m$ by

$$r_i(x) = |\bigoplus_{j=1}^{n} a_{ij}x_j - b_i| \quad (i = 1, ..., m). \tag{2.1}$$

Due to the fact that r_i is either 0 or 1, we can easily see that problem (1.4) is equivalent to

$$\min_{x\in F^n} \|r(x)\|_1. \tag{2.2}$$

In a typical iteration, we have a current iteration point x, and we need to search along a coordinate direction, say the $j-$th coordinate direction e_j. All we need to do is to compare the function value $\|r\|_1$ at the current point x and another point $\bar{x}$ which is defined by

$$\bar{x}_i = x_i, (i \neq j), \qquad \bar{x}_j = x_j \oplus 1. \tag{2.3}$$

Therefore, it is easy to see that $r(\bar{x}) = r(x) \oplus a_j$, where a_j is the $j-th$ column of A. For each $j = 1, 2, ...n$, we define $I_j \subset \{1, 2, ..., m\}$ to be the subset of the indices of all i such that $a_{ij} \neq 0$. Thus we only need to consider the elements of $r(x)$ for those indices belong to I_j. If we move the iterate point from x to $\bar{x}$, the residual vector $\bar{r}$ is given by changing the elements of $r(x)$ for the indices in I_j. Thus, we see that the objective function $\|r\|_1$ will be reduced if and only if the nonzero elements of $r(x)$(with the indices belonging to I_j) are more than a half. This enables us to implement a very simple alternating direction search algorithm, which is given as follows.

Algorithm 2.1. *(An alternating coordinate search algorithm)*

Step 1 Given $A \in F^{m\times n}$, $b \in F^m$. For $j = 1, 2, ...n$, let $I_j \subset \{1, 2, ..., m\}$ be the subset of all i such that $a_{ij} \neq 0$. Randomly generate an initial point $x \in F^n$. Calculate $r = r(x)$.

Step 2 $x_{best} = x$, $j := 1$

Step 3 For all $i \in I_j$ count the number of non-zero elements of $r(x)$, denoted by $COUNT$. If $COUNT > |I_j|/2$ then do $\{\ x_j = x_j \oplus 1$ and $r_i = r_i \oplus 1(i \in I_j)\ \}$.

Step 4 If $j < n$ then $j := j + 1$ and go to Step 3.

Step 5 If $x = x_{best}$ then accept x_{best} as an approximate solution, otherwise go to Step 2.

We notice that because the coefficient matrix A is very sparse and its nonzero elements have been stored in advance, checking r in the step 3 does not cost much computational time. In addition, the new residual vector r can be obtained by modifying very few components. In the algorithm, we only need to compute the whole residual vector once(when we initiate the algorithm), which saves a lot of computational time.

The stopping condition in Algorithm 2.1 does not guarantee a global minimizer of problem (2.2). In fact it will terminate at a point where no improvement on the objective function can be achieved by searching along any coordinate direction.

3. Numerical Results

The problems we tested for our algorithm and its modifications are all generated randomly. Given two positive numbers m and n, we first generate the elements of $A \in F^{m\times n}$ randomly with a certain sparsity. For example a sparsity of 10% indicates that only 10 percent of the elements of A are ones. The exact solution $x^* \in F^n$ of the problem are also generated randomly. Once we have A and x^*, the right hand side vector b is defined by $b_i = \oplus_{j=1}^n a_{ij}(x^*)_j (i = 1, ...m)$. Then, we apply Algorithm 2.1 to recover the exact solution x^*. For some problems, we disturb the right hand side vector b with different disturbance ratios. For example, a vector b disturbed with a disturbance ratio 30% means that a randomly selected 30 percent of the elements in b are changed either from 0 to 1 or from 1 to 0. In all the problems we choose $m = 400000$ and $n = 128$.

The programs were coded in Microsoft Visual C++ 6.0 and ran on a 2.4GHz desktop Pentium IV computer. The numerical results for the undisturbed problems are reported in Table 1. The degree of sparsity is chosen as 99%, 98%, 97%, 96%, 95% respectively. For each case we ran ten times and reported the average results, which are given in Table 1. The first column is the sparsity of the problem, the second column is the 1-norm of the final residual at

the computed point x_{best}, the third column is the 1-norm distance between the computed point and the exact solution, and the final column is the cpu time in seconds. From these results, we can easily conclude that the alternating coordinate search method works very efficient for undisturbed equations.

sparsity	$\|r(x_{best})\|_1$	$\|x^* - x_{best}\|_1$	T
99%	0	0	5
98%	0	0	5
97%	0	0	6
96%	0	0	6
95%	0	0	7

Table 1. *Algorithm 2.1 for undisturbed problems*

In many applications, the right hand side is disturbed. Therefore, we also tested our algorithm for the disturbed cases. The disturbance ratio, as defined above, is set to be 10%, 20%, 30%, 40%, and 45% respectively. The results for the disturbed cases are reported in Table 2. Even though Table 2 looks very similar to Table 1, there are 5 differences. The first one is that in Table 2 the problems are disturbed with difference disturbance ratios from 10% to 45%. The second difference is that in Table 2 the results for the sparsity of 98% and 97% are not listed as they are the same as those for 99%. The third difference is that in Table 1 we give the average of 10 runs, while in Table 2 we give two separate runs for each case. The forth difference is that in Table 2 we do not list the CPU time as every run (for difference case) the cpu time is about the same (about 5 to 6 seconds). The final difference is that in Table 2 we also give the degree of dominance of the final point x_{best}, which is defined by

$$dominance(x_{best}) = \frac{\|r(x_{best})\|}{m} \times 100\%, \tag{3.1}$$

where r is defined in (2.1). Because disturbed systems are often over-determined systems without zero residual solution, it is usual that the *dominance* will not be zero. For example, for the disturbed test problems that we generated, the number *dominance* at x^* is just the disturbance ratio. Thus, for our computed solution x_{best}, we should be happy with it if its *dominance* is not greater than disturbance ratio.

		RUN ONE			RUN TWO		
sparsity	disturb.	$\|r\|_1$	$\|x^* - x_{best}\|_1$	*domin.*	$\|r\|_1$	$\|x^* - x_{best}\|_1$	*domin.*
	10%	40000	0	10	40000	0	10
	20%	80000	0	20	80000	0	20
99%	30%	120000	0	30	120000	0	30
	40%	160000	0	40	160000	0	40
	45%	180000	0	45	180000	0	45
	10%	40000	0	10	40000	0	10
	20%	80000	0	20	80000	0	20
96%	30%	120000	0	30	120000	0	30
	40%	196782	51	49.20	197380	71	49.35
	45%	196875	68	49.22	197293	76	49.32
	10%	40000	0	10	40000	0	10
	20%	80000	0	20	80000	0	20
95%	30%	196505	52	49.13	197668	51	49.42
	40%	196956	56	49.24	196765	41	49.19
	45%	196979	57	49.24	197679	70	49.42

Table 2. *Algorithm 2.1 for disturbed problems*

From the results in Table 2 we notice that none of the computed points x_{best} is better than x^*(the solution of the undisturbed equations). One reasonable guess is that x^* is very likely the global minimizer of (2.2). When the sparsity is 96% and 95% and when the disturbance ratio surpasses 30%, the points computed by our algorithm is much worse than x^*. Therefore, in order to obtain an efficient and practical method for solving problem (2.2), we should modify our algorithm. In the next section we give two straightforward approaches, one is random restarts and the other is genetic crossover.

4. Random Restarts and Genetic Hybrids

Now, we combine Algorithm 2.1 with the random restart technique. Whenever Algorithm 2.1 terminates at a local minimizer, we restart the algorithm with a randomly chosen new starting point. Of course, such an approach would need a stopping technique, which is not obvious, because it is well known that termination criterion for global optimization is very hard to given. However, here we are more interested in exploring the efficiency of random restarts. Thus we terminate the algorithm whenever the degree of dominance is not higher than the disturbance ratio of the problem.

Algorithm 4.1. *(An alternating coordinate search algorithm with random restarts)*

Step 1 Given $A \in F^{m\times n}$, $b \in F^m$. Given the disturbance ratio τ. Let $v_{opt} = +\infty$.

Step 2 Restart Algorithm 2.1 to obtain a new x_{best}.
If $\|r(x_{best})\|_1 \geq v_{opt}$ go to Step 2.

Step 3 $x_{opt} := x_{best}$; $v_{opt} = \|r(x_{best})\|_1$.
If $\|r(x_{opt})\|_1 > \tau m$ go to Step 2.
Return x_{opt} as an approximate solution and Stop.

In practice, this is not feasible unless we know the disturbance ratio of the problem to be solved. One possible way is to stop the overall procedure if no better local minimizer is found after quite a few restarts (see[11]). In table 3 we present the computational results of Algorithm 4.1, where $r_{opt} = r(x_{opt})$ and $\epsilon_{opt} = \|x^* - x_{opt}\|_1$. It is easy to see that random restarts do help to improve our algorithm because the prescribed "exact solution" x^* is found by Algorithm 4.1 for all the five cases that x^* was not found by Algorithm 2.1. However, for low sparsity and high disturbance ratio problems, many restarts may be needed in order to reach the required degree of dominance. For example, in the second run for the case with sparsity 95% and disturbance ratio 45%, it takes more than one hour for Algorithm 4.1 to find an acceptable approximate solution.

			RUN	ONE			RUN	TWO	
sparsity	disturb.	$\|r_{opt}\|_1$	ϵ_{opt}	*domin.*	T	$\|r_{opt}\|_1$	ϵ_{opt}	*domin.*	T
	10%	40000	0	10	6	40000	0	10	6
	20%	80000	0	20	6	80000	0	20	5
96%	30%	120000	0	30	5	120000	0	30	6
	40%	160000	0	40	6	160000	0	40	6
	45%	180000	0	45	17	180000	0	45	7
	10%	40000	0	10	6	40000	0	10	5
	20%	80000	0	20	5	80000	0	20	6
95%	30%	120000	0	30	14	120000	0	30	7
	40%	160000	0	40	114	160000	0	40	116
	45%	180000	0	45	392	180000	0	45	64.5m

Table 3. *Algorithm 4.1 for disturbed problems*

Next, considering the discrete property of this problem, we solve it by a heuristic algorithm, the genetic hybrids algorithm. It is a steady population evolutionary process(see [10]) which was successfully applied to the quadratic assignment problem(see [9, 8]). Here we specifically apply Algorithm 2.1 as the local search because it is fast and efficient.

Algorithm 4.2. *(A genetic hybrids algorithm)*

Step 1 Given $A \in F^{m\times n}$, $b \in F^m$. Given the disturbance ratio τ. Let $v_{opt} = +\infty$.

Step 2 Generate the initial population randomly.
Carry out local search and evaluate the population.

Step 3 Do the Roulette Wheel selection operation, one-point crossover operation and local search.

Step 4 Evaluate the population, find the best and worst individuals: x_{best} and x_{worst}.
If $\|r(x_{best})\|_1 \le v_{opt}$ then $x_{opt} := x_{best}$; $v_{opt} = \|r(x_{best})\|_1$.
Else replace x_{worst} by x_{best}.

Step 5 If $\|r(x_{opt})\|_1 \le \tau m$, return x_{opt} as an approximate solution and stop.

Step 6 If the number of x_{best} is not more than 80% in the current population go to Step 3.
Do the simple bit mutation for each individual except for one x_{best}, evaluate the population and go to Step 3.

In the above algorithm, "evaluate the population" includes calculating the objective function value and the fitness of each individual. In practice, we can stop the overall procedure if no better individual is found after quite a few iterations or if the number of iterations reaches a prescribed maximum number(see [9, 8]). Some parameters are as follows: population size: 10, crossover probability: 0.8, mutation probability: 0.6. In Table 4, we present the computational results of Algorithm 4.2. It can be easily seen that the genetic hybrids algorithm managed to find the prescribed "exact solution" x^* for all cases. It is also interesting to observe that the genetic hybrids algorithm needs less computing time than the random restart method for the hard cases such as sparsity 95% with disturbance ratio 45%.

			RUN	ONE			RUN	TWO	
sparsity	disturb.	$\|r_{opt}\|_1$	ϵ_{opt}	*domin.*	T	$\|r_{opt}\|_1$	ϵ_{opt}	*domin.*	T
	10%	40000	0	10	9	40000	0	10	9
	20%	80000	0	20	10	80000	0	20	9
96%	30%	120000	0	30	10	120000	0	30	10
	40%	160000	0	40	12	160000	0	40	12
	45%	180000	0	45	41	180000	0	45	16
	10%	40000	0	10	11	40000	0	10	11
	20%	80000	0	20	11	80000	0	20	11
95%	30%	120000	0	30	11	120000	0	30	11
	40%	160000	0	40	12	160000	0	40	40
	45%	180000	0	45	29	180000	0	45	45

Table 4. *Algorithm 4.2 for disturbed problems*

However, if we lower the sparsity of the problem further, our generic hybrids algorithm also takes much time to find x^*. Some further results are given as follows in Table 5. This indicates that when sparsity is lower than 94% the problem is really very hard to solve.

sparsity	disturb.	$\|r_{opt}\|_1$	ϵ_{opt}	*domin.*	T
	10%	40000	0	10	17
	20%	80000	0	20	32
94%	30%	120000	0	30	44
	40%	160000	0	40	262
	10%	40000	0	10	113
93%	20%	80000	0	20	17.36m
	30%	120000	0	30	43.62m
92%	10%	40000	0	10	471.48m
	20%	80000	0	20	123.73m

Table 5. *Algorithm 4.2 for low sparsity disturbed problems*

5. Discussions

In this paper we presented an alternating directions method for solving sparse 0-1 linear equations over the 0-1 finite field. Numerical tests on randomly generated problems indicate that our algorithm works very well for undisturbed systems. For disturbed systems, we proposed two ways of modifications. One approach is repeatedly restarting the algorithm randomly, and the other is applying genetic hybrids techniques. Both modifications were tested and we found they do provide improvement over the original algorithm. However, for low sparsity problems with high disturbance ratios, even our modified algorithms are not fast enough. Therefore it worths to investigate more efficient algorithms.

Another point should be addressed is that our unsuccessful attempts on relaxation does not at all rule out the possibility of the existence of fast and efficient relaxation method for sparse linear equations over the 0-1 field. The crucial part of a relaxation method is how to extend the objective function $\|r(x)\|_1$ (or $\|r(x)\|_2$) from F^n to the $n-$dimensional box $B = \{0 \le x_i \le 1, i = 1, ..., n\}$. What we have tried are the two straightforward ways. One is to replace $x \oplus y$ by $(x + y)/2 - xy$. Applying this recursively, we can change (1.3) into a system of polynomial equations. Another is to consider the relaxed problem:

$$\min_{x \in B} \sum_{i=1}^{m} \left(\cos(\pi[\sum_{j=1}^{n} A_{ij}x_j - b_i]) - 1 \right)^2, \tag{5.1}$$

or

$$\min_{x \in B} \sum_{i=1}^{m} |\cos(\pi[\sum_{j=1}^{n} A_{ij}x_j - b_i]) - 1|. \tag{5.2}$$

Neither minimizing cosine functions nor solving high order polynomial equations is an easy task. This can partially explain why our attempts in using relaxations did not work out well. We believe that a clever definition of $r(x)$ in on the box $B = \{0 \le x_i \le 1, i = 1, .., n\}$ would lead to an efficient relaxation method.

References

[1] K.M. Anstreicher, Recent advances in the solution of quadratic assignment Problems. *Mathematical Programming, Ser.B,* **97** (2003), 24-42.

[2] K. Anstreicher, X. Chen, H. Wolkowicz and Y. Yuan, "Strong duality for a trust-region type relaxation of the quadratic assignment problem", *Linear Algebra and its Appl.,* **301** (1999), 121-136.

[3] R.S. Burkard, and T. Bönniger, A heuristic for quadratic Boolean programs with applications to quadratic assignment problems. *European Journal of Operational Research*, **13** (1983), 374-386.

[4] E. Çela, The Quadratic Assignment Problem: Theory and Algorithms, (Kluwer, New York, 1998).

[5] D. Coppersmith, Solving linear equation over GF(2): block Lanczos algorithm, *Linear Algebra and Its Applications*, **192** (1993) 33-60.

[6] D. Coppersmith, Solving linear equations over GF(2) via block Wiedemann algorithm, *Math. Comp.*, **62** (1994), 333-350.

[7] J.E. Dennis, L.N. Vicente, Trust-region interior-point algorithms for minimization problems with simple bounds, Applied Mathematics and Parallel Computin, Springer, New York, pp.97-107, 1996.

[8] Z. Drezner, Compounded genetic algorithms for the quadratic assignment problem, *Operations Research Letters,* **33** (2005), 475-480.

[9] CH. Fleurent and J. A. Ferland, Genetic hybrids for the quadratic assignment problem, *DIMACS, Series in Mathematics and Theoretical Computer Science,* **16** (1994), 190-206.

[10] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, (Addison-Wesley, Reading, 1989.)

[11] F. Hickernell and Y. Yuan, A simple multi-start algorithm for global optimization, *Operational Research Transaction (China),* **1**:2(1997), 1-12.

[12] R. Hooke and T.A. Jeeves, Direct search solution of numerical and statistical problems, *J. ACM,* **8** (1961), 212-229.

[13] B.A. LaMacchia, A.M. Odlyzko, Solving large sparse linear systems over finite fields, Lecture Notes in Computer Science, **537** (1991), 109-133.

[14] V.F. Kolchin, A classification problem in the presence of measurement errors, Discrete Mathematics and Applications, **4** (1994), 19-30.

[15] P.M. Pardalos, F. Rendl and H. Wolkowicz, The quadratic assignment problem: a survey and recent developments. In: P.M. Pardalos and H. Wolkowicz, (eds.), *Quadratic Assignment and Related Problems*, Volume**16** of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS. pp. 1-42.

[16] H.H. Rosenbrock, An automatic method for finding the greatest or least value of a function, *Computer J.,* **3** (1960), 175-184.

[17] D.H. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Transcations on Information Theory*, **32** (1986) 54-62.

[18] Y. Xia and Y. Yuan, A new linearization method for quadratic assignment problems, Technical Report, AMSS, CAS, 2004, to appear in *Optimization Methods and Software*.

[19] Y. Zhang, Interior-Point Gradient Methods with Diagonal-Scalings for Simple-Bound Constrained Optimization, CAAM Technical Report TR04-06, Rice University, 2004.

[20] Q. Zhao, S. Karisch, F. Rendl and H. Wolkowicz, Semidefinite programming relaxations for the quadratic assignment problem. *J. Combinatorial Optimization,* **2** (1998), 71-109.