ATLANTIS PRESS

Advances in Intelligent Systems Research, volume 153

International Conference on Applied Mathematics, Modeling and Simulation (AMMS 2017)

Two Schemes to Improve the Implementation of the Aggregation Based Algebraic Multigrid Preconditioner

Jianping Wu*, Fukang Yin, Jun Peng and Jinhui Yang

College of Meteorology and Oceanography, National University of Defense Technology, Changsha, China

*Corresponding author

***Abstract*—Algebraic multigrid is often used as the preconditioner in Krylov subspace iterations to solve general sparse linear systems, and the aggregation based version is one of the most popular, for its cheap complexity to setup. In this paper, when this version used as the preconditioner, two schemes are provided to improve its implementation. The first is to remove the trivial computation in the smoothing for zero initial vectors before the restriction process. For this case, part of the computation is related to the multiplication of an operator to a zero vector, and then it can be omitted without any changes to the derived result. The second is to reconstruct the restriction process, which can reduce the computation complexity at the cost of a little more storage. The analyses and the numerical experiments for the solution of sparse linear systems from a model partial differential equation with preconditioned conjugate gradients show that the provided schemes can reduce the solution time significantly. The improvements are much more significant when more nodes are aggregated each time, and are more significant to W-cycle than to V-cycle and K-cycle.**

Keywords-aggregation based algebraic multigrid; sparse linear system; preconditioner; conjugate gradient method; computation complexity

I. INTRODUCTION

The solution of sparse linear systems is the kernel of many scientific and engineering computations and often takes a long time. Thus it attracts many attentions and up to now, a lot of methods have been developed. Among them, the so-called Krylov subspace iterations [1] and the multigrid methods [2] are two of the most efficient ones. The Krylov subspace methods are to seek an approximate solution in the Krylov subspace based on project schemes, which is very efficient due to some minimization properties. But the convergence rate is dependent on the distribution of the eigenvalues of the coefficient matrix. The narrower area they are distributed in, the faster the convergence rate will be. To improve the distribution, preconditioning techniques are often used, where a non-singular operator called the preconditioner is applied to the linear system to derive another with the same solution but with better eigenvalues distribution, and then the problem is converted to the solution of the newly derived system [1].

Multigrid is another of the most efficient methods to solve sparse linear systems, which is based on the complementation of two processes, smoothing and correction [2]. The residual vector related to the smoothing on the finer grid is restricted to a coarser level and the solution on this level is prolonged backed to correct the approximate solution on the finer level. For the smoothing can reduce the error components with relatively higher frequencies efficiently, and error components with lower frequencies can be more efficiently processed on coarser grids, the whole convergence rate of the multigrid can be guaranteed only if the coefficient matrix satisfies some property. In fact, many analyses and experiments show that for linear systems with good properties, the multigrid methods have the potential optimal convergence. But its robustness is not very good when used alone. Therefore, it is often used as preconditioners of the Krylov subspace methods.

Algebraic multigrid methods are often used for general sparse linear systems [3], where the construction of coarser grids, the linear systems on the coarser levels, the restriction and the prolong operators are all based on the coefficient matrix only. Among them, the aggregation based version is one of the most popular. In the classical implementation, the prolong operator depends solely on the selection of the coarser grid, leading to the simplicity for the construction and cheap complexity to the setup process [3]. Though the aggregation based multigrid methods are very popular, the focus is mostly on the selection of the coarser grids [4-8] and the cycle type of multigrid [5]. In reference [6], the prolong operator is focused on and it is smoothed to accelerate the convergence rate of the derived multigrid version. But up to now, little focus is on the efficient implementation. In this paper, when it is used as the preconditioner to the conjugate gradient iterations, we focus on the improvements to the implementation.

In section II, the conjugate gradient with aggregation based multigrid as the preconditioner will be given, which is the basis for further description and analyses. The considered multigrid methods include the V-, W- and the K-cycles. The schemes to improve the implementation will be given in section III and some analyses and discussions will be provided here too. In section IV, some numerical experiments will be given to verify the effectiveness of the schemes and some conclusions will be drawn in section V.

II. CONJUGATE GRADIENT WITH AGGREGATION BASED ALGEBRAIC MULTIGRID AS THE PRECONDITIONER

Without loss of generality, consider the following linear system of order n

$$Ax=b, \tag{1}$$

where A is a given symmetric positive definite matrix, b is a known vector, and x is the unknown solution. We can use the preconditioned conjugate gradient (PCG) iteration to solve (1), where it is converted to

$$MAx=Mb, \tag{2}$$

and the inner product (., .) in the conjugate gradient iteration is replaced by $(., .)_M$ $^{-1}$, defined by $(x, y)_M$ $^{-1}$= $(x, M^{-1}y)$. Therefore, the preconditioner M should be symmetric positive definite too. The PCG algorithm [1] can be described as in figure I, where maxIts is the maximum number of iterations allowed and ε is a threshold to control the process.

1. Set $r^{(1)} = b - Ax^{(1)}$, $z^{(1)} = Mr^{(1)}$ and $s^{(1)} = z^{(1)}$.
2. For j = 1, maxIts
3. Compute $\alpha_j = (z^{(j)}, r^{(j)}) / (As^{(j)}, s^{(j)})$.
4. Compute $x^{(j+1)} = x^{(j)} + \alpha_j s^{(j)}$.
5. Compute $r^{(j+1)} = r^{(j)} - \alpha_j As^{(j)}$, if $\| r^{(j+1)} \|_2 / \|b\|_2 < \varepsilon$, then stop.
6. Compute $z^{(j+1)} = Mr^{(j+1)}$.
7. Compute $\beta_{j+1} = (z^{(j+1)}, r^{(j+1)}) / (z^{(j)}, r^{(j)})$.
8. Compute $s^{(j+1)} = z^{(j+1)} + \beta_{j+1} s^{(j)}$.
9. Endfor

FIGURE I. ALGORITHM PCG(PRECONDITIONED CG)

In this section, the aggregation based multigrid method [3] is used as the preconditioner M, which can be described as in figure II. For more details about the V-cycle, W-cycle and K-cycle, we can refer to [3][5]. In figure II, m denotes the maximal number of levels allowed, l is the current level number. On level l, the solution is smoothed with S_l, and the corresponding residual vector $r^{(l)} = b^{(l)} - A_l x^{(l)}$ is transferred to the coarser level l+1 through multiplication by $(P_{l+1,l})^T$ from the left. On the (l+1)-th grid, algorithm MGprec is invoked again to derive the correction vector $y^{(l+1)}$, which includes step 3 to 23 in figure II. It is transferred back to the l-th grid, to correct the approximation $x^{(l)}$. And then a post smoothing is used to $x^{(l)}$ with the smoother S_l^T.

When computing the correction, there are four branches. If the number of levels reaches m, or the linear system is small enough, the correction is on the coarsest level and can be solved with a certain method, either with a direct solution, or with an iterative process. Otherwise, the correction is computed with the algorithm MGprec recursively, including three options V-, W-, and K-cycle. In algorithm MGprec, the grid hierarchy, A_l, $P_{l,l+1}$, and S_l are unchanged with the conjugate gradient iteration. They can be re-computed and put into a set-up process beforehand, and be used directly in latter iterations.

For aggregation based algebraic multigrid, there are several schemes to determine the grid hierarchy. One of them is provided by Kim et al., which aggregates two points each time at most [4]. To reduce the number of levels, it can be repeated two or several times each time [5]. Others include the scheme based on strong coupling [6], the algorithm based on graph partitioning for two-grid method [7], the scheme according to seven known geometry structures [8], and the scheme based on subsets grouping [8]. The researches in reference [9] show that, among these schemes, the two-point scheme and its variants with two or three recursions are the most efficient in most cases. Therefore, these three aggregations are used in this paper.

The prolonger $P_{l,l+1}$ interpolates a vector on level l+1 to another on level l. When the l-th and the (l+1)-th level grid are determined, the basic construction scheme can be used to define $P_{l,l+1}$. The derived prolonger is related to constant interpolation. To improve the performance of the multigrid method, it can be smoothed too, to derive a new prolonger related to more advanced interpolations [6]. In this paper, the basic scheme is considered. In this scheme, if there are n_l nodes on the l-th level, $P_{l,l+1}$ is an n_l by n_{l+1} matrix. If a node i in the l-th level belongs to the j-th aggregation, the entry of $P_{l,l+1}$ on the i-th row and the j-th column is 1. All entries not equal to 1 are zeros. When $P_{l,l+1}$ is determined, A_{l+1} is defined by $A_{l+1} = (P_{l,l+1})^T A_l P_{l,l+1}$.

1. Compute $x^{(l)} = S_l(x^{(l)}, b^{(l)})$.
2. Compute $r^{(l+1)} = (P_{l,l+1})^T (b^{(l)} - A_l x^{(l)})$.
3. If (l=m-1 or the order of A_{l+1} is small enough) then
4. Compute $x^{(l+1)} = (A_{l+1})^{-1} r^{(l+1)}$.
5. Else
6. Set $c^{(l+1)}$=0 and $c^{(l+1)}$ = MGprec($c^{(l+1)}$, $r^{(l+1)}$, A_{l+1}, l+1).
7. If V-cycle then
8. Set $y^{(l+1)} = c^{(l+1)}$.
9. Else if W-cycle then
10. Compute $s^{(l+1)} = r^{(l+1)} - A_{l+1} c^{(l+1)}$.
11. Set $y^{(l+1)}$=0 and $y^{(l+1)}$ = MGprec($y^{(l+1)}$, $s^{(l+1)}$, A_{l+1}, l+1).
12. Else if K-cycle then
13. Compute $v^{(l+1)} = A_{l+1}c^{(l+1)}$, ρ_1=($c^{(l+1),}$ $v^{(l+1)}$) and α_1=($c^{(l+1),}$ $r^{(l+1)}$).
14. Compute $s^{(l+1)} = r^{(l+1)} – (\alpha_1/\rho_1) v^{(l+1)}$.
15. If $\|s^{(l+1)}\|_2 \le t \|r^{(l+1)}\|_2$ then
16. $y^{(l+1)} = (\alpha_1/\rho_1) c^{(l+1)}$.
17. Else
18. Set $d^{(l+1)}$=0 and $d^{(l+1)}$ = MGprec($d^{(l+1)}$, $s^{(l+1)}$, A_{l+1}, l+1).
19. Compute $w^{(l+1)} = A_{l+1}d^{(l+1)}$, γ=($d^{(l+1),}$ $v^{(l+1)}$) and β=($d^{(l+1),}$ $w^{(l+1)}$).
20. Compute α_2 = ($d^{(l+1),}$ $s^{(l+1)}$) and $\rho_2 = \beta - \gamma^2/\rho_1$.
21. Compute $y^{(l+1)} = (\alpha_2/\rho_2) d^{(l+1)} + (\alpha_1/\rho_1 - \gamma\alpha_2/(\rho_1\rho_2)) c^{(l+1)}$.
22. Endif
23. Endif
24. Compute $x^{(l)} = x^{(l)} + P_{l,l+1}y^{(l+1)}$.
25. Compute $x^{(l)} = S_l^T(x^{(l)}, b^{(l)})$.
26. Endif

FIGURE II. ALGORITHM $x^{(l)}$ = MGprec($x^{(l)}$, $b^{(l)}$, A_l, l)

On the l-th level, once the sparse linear system is given, we can denote it as $A_l x^{(l)} = b^{(l)}$. Two kinds of smoothers are

frequently used [3]. One is the Jacobi smoothing. For this case, we have

$$x^{(l)} := D_l^{-1}\{b^{(l)} - (A_l - D_l)\, x^{(l)}\}, \tag{3}$$

where D_l is a matrix with $D_l(i,i)=A_l(i,i)$ for i=1 to n_l and all others zero. The other is the Gauss-Seidel smoothing. For this case, we have

$$x^{(l)} := (D_l + L_l)^{-1}\{b^{(l)} - (A_l - D_l - L_l)\, x^{(l)}\}, \tag{4}$$

where L_l is a matrix, with $L_l(i,j)=A_l(i,j)$ for j=1 to i-1 and i=1 to n_l. All other entries of L_l are zero.

Therefore, in figure II, when Jacobi and Gauss-Seidel smoothing are used, step 1 can be detailed as (3) and (4) respectively. For $A_l - D_l$, $D_l + L_l$ and $A_l - D_l - L_l$ are the non-diagonal part, the lower triangular part and the strictly upper triangular part of A_l respectively, if CSR or MSR storage format [1] is used for matrix A_l, they can be accessed directly from the data structures for A_l in the implementation. Of course, these matrices can be stored independently too, which is beneficial to the computation performance.

Correspondently, for Jacobi and Gauss-Seidel smoothing, step 25 in figure II can be described as the following formulas respectively

$$x^{(l)} := D_l^{-T}\{b^{(l)} - (A_l - D_l)^T\, x^{(l)}\}, \tag{5}$$

$$x^{(l)} := (D_l + L_l)^{-T}\{b^{(l)} - (A_l - D_l - L_l)^T\, x^{(l)}\}.$$

III. Two Schemes to Improve the Implementation of Aggregation Based Multigrid Methods

In this section, two schemes will be considered to improve the implementation of algorithm MGprec described in figure II. Firstly, for algorithm MGprec is only used as the preconditioner of PCG, where the input vector $x^{(1)}$ to MGprec is a given vector. Only if it is invariant to the PCG iterations, $(A_1 - D_1)x^{(1)}$ and $(A_1 - D_1 - L_1)x^{(1)}$ can all be recomputed in the setup. At the same time, for $l > 1$, it is clear that the input vector $x^{(l)}$ to MGprec is always zero, thus the computation of $(A_l - D_l)\, x^{(l)}$ and $(A_l - D_l - L_l)\, x^{(l)}$ are not required at all.

In this paper, we consider the special case where the input vector $x^{(1)}$ to MGprec is zero too. For this case, the formula (3) and (4) can be simplified into

$$x^{(l)} := D_l^{-1} b^{(l)}, \tag{7}$$

$$x^{(l)} := (D_l + L_l)^{-1} b^{(l)}, \tag{8}$$

respectively. This means that the computation complexity of the pre-smoothing can be reduced significantly. For convenience, in the following context, the number of nonzero in a matrix B is denoted as nnz(B). For Jacobi smoothing, the number of floating operations is reduced from (2 nnz(A_l) – n_l) to n_l. For Gauss-Seidel smoothing, the value is reduced from (2 nnz(A_l) – n_l) to (2 nnz(L_l) + n_l).

The second scheme is to reconstruct the restriction process, that is, step 2 in figure II. It can be seen that the computation $r^{(l+1)} = (P_{l,l+1})^T (b^{(l)} - A_l\, x^{(l)})$ is equivalent to

$$r^{(l+1)} = (P_{l,l+1})^T b^{(l)} - ((P_{l,l+1})^T A_l)\, x^{(l)}. \tag{9}$$

If we denote $B_{l+1,l}=(P_{l,l+1})^T A_l$, it is known that it is a n_{l+1} by n_l matrix, and (9) can be rewritten as

$$r^{(l+1)} = (P_{l,l+1})^T b^{(l)} - B_{l+1,l}\, x^{(l)}. \tag{10}$$

For $P_{l,l+1}$ and A_l are both determined in the setup, $B_{l+1,l}$ can be recomputed too. Then the number of float operations of step 2 in figure II can be improved approximately from 2 nnz(A_l) + $n_l(k\text{-}1)/k$ to 2 nnz($B_{l+1,l}$) + $n_l(k\text{-}1)/k$, where k is the number of nodes aggregated each time on the average. Only if the number of non-zeros of matrix $B_{l+1,l}$ is smaller than that of A_l, the reformulation is profitable.

On the other hand, $(P_{l,l+1})^T$ is an n_{l+1} by n_l matrix, for a node i in level l+1, if it relates to the aggregation of nodes i_1, i_2, …, i_j in level l, for the i-th row, the entries in these columns are 1, and others 0. From the definition of aggregations, there is no joint between different aggregations, which means that in each column of $(P_{l,l+1})^T$, there is only one non-zero element 1, and all the others are 0. Therefore, when computing the matrix $B_{l+1,l} = (P_{l,l+1})^T A_l$, the i-th row is derived from the sum of the i_1-th, the i_2-th, …, and the i_j-th row of matrix A_l, and each row of A_l is used only once. Therefore, if the non-zero structure of a vector v is denoted as nz(v), nz($B_{l+1,l,i*}$) is equal to the union of nz($A_{l,k*}$), $k = i_1, i_2, \ldots, i_j$, where $B_{l+1,l,i*}$ is the i-th row of $B_{l+1,l}$, and $A_{l,k*}$ is the k-th row of A_l, that is,

$$\mathrm{nz}(B_{l+1,l,i*}) = \cup\{k \in i_1, i_2, \ldots, i_j : \mathrm{nz}(A_{l,k*})\}. \tag{11}$$

While the number of non-zeros in $B_{l+1,l}$ is equal to the sum of $|\mathrm{nz}(B_{l+1,l,i*})|$ for i=1,2,…,n_{l+1}, we can deduce that

$$\begin{aligned}\mathrm{nnz}(B_{l+1,l}) &= \textstyle\sum |\mathrm{nz}(B_{l+1,l,i*})| \\ &\le \textstyle\sum\sum\{k \in i_1, i_2, \ldots, i_j : |\mathrm{nz}(A_{l,k*})|\} = \mathrm{nnz}(A_l),\end{aligned}$$

that is

$$\mathrm{nnz}(B_{l+1,l}) \le \mathrm{nnz}(A_l). \tag{12}$$

This means that the number of non-zeros of matrix $B_{l+1,l}$ is actually smaller than that of A_l, so this reconstruction is more efficient than the original version.

IV. Numerical Experiments

In this section, all the experiments are performed on a processor of Intel(R) Xeon(R) CPU E5-4640 0 @ 2.40GHz (cache 20480 KB). The operating system is Linux version 2.6.32-431.el6.x86_64 and the compiler is Intel FORTRAN

Version 15.0.0.090. When algorithm PCG is used to solve linear systems, the initial approximation is selected as zero all the time and ε is selected as 1E-10. In multigrid method, the linear system on the coarsest level is solved with ILU(0) preconditioned CG and ε is selected as 1E-10 too.

Experiments are done for discrete linear systems derived from a 3-D model partial differential equations with finite difference method. The PDE is

$$-\partial^2 u/\partial x^2 - \partial^2 u/\partial y^2 - \partial^2 u/\partial z^2 = f, \quad (13)$$

where $x, y, z \in (0,1)$ and the function f and the boundary values are given from a true solution u=1. There are n+2 points are selected in each dimension and $u(x_i, y_j, z_k)$ is denoted as $u_{i,j,k}$ for any function u, where n=128 and

$$x_i = ih, y_j = jh, z_k = kh, i, j, k = 0,1, \ldots, n+1,$$

and h=1/(n+1). The discrete form used is

$$-\sum_{i'=i-1}^{i+1}\sum_{j'=j-1}^{j+1}\sum_{k'=k-1}^{k+1} u_{i',j',k'} + 27\, u_{i,j,k} = h^2 f_{i,j,k}. \quad (14)$$

Some experiments are given in table I and II for Jacobi and Gauss-Seidel smoothing respectively. In all the tests, the recursion process of the multigrid is stopped when the number of nodes on some level is not larger than 100, and all the timing results are given in seconds. In the tables, S2 denotes the aggregation scheme based on strong connection. S4 and S8 denote the scheme recursively invoking S2 with two and three times respectively. V-cyc and W-cyc means the multigrid with V-cycle and W-cycle respectively. K000 and K025 denote the K-cycle multigrid with t is selected as 0 and 0.25 respectively. Its denotes the number of PCG iterations. Itm1 denotes the time used for PCG iterations with the original implementation. Itm2 denotes the iteration time with the first scheme only. Item3 denotes the iteration time with both schemes. Sav2 and Sav3 mean the percentage of the iteration time saved when the first and both schemes are adopted respectively. In detail, they are computed as 1–Itm2/Itm1 and as 1–Itm3/Itm1 respectively.

TABLE I. RESULTS FROM JACOBI SMOOTHING

Method		Its	Itm1	Itm2	Itm3	Sav2	Sav3
S2	Vcyc	33	44.81	38.93	35.83	13.12%	20.04%
	Wcyc	14	128.4	105.0	96.10	18.22%	25.16%
	K000	15	154.2	131.3	123.7	14.85%	19.78%
	K025	16	54.77	47.38	44.95	13.49%	17.93%
S4	Vcyc	40	38.72	33.65	30.16	13.09%	22.11%
	Wcyc	17	24.40	21.29	18.33	12.75%	24.88%
	K000	15	23.17	20.50	17.93	11.52%	22.62%
	K025	19	26.38	23.23	20.62	11.94%	21.83%
S8	Vcyc	43	36.95	32.26	27.97	12.69%	24.30%
	Wcyc	22	22.17	19.57	16.30	11.73%	26.48%
	K000	16	16.88	15.02	12.61	11.02%	25.30%
	K025	21	21.32	18.84	16.15	11.63%	24.25%

TABLE II. RESULTS FROM GAUSS-SEIDEL SMOOTHING

Method		Its	Itm1	Itm2	Itm3	Sav2	Sav3
S2	Vcyc	30	54.00	46.86	42.62	13.22%	21.07%
	Wcyc	13	157.0	129.7	121.8	17.39%	22.42%
	K000	13	179.4	147.0	146.1	18.06%	18.56%
	K025	16	61.98	50.70	48.29	18.20%	22.09%
S4	Vcyc	36	47.28	38.36	36.28	18.87%	23.27%
	Wcyc	16	31.41	26.38	23.59	16.01%	24.90%
	K000	14	27.92	24.74	21.60	11.39%	22.64%
	K025	15	27.14	23.89	21.82	11.97%	19.60%
S8	Vcyc	40	46.17	37.67	34.66	18.41%	24.93%
	Wcyc	19	25.95	21.91	19.07	15.57%	26.51%
	K000	14	19.13	16.91	14.36	11.60%	24.93%
	K025	24	30.55	26.85	23.86	12.11%	21.90%

From the results, it is clear that, when both schemes are adopted, for Jacobi smoothing the improvement is from 17.9% to 26.5%. While in table II, when the Gauss-Seidel smoothing is used, it is from 18.5% to 26.5%. In addition, when the same smoothing and the same aggregation are used, the savings with W-cycle is much larger than that with V-cycle, which reflects the fact that there are two times to invoke MGprec in each W-cycle, and only one in V-cycle. For similar reasons, the savings with W-cycle is larger than that with V-cycle too in general. And when the same smoothing and the same cycle are used, in general, the savings with aggregation S8 is much larger than that with S4, which is again much larger than that with S2. It is consistent to the differences between the number of non-zeros of $(P_{l,l+1})^T A_l$ and that of A_l.

V. CONCLUSION

Two schemes to improve the aggregation based multigrid when used as the preconditioner are provided in this paper. One is aimed to the smoothing before the restriction process, to remove the trivial computation as much as possible. The other is to reconstruct the restriction process, using the fact that the number of non-zeros in matrix $(P_{l,l+1})^T A_l$ is smaller than that in A_l. The analyses and the numerical experiments for a model partial differential equation show that the provided schemes can reduce the solution time significantly. The improvements are more significant to W-cycle than to V-cycle and K-cycle. In future, these schemes will be tested in actual applications, including the meso-scale numerical simulation of concrete samples, where the coefficient matrix is more dense. Much larger efficiency improvements can be expected for this case.

ACKNOWLEDGMENT

This work is funded by NSFC(61379022).

REFERENCES

[1] Y. Saad. Iterative methods for sparse linear systems. PWS Publication Corporation, Boston, 1996

[2] R. Wienands, W. Joppich. Practical Fourier analysis for multigrid methods. Taylor and Francis Inc., 2004

[3] C. Wagner. Introduction to algebraic multigrid. Course Notes, University of Heidelberg, 1998/1999; available at: http://www.iwr.uni-heidelberg.de/~Christian.Wagner/, 1999

[4] H. Kim, J. Xu, and L. Zikatanov. A multigrid method based on graph matching for convection-diffusion equations, Numer. Linear Algebra Appl., 10(2003), 181-195

[5] Notay, Y.. An aggregation-based algebraic multigrid method. Electronic Transactions On Numerical Analysis, 37(2010), 123-146.

[6] Vanek P., Mandel J., and Brezina M.. Algebraic multigrid by smoothed aggregation for second order and fourth order elliptic problems. Computing, 56(1996):179-196

[7] P. Kumar. Aggregation based on graph matching and inexact coarse grid solve for algebraic two grid. International Journal of Computer Mathematics, 91:5(2014), 1061-1081

[8] D. Braess. Towards algebraic multigrid for elliptic problems of second order. Computing, 55(1995), 379-393

[9] Wu Jian-ping, Yin Fu-kang, Peng Jun, Yang Jin-hui. Research on Aggregations for Algebraic Multigrid Preconditioning Methods. In: 2017 2nd International Conference on Computer Science and Technology [CST2017], Guilin, China, 2017.