Accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization

NECULAI ANDREI
Research Institute for Informatics,
Center for Advanced Modeling and Optimization,
8-10, Averescu Avenue, Bucharest 1, Romania
and
Academy of Romanian Scientists,
54, Splaiul Independentei, Bucharest 5, Romania.
E-mail: nandrei@ici.ro

Abstract. An accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for solving unconstrained optimization problems is presented. The basic idea is to combine the scaled memoryless BFGS method and the preconditioning technique in the frame of the conjugate gradient method. The preconditioner, which is also a scaled memoryless BFGS matrix, is reset when the Beale-Powell restart criterion holds. The parameter scaling the gradient is selected as a spectral gradient. For the steplength computation the method has the advantage that in conjugate gradient algorithms the step lengths may differ from 1 by two order of magnitude and tend to vary unpredictably. Thus, we suggest an acceleration scheme able to improve the efficiency of the algorithm. Under common assumptions, the method is proved to be globally convergent. It is shown that for uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in the function values is significantly improved. In mild conditions the algorithm is globally convergent for strongly convex functions. Computational results for a set consisting of 750 unconstrained optimization test problems show that this new accelerated scaled conjugate gradient algorithm substantially outperforms known conjugate gradient methods: SCALCG [3-6], CONMIN by Shanno and Phua [42,43], Hestenes and Stiefel [25], Polak-Ribière-Polyak [32,33], Day and Yuan [17], Dai and Liao (t=1) [14], conjugate gradient with sufficient descent condition [7], hybrid Day and Yuan [17], hybrid Dai and Yuan zero [17], CG_DESCENT by Hager and Zhang [22,23], as well as quasi-Newton LBFGS method [26] and truncated Newton method by Nash [27].

Keywords: Unconstrained optimization; conjugate gradient method; spectral gradient method; Wolfe line search; BFGS preconditioning
49M07; 49M10; 90C06; 65K

1. Introduction

In this paper we consider the following unconstrained optimization problem:

$$min\ f(x) \tag{1.1}$$

where $f:R^n \to R$ is continuously differentiable and its gradient is available. We are interested in elaborating an algorithm for solving large-scale cases for which the Hessian of f is either not available or requires a large amount of storage and computational costs. Plenty of conjugate gradient methods are known, and an excellent survey of these methods, with a special attention on their global convergence, is given by Hager and Zhang [24]. Different conjugate gradient algorithms correspond to different choices for the scalar parameter β_k [8,16,21,36,37]. Line search in the conjugate gradient algorithms often is based on the standard Wolfe conditions. A numerical comparison of conjugate gradient algorithms with Wolfe line search, for different formulae of parameter β_k computation, including the Dolan and Moré performance profile, is given in [8].

The paper presents a conjugate gradient algorithm based on a combination of the scaled memoryless BFGS method and the preconditioning technique [3-6]. For general nonlinear functions a good preconditioner is any matrix that approximates $\nabla^2 f(x^*)^{-1}$, where x^* is a local solution of (1.1). In this algorithm the preconditioner is a scaled memoryless BFGS matrix which is reset when the Powell restart criterion holds. The scaling factor in the preconditioner is selected as spectral gradient [38].

The algorithm uses the conjugate gradient direction where the famous parameter β_k is obtained by equating the conjugate gradient direction with the direction corresponding to the Newton method. Thus, we get a general formula for the direction computation, which could be particularized to include the Polak-Ribiére [32] and Polyak [33] and the Fletcher and Reeves [20] conjugate gradient algorithms, the spectral conjugate gradient (SCG) by Birgin and Martínez [11] or the algorithm of Dai and Liao [14], for $t = 1$. This direction is then modified in a canonical manner as it was considered earlier by Oren and Luenberger [29], Oren and Spedicato [30], Perry [31] and Shanno [39-41], by means of a scaled, memoryless BFGS preconditioner placed into the Beale-Powell restart technology. This is the reason we call this a scaled memoryless BFGS preconditioned conjugate gradient algorithm. The scaling factor is computed in a spectral manner based on the inverse Rayleigh quotient, as suggested by Raydan [38]. The method could be considered as an extension of the spectral conjugate gradient (SCG) by Birgin and Martínez [11] or of a variant of the conjugate gradient algorithm by Dai and Liao [14] (for $t = 1$) suggested to overcome the lack of positive definiteness of the matrix defining their search direction.

In [28] Jorge Nocedal articulated a number of open problems in conjugate gradient algorithms. One of them focuses on the step length. Intensive numerical experiments with conjugate gradient algorithms proved that the step length may differ from 1 up to two orders of magnitude, being larger or smaller than 1, depending on how the problem is scaled. Moreover, the sizes of the step length tend to vary in a totally unpredictable way. This is in sharp contrast with the Newton and quasi-Newton methods, as well as with the limited memory quasi-Newton methods, which usually admit the unit step length for most of the iterations and require only very few function evaluations for step length determination. Therefore, in this paper we take the advantage of this behavior of the step lengths in conjugate gradient algorithms and present an acceleration scheme, which modify the step length in such a manner to improve the reduction in functions values.

The paper is organized as follows: In section 2 we present the scaled conjugate gradient algorithm BFGS preconditioned. The algorithm performs two types of steps: a standard one in which a double quasi-Newton updating scheme is used and a restart one where the current information is used to define the search direction. The convergence of the algorithm for strongly convex functions is proved in section 3. In section 4 we present an acceleration scheme of the algorithm. The idea of this computational scheme is to take advantage that the step lengths α_k in conjugate gradient algorithms are very different from 1. Therefore, we suggest we modify α_k in such a manner as to improve the reduction of the function values along the iterations. In section 5 we present the ASCALCG algorithm and we prove that for uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in function values is significantly improved. Finally, in section 6 we present computational results on a set of 750 unconstrained optimization problems from the CUTE [12] collection along with some other large-scale unconstrained optimization problems presented in [1]. The Dolan-Moré [19] performance profiles of ASCALCG versus some known conjugate gradient algorithms including Hestenes and Stiefel [25], Polak-Ribière-Polyak [32,33], Day and Yuan [17], hybrid Dai and Yuan [17], SCALCG by Andrei [3-6], CONMIN by Shanno and Phua [42,43], CG_DESCENT by Hager and Zhang [22,23], or limited memory quasi-Newton LBFGS by Liu and Nocedal [26] and truncated Newton TN by Nash [27] prove that ASCALCG is top performer among these algorithms.

2. Scaled Conjugate Gradient Method

The algorithm generates a sequence x_k of approximations to the minimum x^* of f, in which

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2.1}$$

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k, \tag{2.2}$$

where $g_k = \nabla f(x_k)$, α_k is selected to minimize $f(x)$ along the search direction d_k, β_k is a scalar parameter, $s_k = x_{k+1} - x_k$ and θ_{k+1} is a parameter or a matrix to be determined. The iterative process is initialized with an initial point x_0 and $d_0 = -g_0$.

Observe that if $\theta_{k+1} = 1$, then we get the classical conjugate gradient algorithms according to the value of the scalar parameter β_k. On the other hand, if $\beta_k = 0$, then we get another class of algorithms according to the selection of the parameter θ_{k+1}. Considering $\beta_k = 0$, there are two possibilities for θ_{k+1} : a positive scalar or a positive definite matrix. If $\theta_{k+1} = 1$, then we have the steepest descent algorithm. If $\theta_{k+1} = \nabla^2 f(x_{k+1})^{-1}$, or an approximation of it, then we get the Newton or the quasi-Newton algorithms, respectively. Therefore, we see that in the general case, when $\theta_{k+1} \neq 0$ is selected in a quasi-Newton manner, and $\beta_k \neq 0$, (2.2) represents a combination between the quasi-Newton and the conjugate gradient methods. However, if θ_{k+1} is a matrix containing some useful information about the inverse Hessian of function f, we are better off using $d_{k+1} = -\theta_{k+1} g_{k+1}$ since the addition of the term $\beta_k s_k$ in (2.2) may prevent the direction d_k from being a descent direction unless the line search is sufficiently accurate. Therefore, in this paper we shall consider θ_{k+1} as a positive scalar which contains some useful information to the inverse Hessian of function f.

As we know, when the initial point x_0 is close enough to a local minimum point x^*, then the best direction to be followed in the current point x_{k+1} is the Newton direction $-\nabla^2 f(x_{k+1})^{-1} g_{k+1}$. Therefore, our motivation is to choose the parameter β_k in (2.2) so that for every $k \geq 1$ the direction d_{k+1} given by (2.2) can be the best direction we know, i.e. the Newton direction. Hence, using the Newton direction from the equality

$$-\nabla^2 f(x_{k+1})^{-1} g_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k, \tag{2.3}$$

we get:

$$\beta_k = \frac{s_k^T \nabla^2 f(x_{k+1}) \theta_{k+1} g_{k+1} - s_k^T g_{k+1}}{s_k^T \nabla^2 f(x_{k+1}) s_k}. \tag{2.4}$$

Observe that the Newton direction is being used here only as a motivation for formula (2.4). The salient point with this formula for β_k computation is the presence of the Hessian. If the line search is exact we get the scaled Daniel method [18]. For large-scale problems, choices for the update parameter that do not require the evaluation of the Hessian matrix are often preferred in practice to the methods that require the Hessian.

Now, for quasi-Newton methods an approximation matrix B_k to the Hessian $\nabla^2 f(x_k)$ is used and updated so that the new matrix B_{k+1} satisfies the secant condition $B_{k+1} s_k = y_k$, where $y_k = g_{k+1} - g_k$. Therefore, in order to have an algorithm for solving large-scale problems we can assume that the pair (s_k, y_k) satisfies the secant condition. In this case, Zhang, Deng and Chen [46] proved that if $\|s_k\|$ is sufficiently small, then $s_k^T \nabla^2 f(x_{k+1}) s_k - s_k^T y_k = O(\|s_k\|^3)$. Therefore, using this assumption we get:

$$\beta_k = \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k}. \tag{2.5}$$

Birgin and Martínez [11] arrived at the same formula for β_k, but using a geometric interpretation of quadratic function minimization. The direction corresponding to β_k given in (2.5) is as follows:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k} s_k. \tag{2.6}$$

The following particularizations are obvious. If $\theta_{k+1} = 1$, then (2.6) is the direction considered by Perry [31]. At the same time we see that (2.6) is the direction given by Dai and Liao [14] for $t = 1$, obtained this time by an interpretation of the conjugacy condition. Additionally, if $s_j^T g_{j+1} = 0$, $j = 0,1,\dots,k$, then from (2.6) we get:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} y_k^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{2.7}$$

which is the direction corresponding to a *generalization of the Polak and Ribière* formula. Of course, if $\theta_{k+1} = \theta_k = 1$ in (2.7), we get the *classical Polak and Ribière* formula [32,33]. If $s_j^T g_{j+1} = 0$, $j = 0,1,\dots,k,$ and additionally the successive gradients are orthogonal, then from (2.6) we get:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} g_{k+1}^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{2.8}$$

which is the direction corresponding to a *generalization of the Fletcher and Reeves* formula [20]. Therefore, (2.6) is a general formula for direction computation in a conjugate gradient manner including the classical Fletcher and Reeves [20], and Polak-Ribière and Polyak [32, 33] formulae.

There is a result by Shanno [40, 41] that says that the conjugate gradient method is precisely the BFGS quasi-Newton method for which the initial approximation to the inverse of the Hessian, at every step, is taken as the identity matrix. The extension to the scaled conjugate gradient is very simple. Using the same methodology as considered by Shanno [40] we get the following direction d_{k+1}:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1} \left(\frac{g_{k+1}^T s_k}{y_k^T s_k} \right) y_k - \left[\left(1 + \theta_{k+1} \frac{y_k^T y_k}{y_k^T s_k} \right) \frac{g_{k+1}^T s_k}{y_k^T s_k} - \theta_{k+1} \frac{g_{k+1}^T y_k}{y_k^T s_k} \right] s_k, \tag{2.9}$$

involving only 4 scalar products. Again observe that if $g_{k+1}^T s_k = 0$, then (2.9) reduces to:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1} \frac{g_{k+1}^T y_k}{y_k^T s_k} s_k. \tag{2.10}$$

Thus, in this case, the effect is simply one of multiplying the Hestenes and Stiefel [25] search direction by a positive scalar.

In order to ensure the convergence of the algorithm (2.1), with d_{k+1} given by (2.9), we need to constrain the choice of α_k. We consider line searches that satisfy the Wolfe conditions [44,45]:

$$f(x_k + \alpha_k d_k) - f(x_k) \le \rho \alpha_k g_k^T d_k, \tag{2.11}$$

$$\nabla f(x_k + \alpha_k d_k)^T d_k \ge \sigma g_k^T d_k, \tag{2.12}$$

where $0 < \rho < 1/2 \le \sigma < 1$.

Theorem 2.1. *Suppose that α_k in (2.1) satisfies the Wolfe conditions (2.11) and (2.12), then the direction d_{k+1} given by (2.9) is a descent direction.*

Proof: Since $d_0 = -g_0$, we have $g_0^T d_0 = -\|g_0\|^2 \le 0$. Multiplying (2.9) by g_{k+1}^T, we have

$$g_{k+1}^T d_{k+1} = \frac{1}{(y_k^T s_k)^2}\Big[-\theta_{k+1}\|g_{k+1}\|^2 (y_k^T s_k)^2 + 2\theta_{k+1}(g_{k+1}^T y_k)(g_{k+1}^T s_k)(y_k^T s_k)$$
$$-(g_{k+1}^T s_k)^2 (y_k^T s_k) - \theta_{k+1}(y_k^T y_k)(g_{k+1}^T s_k)^2\Big].$$

Applying the inequality $u^T v \le \frac{1}{2}(\|u\|^2 + \|v\|^2)$ to the second term of the right hand side of the above equality, with $u = (s_k^T y_k) g_{k+1}$ and $v = (g_{k+1}^T s_k) y_k$ we get:

$$g_{k+1}^T d_{k+1} \le -\frac{(g_{k+1}^T s_k)^2}{y_k^T s_k}. \tag{2.13}$$

But, by Wolfe condition (2.12), $y_k^T s_k > 0$. Therefore, $g_{k+1}^T d_{k+1} < 0$ for every $k = 0,1,\ldots$ ■

Observe that the second Wolfe condition (2.12) is crucial for the descent character of direction (2.9). Besides, we see that the estimation (2.13) is independent of the parameter θ_{k+1}.

Usually, all conjugate gradient algorithms are periodically restarted. The Powell restarting procedure [34, 35] is to test if there is very little orthogonality left between the current gradient and the previous one. At step r when:

$$\left|g_{r+1}^T g_r\right| \ge 0.2\left\|g_{r+1}\right\|^2, \tag{2.14}$$

we restart the algorithm using the direction given by (2.9).

At step r we know s_r, y_r and θ_{r+1}. If (2.14) is satisfied, then a restart step is considered, i.e. the direction is computed as in (2.9). For $k \ge r+1$, we consider the same philosophy used by Shanno [40], where the gradient g_{k+1} is modified by a positive definite matrix which best estimates the inverse Hessian without any additional storage requirements, i.e. we compute:

$$v = \theta_{r+1} g_{k+1} - \theta_{r+1}\left(\frac{g_{k+1}^T s_r}{y_r^T s_r}\right) y_r$$
$$+\left[\left(1 + \theta_{r+1}\frac{y_r^T y_r}{y_r^T s_r}\right)\frac{g_{k+1}^T s_r}{y_r^T s_r} - \theta_{r+1}\frac{g_{k+1}^T y_r}{y_r^T s_r}\right] s_r, \tag{2.15}$$

and

$$w = \theta_{r+1} y_k - \theta_{r+1}\left(\frac{y_k^T s_r}{y_r^T s_r}\right) y_r$$
$$+\left[\left(1 + \theta_{r+1}\frac{y_r^T y_r}{y_r^T s_r}\right)\frac{y_k^T s_r}{y_r^T s_r} - \theta_{r+1}\frac{y_k^T y_r}{y_r^T s_r}\right] s_r, \tag{2.16}$$

involving 6 scalar products. With these, at any nonrestart step, the direction d_{k+1} for $k \ge r+1$, is computed using a double update scheme as in Shanno [40]:

$$d_{k+1} = -v + \frac{(g_{k+1}^T s_k) w + (g_{k+1}^T w) s_k}{y_k^T s_k} - \left(1 + \frac{y_k^T w}{y_k^T s_k}\right)\frac{g_{k+1}^T s_k}{y_k^T s_k} s_k, \tag{2.17}$$

involving only 4 scalar products. Observe that $y_k^T s_k > 0$ is sufficient to ensure that the direction d_{k+1} given by (2.17) is well defined and it is always a descent direction.

Motivated by the efficiency of the spectral gradient method introduced by Raydan [38] and used by Birgin and Martínez [11] in their spectral conjugate gradient method for unconstrained optimization, in our algorithm θ_{k+1} is defined as a scalar approximation to the inverse Hessian. This is given as the inverse of the Rayleigh quotient:

$$s_k^T \left[\int_0^1 \nabla^2 f(x_k + ts_k) dt \right] s_k \,/\, s_k^T s_k ,$$

i.e.

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k}. \tag{2.18}$$

The inverse of Rayleigh quotient lies between the smallest and the largest eigenvalue of the Hessian average $\int_0^1 \nabla^2 f(x_k + ts_k) dt$. Again observe $y_k^T s_k > 0$ is sufficient to ensure that θ_{k+1} in (2.18) is well defined.

3. Convergence Analysis for Strongly Convex Functions

Throughout this section we assume that f is strongly convex and ∇f is Lipschitz continuous on the level set

$$S = \left\{ x \in R^n : f(x) \le f(x_0) \right\}. \tag{3.1}$$

That is, there exists constants $\mu > 0$ and L such that

$$(\nabla f(x) - \nabla f(y))^T (x - y) \ge \mu \|x - y\|^2 \tag{3.2}$$

and

$$\|\nabla f(x) - \nabla f(y)\| \le L \|x - y\|, \tag{3.3}$$

for all x and y from S. For the convenience of the reader we include here the following lemma (see [22]).

Lemma 3.1. *Assume that d_k is a descent direction and ∇f satisfies the Lipschitz condition*

$$\|\nabla f(x) - \nabla f(x_k)\| \le L \|x - x_k\|, \tag{3.4}$$

for every x on the line segment connecting x_k and x_{k+1}, where L is a constant. If the line search satisfies the second Wolfe condition (2.12), then

$$\alpha_k \ge \frac{1-\sigma}{L} \frac{|g_k^T d_k|}{\|d_k\|^2}. \tag{3.5}$$

Proof: Subtracting $g_k^T d_k$ from both sides of (2.12) and using the Lipschitz condition we have

$$(\sigma - 1) g_k^T d_k \le (g_{k+1} - g_k)^T d_k \le L\alpha_k \|d_k\|^2 . \tag{3.6}$$

Since d_k is a descent direction and $\sigma < 1$, (3.5) follows immediately from (3.6). ■

Therefore, satisfying the Wolfe line search conditions α is bounded away from zero, i.e. there exists a positive constant ω, such that $\alpha \ge \omega$.

Lemma 3.2. *Assume that f is strongly convex and ∇f is Lipschitz continuous on S. If θ_{k+1} is selected by spectral gradient, then the direction d_{k+1} given by (2.9) satisfies:*

$$\|d_{k+1}\| \le \left(\frac{2}{\mu} + \frac{2L}{\mu^2} + \frac{L^2}{\mu^3} \right) \|g_{k+1}\|. \tag{3.7}$$

Proof: By Lipschitz continuity (3.3) we have

$$\|y_k\| = \|g_{k+1} - g_k\| = \|\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)\| \le L\alpha_k \|d_k\| = L \|s_k\|. \tag{3.8}$$

On the other hand, by strong convexity (3.2)

$$y_k^T s_k \geq \mu \|s_k\|^2. \tag{3.9}$$

Selecting θ_{k+1} as in (2.18), it follows that

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k} \leq \frac{\|s_k\|^2}{\mu \|s_k\|^2} = \frac{1}{\mu}. \tag{3.10}$$

Now, using the triangle inequality and the above estimates (3.8)-(3.10), after some algebra on $\|d_{k+1}\|$, where d_{k+1} is given by (2.9), we get (3.7). ■

The convergence of the scaled conjugate gradient algorithm when f is strongly convex is given by

Theorem 3.1. *Assume that f is strongly convex and ∇f is Lipschitz continuous on the level set S. If at every step of the conjugate gradient (2.1) with d_{k+1} given by (2.9) and the step length α_k selected to satisfy the Wolfe conditions (2.11) and (2.12), then either $g_k = 0$ for some k, or $\lim_{k\to\infty} g_k = 0.$*

Proof: Suppose $g_k \neq 0$ for all k. By strong convexity we have

$$y_k^T d_k = (g_{k+1} - g_k)^T d_k \geq \mu \alpha_k \|d_k\|^2. \tag{3.11}$$

By theorem 2.1, $g_k^T d_k < 0$. Therefore, the assumption $g_k \neq 0$ implies $d_k \neq 0$. Since $\alpha_k > 0$, from (3.11) it follows that $y_k^T d_k > 0$. But f is strongly convex over S, therefore f is bounded from below. Now, summing over k the first Wolfe condition (2.11) we have

$$\sum_{k=0}^{\infty} \alpha_k g_k^T d_k > -\infty.$$

Considering the lower bound for α_k given by (3.5) in Lemma 3.1 and having in view that d_k is a descent direction it follows that

$$\sum_{k=1}^{\infty} \frac{|g_k^T d_k|^2}{\|d_k\|^2} < \infty. \tag{3.12}$$

Now, from (2.13), using the inequality of Cauchy and (3.9) we get

$$g_{k+1}^T d_{k+1} \leq -\frac{(g_{k+1}^T s_k)^2}{y_k^T s_k} \leq -\frac{\|g_{k+1}\|^2 \|s_k\|^2}{\mu \|s_k\|^2} = -\frac{\|g_{k+1}\|^2}{\mu}.$$

Therefore, from (3.12) it follows that

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} < \infty. \tag{3.13}$$

Now, inserting the upperbound (3.7), for d_k in (3.13) yields

$$\sum_{k=0}^{\infty} \|g_k\|^2 < \infty,$$

which completes the proof. ■

For general functions the convergence of the algorithm is coming from theorem 2.1 and the restart procedure. Therefore, for strongly convex functions and under inexact line search it is global convergent. To a great extent, however, the algorithm is very close to the Perry/Shanno computational scheme [40, 41] which is a scaled memoryless BFGS preconditioned algorithm where the scaling factor is the inverse of a scalar approximation of the Hessian. If the Powell restart criterion (2.14) is used, for general functions f bounded from below with bounded second partial derivatives and bounded level set, using the same arguments considered by Shanno in [40] it is possible to prove that the iterates either converge to a point x^* satisfying $\|g(x^*)\| = 0,$ or the iterates cycle. It remains for further study to determine a complete global convergence result and whether cycling can occur for general functions with bounded second partial derivatives and bounded level set.

More sophisticated reasons for restarting the algorithms have been proposed in the literature, but we are interested in the performance of an algorithm that uses the Powell restart criterion, associated with the scaled memoryless BFGS preconditioned direction choice for restart. Additionally, some convergence analysis with Powell restart criterion was given by Dai and Yuan [15] and can be used in this context of the preconditioned and scaled memoryless BFGS algorithm.

4. Acceleration of the algorithm

It is common to see that in conjugate gradient algorithms the search directions tend to be poorly scaled and as a consequence the line search must perform more function evaluations in order to obtain a suitable steplength α_k. In order to improve the performances of the conjugate gradient algorithms the efforts were directed to design procedures for direction computation based on the second order information. For example, CONMIN [42], and SCALCG [3-6] take this idea of BFGS preconditioning. In this section we focus on the step length modification. In the context of gradient descent algorithm with backtracking the step length modification has been considered for the first time in [2].

Jorge Nocedal [28] pointed out an open problem in conjugate gradient algorithms that in these methods the step lengths may differ from 1 in a very unpredictable manner. They can be larger or smaller than 1 depending on how the problem is scaled. Numerical comparisons between conjugate gradient methods and the limited memory quasi Newton method, by Liu and Nocedal [26], show that the latter is more successful [8]. One explanation of the efficiency of the limited memory quasi-Newton method is given by its ability to accept unity step lengths along the iterations. In this section we take advantage of this behavior of conjugate gradient algorithms and present an acceleration scheme. Basically, this modifies the step length in a multiplicative manner to improve the reduction of the function values along the iterations [9, 10].

Given the initial point x_0 we can compute $f_0 = f(x_0)$, $g_0 = \nabla f(x_0)$ and by Wolfe line search conditions (2.11) and (2.12) the steplength α_0 is determined. With these, the next iteration is computed as: $x_1 = x_0 + \alpha_0 d_0$, $(d_0 = -g_0)$ where f_1 and g_1 are immediately determined and the direction d_1 can be computed as in (2.9). Therefore, at the iteration $k = 1, 2, \ldots$ we know x_k, f_k, g_k and d_k. Suppose that d_k is a descent direction. By the Wolfe line search (2.11) and (2.12) we can compute α_k with which the following point $z = x_k + \alpha_k d_k$ is determined. The first Wolfe condition (2.11) shows that the steplength $\alpha_k > 0$ satisfies:

$$f(z) = f(x_k + \alpha_k d_k) \le f(x_k) + \rho \alpha_k g_k^T d_k.$$

With these, let us introduce the accelerated conjugate gradient algorithm by means of the following iterative scheme:

$$x_{k+1} = x_k + \gamma_k \alpha_k d_k, \tag{4.1}$$

where $\gamma_k > 0$ is a parameter which follows to be determined in such a manner as to improve the behavior of the algorithm. Now, we have:

$$f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k g_k^T d_k + \frac{1}{2}\alpha_k^2 d_k^T \nabla^2 f(x_k) d_k + o\left(\|\alpha_k d_k\|^2\right). \tag{4.2}$$

On the other hand, for $\gamma > 0$ we have:

$$f(x_k+\gamma\alpha_k d_k)=f(x_k)+\gamma\alpha_k g_k^T d_k+\frac{1}{2}\gamma^2\alpha_k^2 d_k^T\nabla^2 f(x_k)d_k+o\left(\left\|\gamma\alpha_k d_k\right\|^2\right). \tag{4.3}$$

With these we can write:

$$f(x_k+\gamma\alpha_k d_k)=f(x_k+\alpha_k d_k)+\Psi_k(\gamma), \tag{4.4}$$

where

$$\begin{aligned}\Psi_k(\gamma)=&\frac{1}{2}(\gamma^2-1)\alpha_k^2 d_k^T\nabla^2 f(x_k)d_k+(\gamma-1)\alpha_k g_k^T d_k\\&+\gamma^2\alpha_k o\left(\alpha_k\left\|d_k\right\|^2\right)-\alpha_k o\left(\alpha_k\left\|d_k\right\|^2\right).\end{aligned} \tag{4.5}$$

Let us denote:

$$\begin{aligned}a_k&=\alpha_k g_k^T d_k\le 0,\\ b_k&=\alpha_k^2 d_k^T\nabla^2 f(x_k)d_k,\\ \varepsilon_k&=o\left(\alpha_k\left\|d_k\right\|^2\right).\end{aligned}$$

Observe that $a_k\le 0$, since d_k is a descent direction, and for convex functions $b_k\ge 0$. Therefore,

$$\Psi_k(\gamma)=\frac{1}{2}(\gamma^2-1)b_k+(\gamma-1)a_k+\gamma^2\alpha_k\varepsilon_k-\alpha_k\varepsilon_k. \tag{4.6}$$

Now, we see that $\Psi_k'(\gamma)=(b_k+2\alpha_k\varepsilon_k)\gamma+a_k$ and $\Psi_k'(\gamma_m)=0$, where

$$\gamma_m=-\frac{a_k}{b_k+2\alpha_k\varepsilon_k}. \tag{4.7}$$

Observe that $\Psi_k'(0)=a_k<0$. Therefore, assuming that $b_k+2\alpha_k\varepsilon_k>0$, then $\Psi_k(\gamma)$ is a convex quadratic function with minimum value in point γ_m and

$$\Psi_k(\gamma_m)=-\frac{(a_k+(b_k+2\alpha_k\varepsilon_k))^2}{2(b_k+2\alpha_k\varepsilon_k)}\le 0.$$

Considering $\gamma=\gamma_m$ in (4.4) and since $b_k\ge 0$, we see that for every k

$$f(x_k+\gamma_m\alpha_k d_k)=f(x_k+\alpha_k d_k)-\frac{(a_k+(b_k+2\alpha_k\varepsilon_k))^2}{2(b_k+2\alpha_k\varepsilon_k)}\le f(x_k+\alpha_k d_k),$$

which is a possible improvement of the values of function f (when $a_k+(b_k+2\alpha_k\varepsilon_k)\ne 0$). Therefore, using this simple multiplicative modification of the stepsize α_k as $\gamma_k\alpha_k$ where $\gamma_k=\gamma_m=-a_k/(b_k+2\alpha_k\varepsilon_k)$ we get:

$$\begin{aligned}f(x_{k+1})&=f(x_k+\gamma_k\alpha_k d_k)\le f(x_k)+\rho\alpha_k g_k^T d_k-\frac{(a_k+(b_k+2\alpha_k\varepsilon_k))^2}{2(b_k+2\alpha_k\varepsilon_k)}\\&=f(x_k)-\left[\frac{(a_k+(b_k+2\alpha_k\varepsilon_k))^2}{2(b_k+2\alpha_k\varepsilon_k)}-\rho a_k\right]\le f(x_k),\end{aligned} \tag{4.8}$$

since $a_k\le 0$, (d_k is a descent direction).

Since $\Psi(\gamma_k)\le\Psi(1)=0$, $\rho<1$ and $a_k\le 0$, then neglecting the contribution of ε_k, we still get an improvement on the function values as

$$f(x_{k+1})\le f(x_k)-\left[\frac{(a_k+b_k)^2}{2b_k}-\rho a_k\right]\le f(x_k).$$

Now, in order to get the algorithm we have to determine a way for b_k computation. For this, at point $z = x_k + \alpha_k d_k$ we have:

$$f(z) = f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k g_k^T d_k + \frac{1}{2}\alpha_k^2 d_k^T \nabla^2 f(\tilde{x}_k) d_k,$$

where $\tilde{x}_k$ is a point on the line segment connecting x_k and z. On the other hand, at point $x_k = z - \alpha_k d_k$ we have:

$$f(x_k) = f(z - \alpha_k d_k) = f(z) - \alpha_k g_z^T d_k + \frac{1}{2}\alpha_k^2 d_k^T \nabla^2 f(\bar{x}_k) d_k,$$

where $g_z = \nabla f(z)$ and $\bar{x}_k$ is a point on the line segment connecting x_k and z. Having in view the local character of searching and that the distance between x_k and z is small enough, we can consider $\tilde{x}_k = \bar{x}_k = x_k$. So, adding the above equalities we get:

$$b_k = -\alpha_k y_k^T d_k, \tag{4.9}$$

where $y_k = g_k - g_z$.

Observe that if $|a_k| > b_k$, then $\gamma_k > 1$. On the other hand, if $|a_k| \le b_k$, then $\gamma_k \le 1$. Therefore, if $|a_k| \ne b_k$, then $\gamma_k \ne 1$ and the steplength α_k computed by Wolfe conditions will be modified by its increasing or its reducing through factor γ_k.

5. ASCALCG Algorithm

Having in view the above developments and the definitions of g_k, s_k and y_k, as well as the selection procedure for θ_{k+1} computation, the following accelerated scaled conjugate gradient algorithm can be presented.

Step 1. Initialization. Select $x_0 \in R^n$, and the parameters $0 < \rho \le \sigma < 1$. Compute $f(x_0)$ and $g_0 = \nabla f(x_0)$. Set $d_0 = -g_0$ and $\alpha_0 = 1/\|g_0\|$. Set $k = 0$.

Step 2. Line search. Compute α_k satisfying the Wolfe conditions (2.11) and (2.12). Update the variables $x_{k+1} = x_k + \alpha_k d_k$. Compute $f(x_{k+1}), g_{k+1}$ and $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$.

Step 3. Test for continuation of iterations. If this test is satisfied the iterations are stopped, else set $k = k+1$.

Step 4. Scaling factor computation. Compute θ_k using (2.18).

Step 5. Restart direction. Compute the (restart) direction d_k as in (2.9).

Step 6. Line search. Compute the initial guess: $\alpha_k = \alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2$. Using this initialization compute α_k satisfying the Wolfe conditions. Update the variables $x_{k+1} = x_k + \alpha_k d_k$. Compute $f(x_{k+1})$ and g_{k+1}.

Step 7. Acceleration scheme. Compute $a_k = g_k^T d_k$ and $b_k = (g_k - g_{k+1})^T d_k$. If $b_k \ne 0$, then compute $\gamma_k = a_k / b_k$ and update the variables as: $x_{k+1} = x_k + \gamma_k \alpha_k d_k$. Compute $f(x_{k+1})$, g_{k+1} and $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$. Otherwise (if $b_k = 0$), then compute $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$.

Step 8. Store: $\theta = \theta_k$, $s = s_k$ and $y = y_k$.

Step 9. Test for continuation of iterations. If this test is satisfied the iterations are stopped, else set $k = k+1$.

Step 10. Restart. If the Powell restart criterion (2.14) is satisfied, then go to step 4 (a restart step); otherwise continue with step 11 (a standard step).

Step 11. Standard direction. Compute the direction d_k as in (2.17), where v and w are computed as in (2.15) and (2.16) with saved values θ, s and y.

Step 12. Line search. Compute the initial guess: $\alpha_k = \alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2$. Using this initialization compute α_k satisfying the Wolfe conditions. Update the variables $x_{k+1} = x_k + \alpha_k d_k$. Compute $f(x_{k+1})$ and g_{k+1}.

Step 13. Acceleration scheme. Compute $a_k = g_k^T d_k$ and $b_k = (g_k - g_{k+1})^T d_k$. If $b_k \neq 0$, then compute $\gamma_k = a_k / b_k$ and update the variables as: $x_{k+1} = x_k + \gamma_k \alpha_k d_k$. Compute $f(x_{k+1})$, g_{k+1} and $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$. Otherwise (if $b_k = 0$), then compute $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$.

Step 14. Test for continuation of iterations. If this test is satisfied the iterations are stopped, else set $k = k+1$ and go to step 10. ■

It is well known that if f is bounded below along the direction d_k, then there exists a step length α_k satisfying the Wolfe conditions. The initial selection of the step length crucially affects the practical behavior of the algorithm. At every iteration $k \geq 1$ the starting guess for the step α_k in line search is computed as $\alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2$. This procedure was considered for the first time by Shanno and Phua in CONMIN [42]. The same one is taken by Birgin and Martínez in SCG [11] and in the SCALCG algorithm [3-6]. In steps 3, 9 and 14 we can consider, for example, the following test: $\|\nabla f(x_k)\|_\infty \leq 10^{-6}$.

Proposition 5.1. *Suppose that f is an uniformly convex function on the level set $S = \{x : f(x) \leq f(x_0)\}$, and d_k satisfies the sufficient descent condition $g_k^T d_k < -c_1 \|g_k\|^2$, where $c_1 > 0$, and $\|d_k\|^2 \leq c_2 \|g_k\|^2$, where $c_2 > 0$. Then the sequence generated by ASCALCG converges linearly to x^*, solution to the problem (1.1).*

Proof. From (4.8) we have that $f(x_{k+1}) \leq f(x_k)$ for all k. Since f is bounded below, it follows that

$$\lim_{k\to\infty}(f(x_k) - f(x_{k+1})) = 0.$$

Now, since f is uniformly convex there exist positive constants m and M, such that $mI \leq \nabla^2 f(x) \leq MI$ on S. Suppose that $x_k + \alpha d_k \in S$ and $x_k + \gamma_m \alpha d_k \in S$ for all $\alpha > 0$. We have:

$$f(x_k + \gamma_m \alpha d_k) \leq f(x_k + \alpha d_k) - \frac{(a_k + b_k)^2}{2b_k}.$$

But, from uniform convexity we have the following quadratic upper bound on $f(x_k + \alpha d_k)$:

$$f(x_k + \alpha d_k) \leq f(x_k) + \alpha g_k^T d_k + \frac{1}{2} M \alpha^2 \|d_k\|^2.$$

Therefore,

$$f(x_k + \alpha d_k) \leq f(x_k) - \alpha c_1 \|g_k\|^2 + \frac{1}{2} M c_2 \alpha^2 \|g_k\|^2$$
$$= f(x_k) + \left[-c_1 \alpha + \frac{1}{2} M c_2 \alpha^2 \right] \|g_k\|^2.$$

Observe that for $0 \le \alpha \le c_1/(Mc_2)$, $-c_1\alpha + \frac{1}{2}Mc_2\alpha^2 \le -\frac{c_1}{2}\alpha$ which follows from the convexity of $-c_1\alpha + (Mc_2/2)\alpha^2$. Using this result, since $\rho < 1/2$, we get:

$$f(x_k + \alpha d_k) \le f(x_k) - \frac{1}{2}c_1\alpha \|g_k\|^2 \le f(x_k) - \rho c_1\alpha \|g_k\|^2,$$

From Lemma 3.1 the Wolfe line search terminates with a value $\alpha \ge \omega > 0$. Therefore, for $0 \le \alpha \le c_1/(Mc_2)$, this provides a lower bound on the decrease in the function f, i.e.

$$f(x_k + \alpha d_k) \le f(x_k) - \rho c_1 \omega \|g_k\|^2. \tag{5.1}$$

On the other hand,

$$\frac{(a_k + b_k)^2}{2b_k} \ge \frac{(\alpha^2 Mc_2 - \alpha c_1)^2 \|g_k\|^4}{2\alpha^2 Mc_2 \|g_k\|^2} \ge \frac{(\omega Mc_2 - c_1)^2}{2Mc_2}\|g_k\|^2. \tag{5.2}$$

Considering (5.1) and (5.2) we get:

$$f(x_k + \gamma_m \alpha d_k) \le f(x_k) - \rho c_1 \omega \|g_k\|^2 - \frac{(\omega Mc_2 - c_1)^2}{2Mc_2}\|g_k\|^2. \tag{5.3}$$

Therefore,

$$f(x_k) - f(x_k + \gamma_m \alpha d_k) \ge \left[\rho c_1 \omega + \frac{(\omega Mc_2 - c_1)^2}{2Mc_2}\right]\|g_k\|^2.$$

But, $f(x_k) - f(x_{k+1}) \to 0$ and as a consequence g_k goes to zero, i.e. x_k converges to x^*. Having in view that $f(x_k)$ is a nonincreasing sequence, it follows that $f(x_k)$ converges to $f(x^*)$. From (5.3) we see that

$$f(x_{k+1}) \le f(x_k) - \left[\rho c_1 \omega + \frac{(\omega Mc_2 - c_1)^2}{2Mc_2}\right]\|g_k\|^2. \tag{5.4}$$

Combining this with $\|g_k\|^2 \ge 2m(f(x_k) - f^*)$ and subtracting f^* from both sides of (5.4) we conclude:

$$f(x_{k+1}) - f^* \le c(f(x_k) - f^*),$$

where

$$c = 1 - 2m\left[\rho c_1 \omega + \frac{(\omega Mc_2 - c_1)^2}{2Mc_2}\right] < 1.$$

Therefore, $f(x_k)$ converges to f^* at least as fast as a geometric series with a factor that depends on the parameter ρ in the first Wolfe condition and the bounds m and M. Therefore, the convergence of the acceleration scheme is at least linear. ■

Observe that for strongly convex functions f with ∇f Lipschitz continuous, in Lemma 3.2 we proved that the direction d_{k+1} given by (2.9) is bounded as in (3.7). Therefore the condition $\|d_k\|^2 \le c_2 \|g_k\|^2$ is satisfied with $c_2 = \left(\frac{2}{\mu} + \frac{2L}{\mu^2} + \frac{L^2}{\mu^3}\right)^2$.

6. Computational results and comparisons

In this section we present the performance of a Fortran implementation of the *ASCALCG – accelerated scaled conjugate gradient algorithm* on a set of 750 unconstrained optimization test problems. At the same time, we compare the performance of ASCALCG with some conjugate gradient algorithms including SCALCG [3-6], CONMIN [42], Hestenes-Stiefel

(HS) [25], Polak-Ribière-Polyak (PRP) [32,33], Dai-Yuan (DY) [17], Dai-Liao (DL) [14], conjugate gradient with sufficient descent CGSD [7], hybrid Dai-Yuan (hDY) [17], hybrid Dai-Yuan zero (hDYz) [17], CG_DESCENT [22,23], and limited memory quasi-Newton LBFGS (m=3, m=5) by Liu and Nocedal [26] and truncated Newton TN by Nash [27].

All codes are written in Fortran and compiled with f77 (default compiler settings) on an Intel Pentium 4, 1.5Ghz. All algorithms implement the same stopping criterion $\|g_k\|_\infty \le \varepsilon_g$, where $\|.\|_\infty$ denotes the maximum absolute component of a vector and $\varepsilon_g = 10^{-6}$.

The test problems are the unconstrained problems in the CUTE [12] collection, along with other large-scale optimization problems [1]. We selected 75 large-scale unconstrained optimization problems in extended or generalized form. For each function we have considered 10 numerical experiments with number of variables $n = 1000, 2000, \ldots, 10000$.

The comparisons of algorithms are given in the following context. Let f_i^{ALG1} and f_i^{ALG2} be the optimal value found by ALG1 and ALG2, for problem $i = 1, \ldots, 750$, respectively. We say that, in the particular problem i, the performance of ALG1 was better than the performance of ALG2 if:

$$\left| f_i^{ALG1} - f_i^{ALG2} \right| < 10^{-3} \tag{6.1}$$

and the number of iterations, or the number of function-gradient evaluations, or the CPU time of ALG1 was less than the number of iterations, or the number of function-gradient evaluations, or the CPU time corresponding to ALG2, respectively.

In the first set of numerical experiments we compare ASCALCG with SCALCG. Basically, SCALCG [3-6] is the unaccelerated variant of ASCALCG. Figure 1 presents the Dolan-Moré [19] CPU performance profiles of these algorithms, i.e. we plot the fraction of problems for which the given method is within a factor τ of the best time.

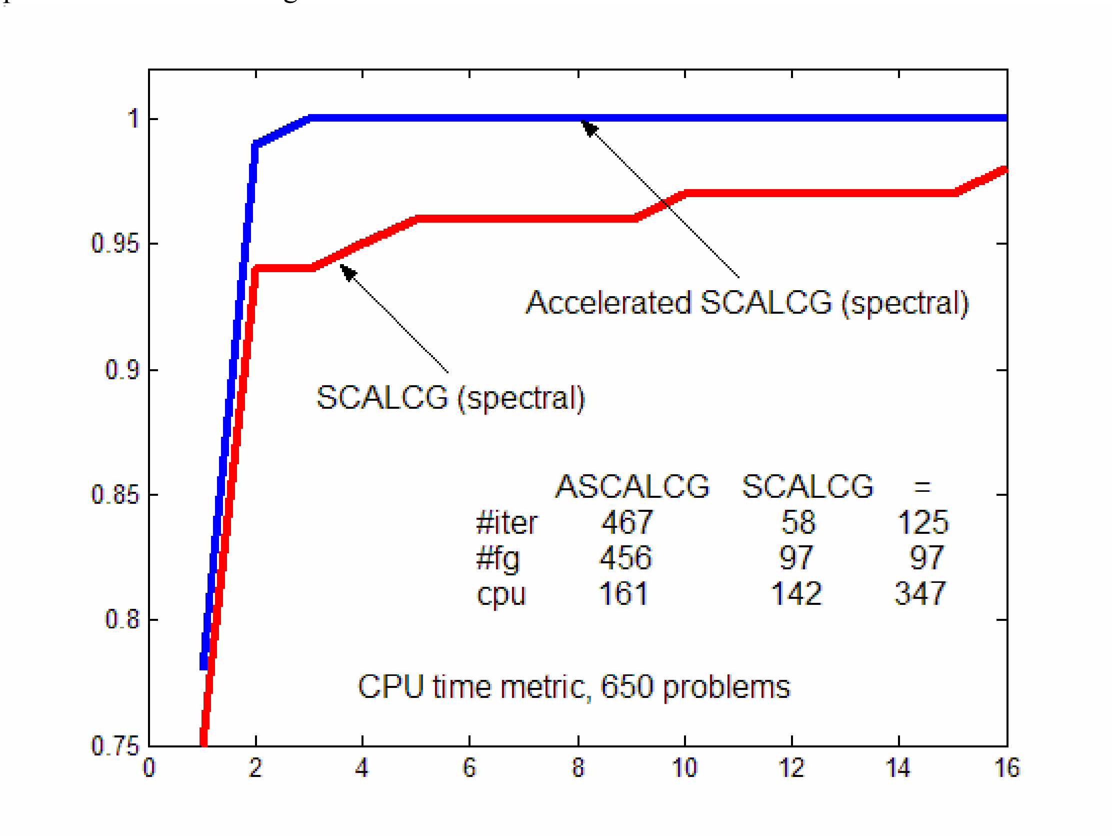

Fig. 1. ASCALCG versus SCALCG.

The percentage of the test problems for which a method is the fastest is given on the left axis of the plot. The right side of the plot gives the percentage of the test problems that were successfully solved by these algorithms, respectively. When comparing ASCALCG with SCALCG (Figure 1), subject to the number of iterations, we see that ASCALCG was better in 467 problems (i.e. it achieved the minimum number of iterations in 467 problems). SCALCG was better in 58 problems and they achieved the same number of iterations in 125 problems, etc. Out of 750 problems, only for 650 problems does the criterion (6.1) hold. Clearly, introducing the acceleration scheme represents an important ingredient in getting an efficient conjugate gradient algorithm. Numerical experiments proved that for the majority of iterations $\gamma_k = a_k / b_k < 1$, i.e. the acceleration scheme has the propensity to reduce the values of the step lengths. We see that the best performance, relative to the CPU time metric, was obtained by ASCALCG, the top curve in Figure 1. Hence, ASCALCG appears to generate the best steplength, on average.

In the second set of numerical experiments we compare ASCALCG versus to CONMIN developed by Shanno and Phua [42,43] (See also [40]). Shanno [40] and Perry [31] showed that their version of conjugate gradient algorithm can be viewed as memoryless BFGS method. The advantage of their approach is that their methods generate directions of descent. Furthermore, versions of their method are globally convergent for strictly convex functions and for Lipschitzian functions under the assumption that $\lim_{k\to\infty} \|s_k\| = 0$. Figure 2 presents the Dolan-Moré CPU performance profiles of ASCALCG versus CONMIN.

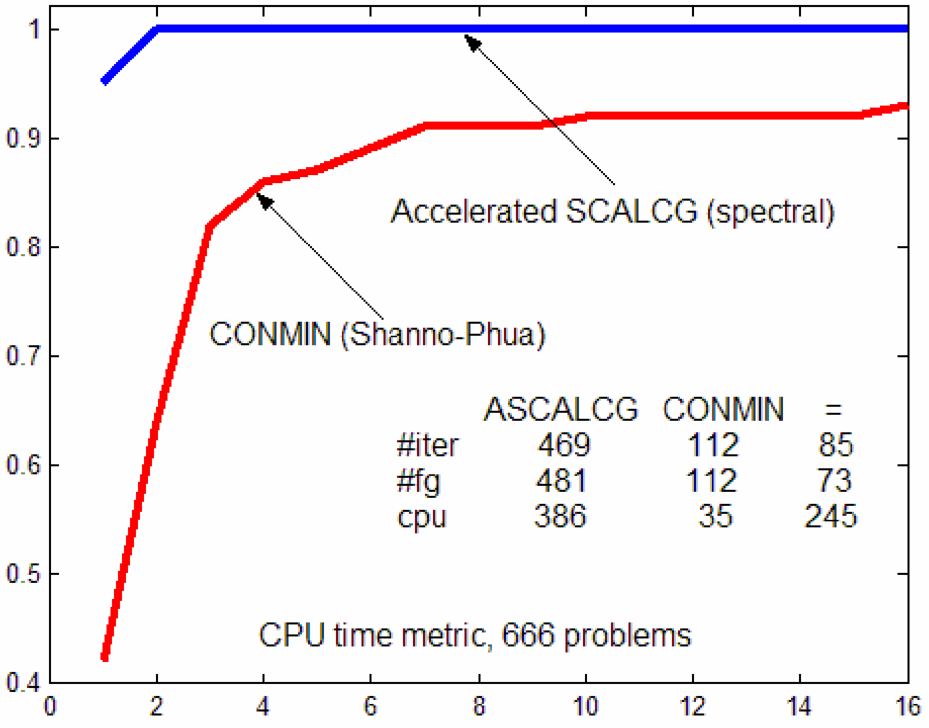

Fig. 2. ASCALCG versus CONMIN (Shanno-Phua)

We see that ASCALCG is clearly superior. One reason is that it uses more vectors to calculate the direction d_k. (ASCALCG uses 11 vectors and CONMIN only 7). Besides, ASCALCG uses the accelerate scheme which is a crucial ingredient on the performance of the algorithm.

In the third set of numerical experiments we compare ASCALCG to: HS ($\beta_k^{HS} = y_k^T g_{k+1} / y_k^T s_k$), PRP ($\beta_k^{PRP} = y_k^T g_{k+1} / g_k^T g_k$), DY ($\beta_k^{DY} = g_{k+1}^T g_{k+1} / y_k^T s_k$), DL (t=1) ($\beta_k^{DL} = g_{k+1}^T (y_k - t s_k) / y_k^T s_k$),CGSD ($\beta_k^{CGSD} = g_{k+1}^T g_{k+1} / y_k^T s_k - (y_k^T g_{k+1})(s_k^T g_{k+1}) / (y_k^T s_k)^2$), hDY ($\beta_k^{hDY} = \max\left\{-c\beta_k^{DY}, \min\left\{\beta_k^{HS}, \beta_k^{DY}\right\}\right\}, c = (1-\sigma)/(1+\sigma)$, $\sigma = 0.8$) and hDYz ($\beta_k^{hDYz} = \max\left\{0, \min\left\{\beta_k^{HS}, \beta_k^{DY}\right\}\right\}$). Figures 3-9 present the Dolan-Moré CPU performance profiles of these conjugate gradient algorithms.

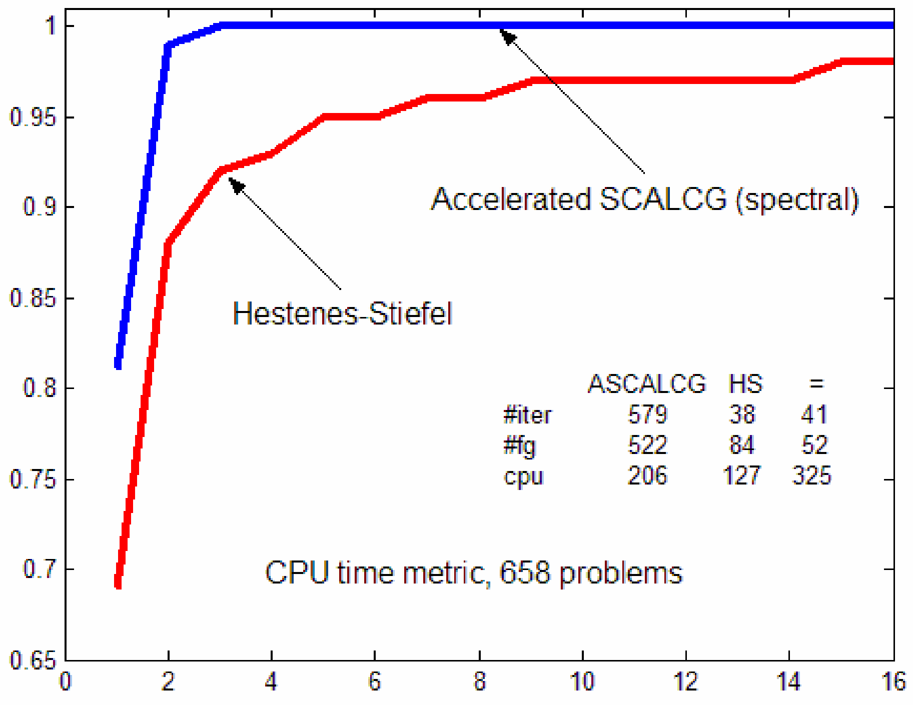

Fig. 3. ASCALCG versus Hestenes – Stiefel (HS).

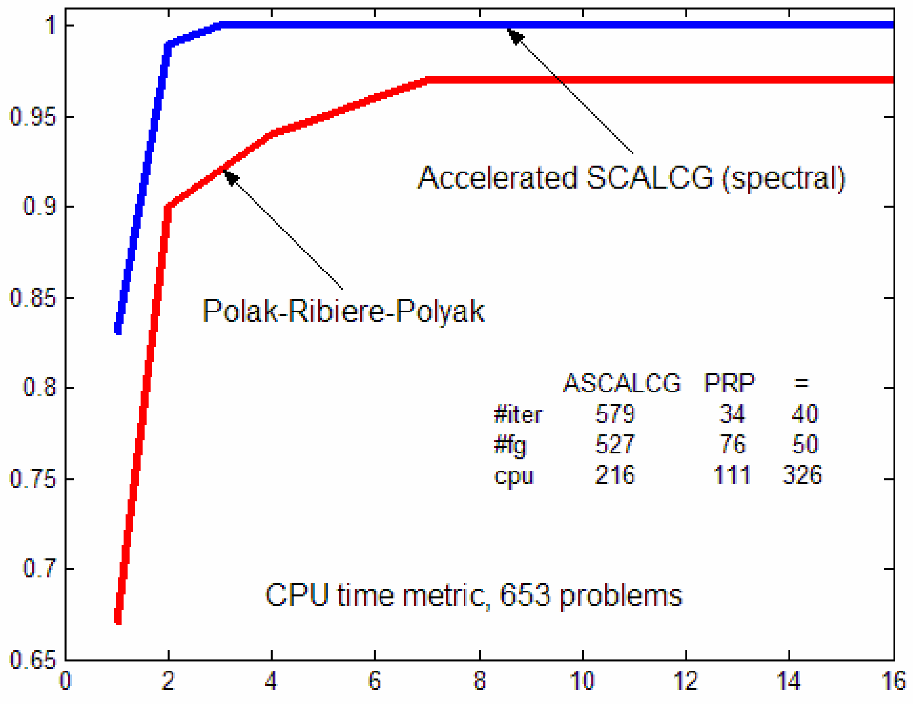

Fig. 4. ASCALCG versus Polak-Ribière-Polyak (PRP).

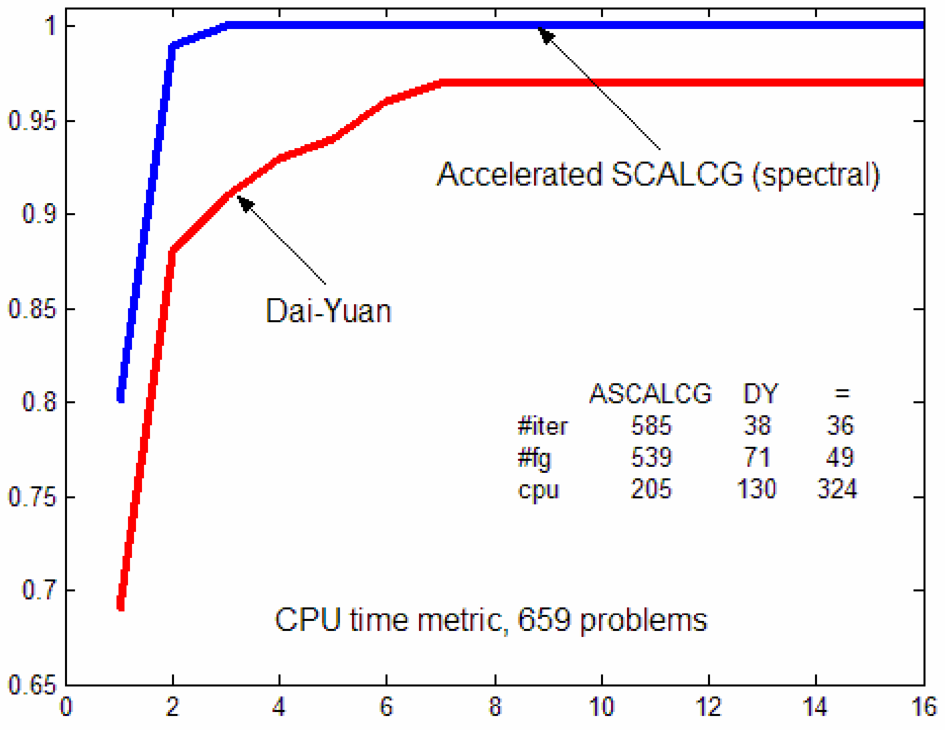

Fig. 5. ASCALCG versus Dai-Yuan (DY).

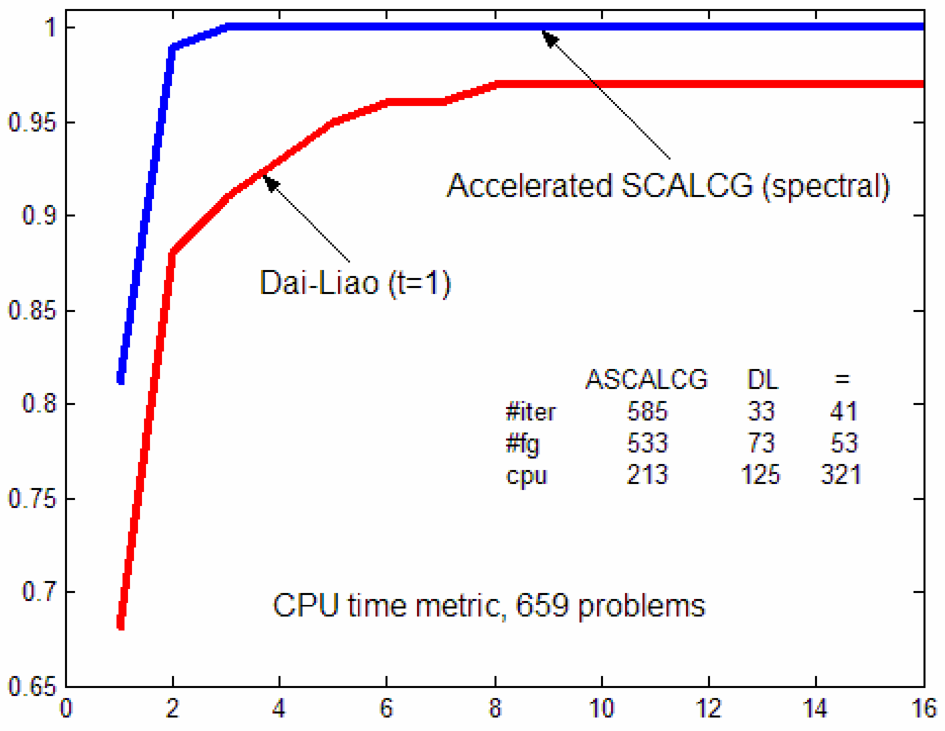

Fig. 6. ASCALCG versus Dai-Liao (t=1) (DL).

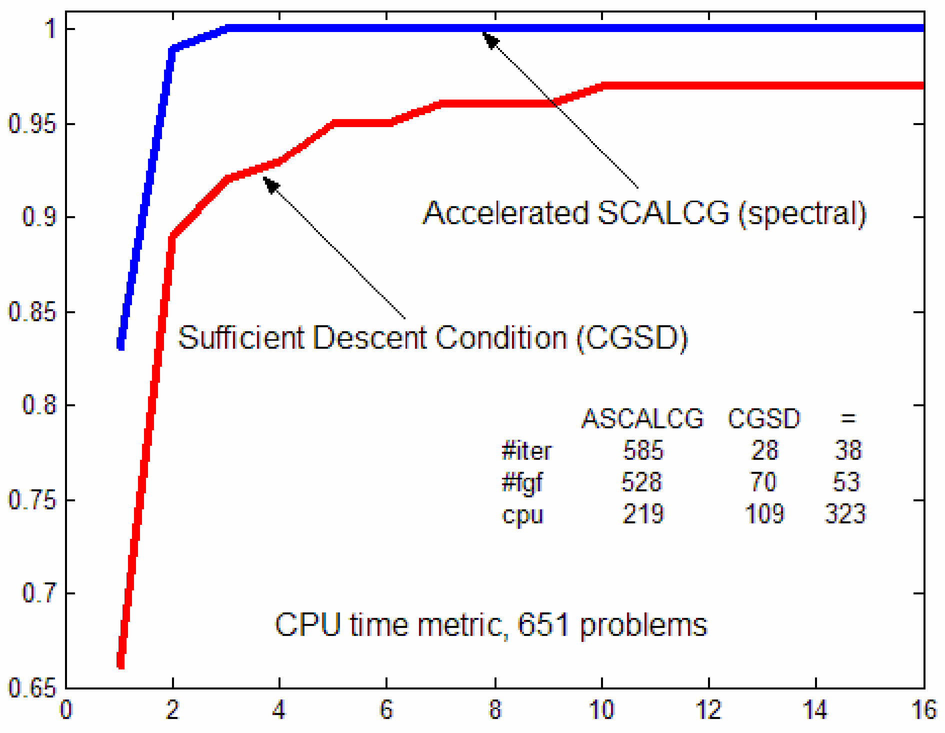

Fig. 7. ASCALCG versus CG with Sufficient Descent Condition (CGSD).

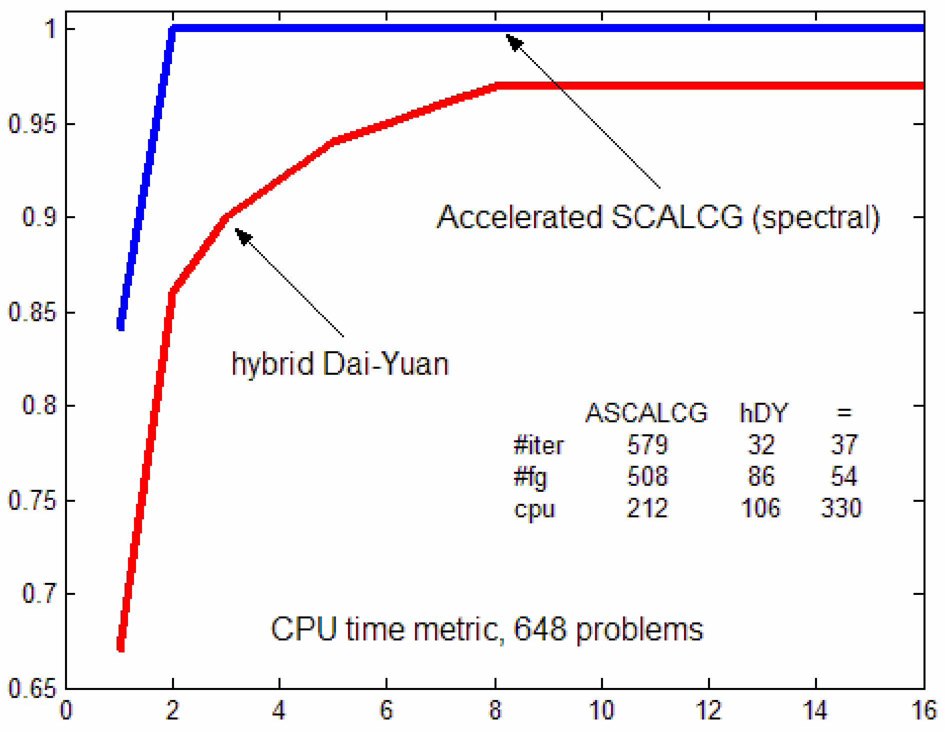

Fig. 8. ASCALCG versus Hybrid Dai-Yuan (hDY).

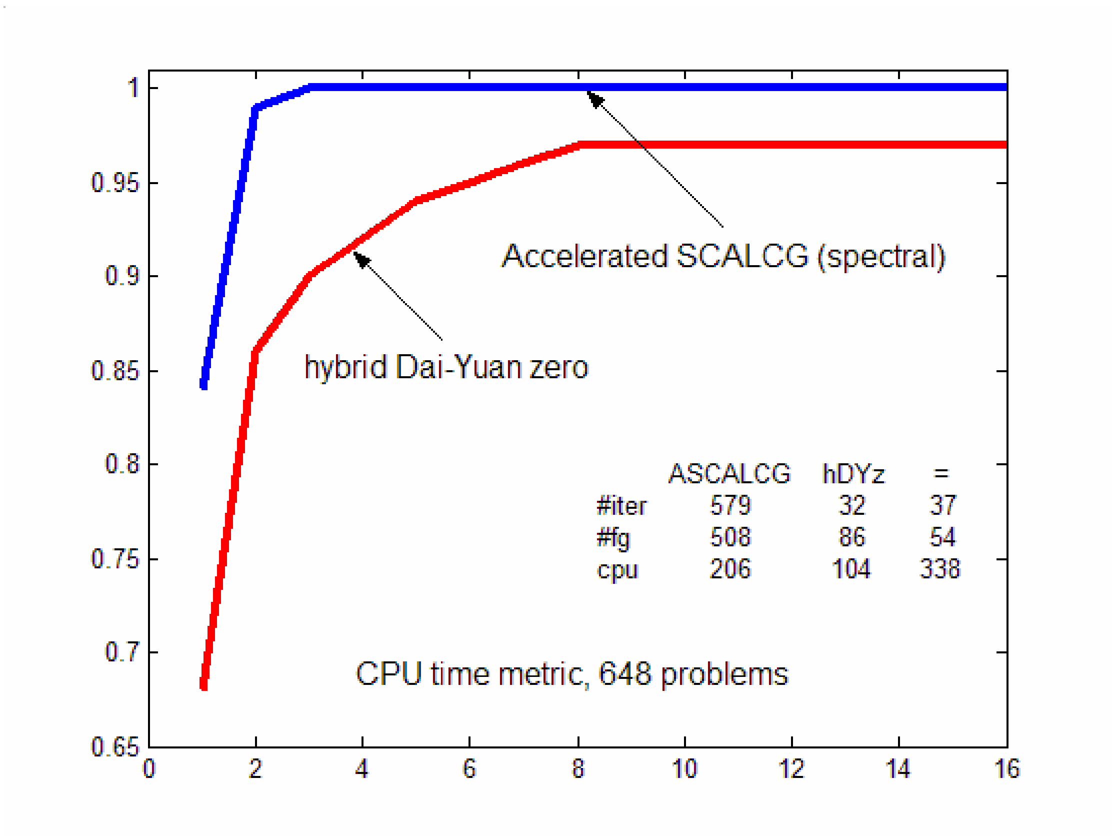

Fig. 9. ASCALCG versus Hybrid Dai-Yuan zero (hDYz).

From Figures 3-9 we see that the accelerated conjugate gradient algorithm ASCALCG is more successful and more robust than the considered classical and hybrid conjugate gradient algorithms considered in this numerical study. Hence, ASCALCG appears to generate the best search direction and the best steplength, on average. Not only ASCALCG is the fastest among these algorithms, but it is also more robust in solving a large variety of unconstrained optimization problems.

An attractive feature of the Hestenes and Stiefel conjugate gradient algorithm is that the pure conjugacy condition $y_k^T d_{k+1} = 0$ always is satisfied, independent of the line search. However, for an exact line search the convergence properties of the HS method are similar to the convergence properties of the PRP method. Therefore, by Powell's example [34], the HS method with exact line search may not converge for a general nonlinear function. Both the HS and PRP methods possess a built-in restart feature that addresses directly to the jamming phenomenon. When the step $x_{k+1} - x_k$ is small, the factor $y_k = g_{k+1} - g_k$ in the numerator of β_k tends to zero. Therefore, β_k becomes small and the new search direction d_{k+1} essentially becomes the steepest descent direction $-g_{k+1}$. Hence, both HS and PRP methods automatically adjust β_k to avoid jamming. The performance of these methods is better than the performance of DY. On the other hand, the DY method always generates descent directions, and in [13] Dai established a remarkable property for the DY conjugate gradient algorithm, relating the descent directions to the sufficient descent condition. It is shown that if there exist constants λ_1 and λ_2 such that $\lambda_1 \le \|g_k\| \le \lambda_2$ for all k, then for any $p \in (0,1)$, there exists a constant $c > 0$ such that the sufficient descent condition $g_i^T d_i \le -c\|g_i\|^2$ holds for at least $\lfloor pk \rfloor$ indices $i \in [0,k]$, where $\lfloor j \rfloor$ denotes the largest integer $\le j$. However, the DY method does not satisfy the conjugacy condition. The hDY method reduces to the Fletcher and Reeves method [16] if f is a strictly convex quadratic function and the line search is exact. For a standard Wolfe line search, Dai and Yuan [17] proved that it produces descent directions at every iteration and they established the global convergence of their hybrid conjugate gradient algorithm when the Lipschitz assumption holds.

In the fourth set of numerical experiments we compare ASCALCG to CG_DESCENT with Wolfe line search. In Figure 10 the Dolan-Moré CPU performance profiles of ASCALCG and CG_DESCENT are presented.

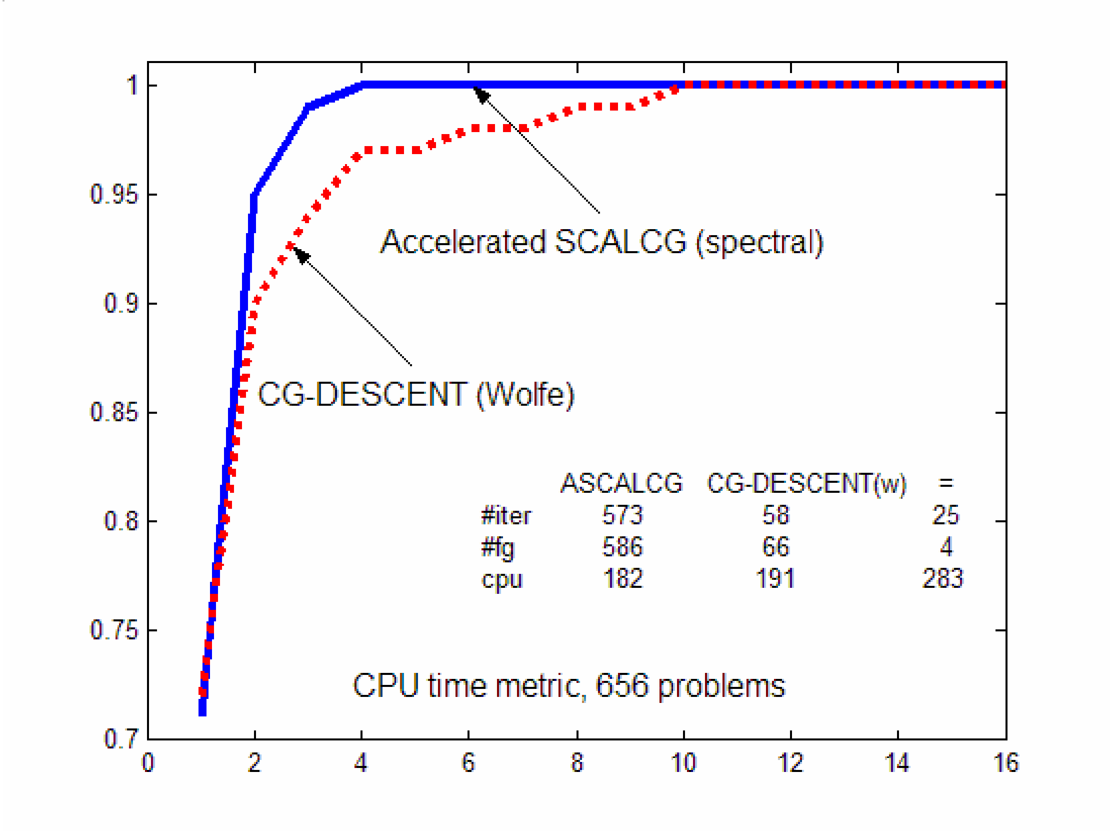

Fig. 10. ASCALCG versus CG_DESCENT (Wolfe conditions).

The top solid curve in Figure 10 corresponds to ASCALCG. Observe that subject to the CPU time metric, ASCALCG is more robust. Also, it is interesting to observe in Figure 10 that for $\tau = 1$, relative to the CPU time metric, CG_DESCENT is slighter better, ASCALCG is at best competitive with CG_DESCENT. However, for $\tau > 1$, ASCALCG turns out to be faster and more robust than CG_DESCENT. Presently CG_DESCENT is the practical conjugate gradient algorithm with more reputation. In this computational scheme the direction d_{k+1} is generated by the rule:

$$d_{k+1} = -g_{k+1} + \beta_k^{HZ} d_k, \quad \beta_k^{HZ} = \frac{1}{y_k^T d_k}\left(y_k - 2\frac{\|y_k\|^2}{y_k^T d_k} d_k \right)^T g_{k+1}. \tag{6.2}$$

This scheme is obtained by deleting a term from the search direction for the memoryless quasi-Newton scheme of Perry [31] and Shanno [40]. Observe that CG_DESCENT is a modification of HS and was devised in order to ensure sufficient descent, independent of the accuracy of the line search. The ASCALCG and CG_DESCENT algorithms (and codes) differ in many respects. Although the update formula (2.9) is more complicated than (6.2), this computational scheme proved to be efficient and more robust in numerical experiments. However, since each of these codes are different in the number of parameters which can be modified by the user to establish a context of optimization (CG_DESCENT has 26 parameters while ASCALCG has only 10 parameters) and in the amount of linear algebra required in each iteration, it is quite clear that different codes will be superior in different problem sets.

In the fifth set of numerical experiments in Figures 11 and 12 the Dolan-Moré performance profiles of ASCALCG versus LBFGS (m=3) and LBFGS (m=5) are presented.

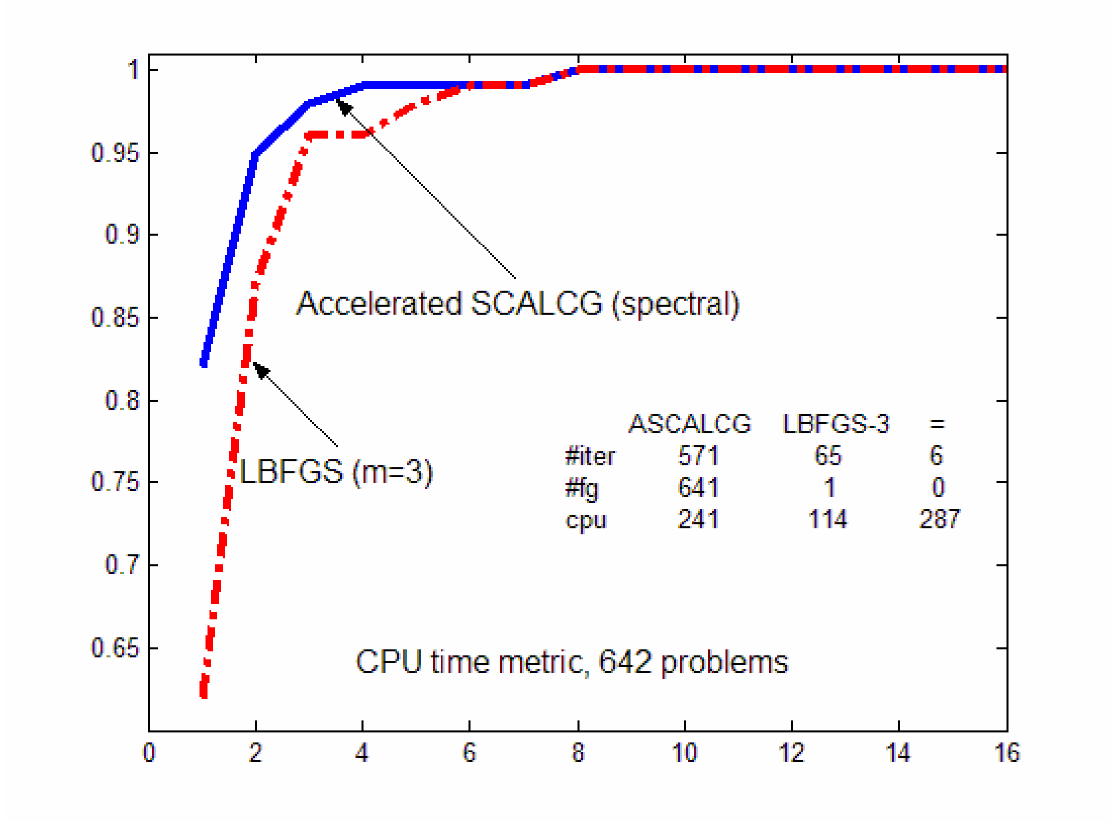

Fig. 11. ASCALCG versus LBFGS (m=3).

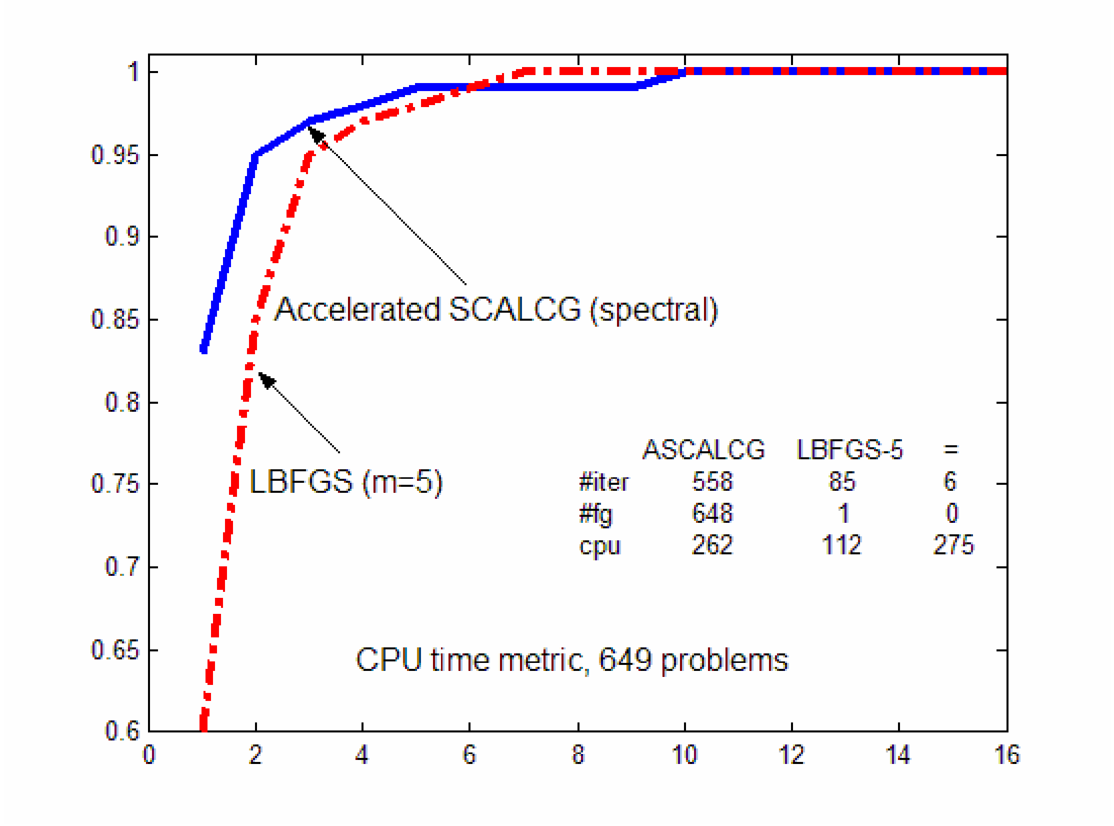

Fig. 12. ASCALCG versus LBFGS (m=5).

Finally, Figures 13 illustrates the performance profiles of ASCALCG versus the truncated Newton TN algorithm by Nash [27].

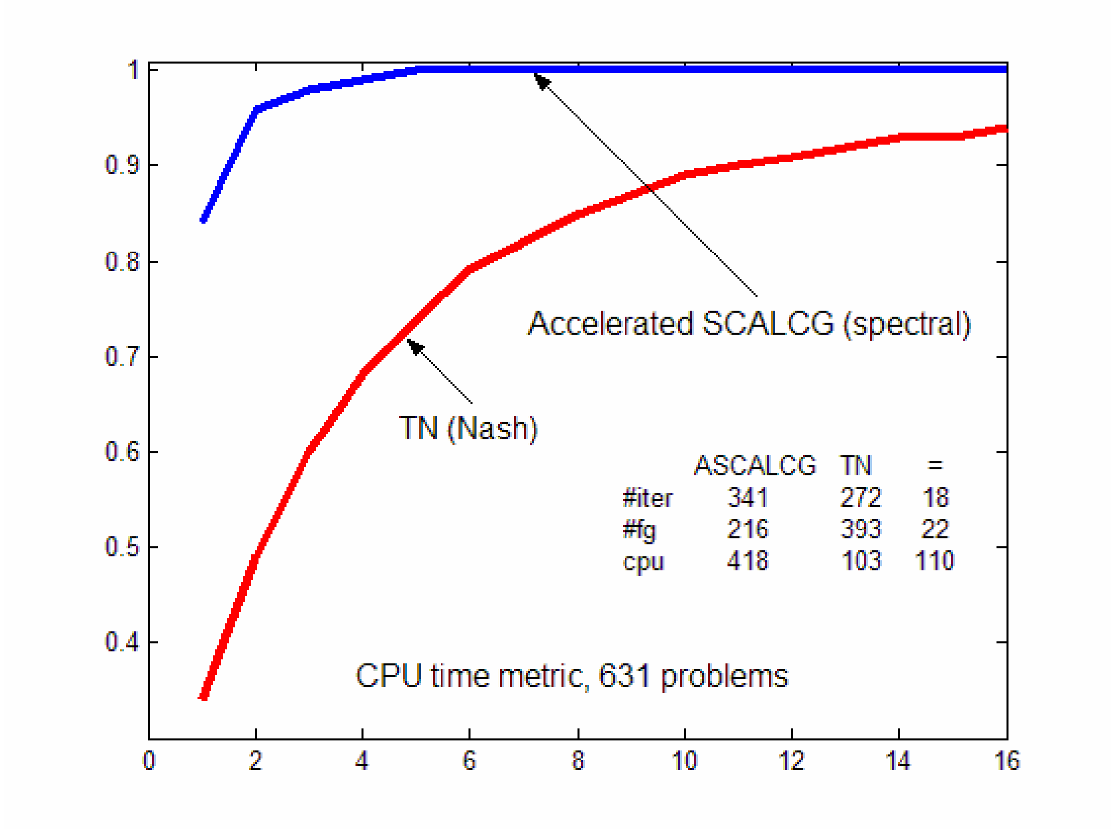

Fig. 13. ASCALCG versus truncated Newton TN (Nash).

From Figures 11 and 12 again we see that the best performance is obtained by ASCALCG. LBFGS is a professional implementation of the limited memory quasi-Newton method [26], where m is the number of the stored pairs (s_k, y_k). Even that LBFGS admits unit step lengths for the most of the iterations, thus requiring only few function and gradient evaluations for steplength determination, ASCALCG combines in a more adequate way the direction determination and step length to be the top performer.
It is interesting to observe in Figure 13 that TN by Nash [27] is completely outperformed by ASCALCG, at least for this set of unconstrained optimization test functions.

7. Conclusion

We have presented a new conjugate gradient algorithm which mainly is an acceleration of SCALCG – scaled BFGS preconditioned conjugate gradient algorithm [3-6]. The acceleration scheme is simple and proved to be robust in numerical experiments. In very mild conditions we proved that the algorithm is globally convergent. For uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in the function values is significantly improved. For a set of 750 test unconstrained optimization problems (some from CUTE library) with dimensions ranging between 1000 and 10000 variables, the CPU time performance profile for ASCALCG was higher than those of SCALCG, CONMIN, HS, PRP, DY, DL (t=1), CGSD, hDY, hDYz, CG_DESCENT, LBFGS and TN.

References

1. Andrei, N., *An unconstrained optimization test functions collection.* Advanced Modeling and Optimization. An Electronic International Journal, **10,** 147-161 (2008)
2. Andrei, N., *An acceleration of gradient descent algorithm with backtracking for unconstrained optimization*, Numerical Algorithms **42**, 63-73, (2006)

3. Andrei, N., *Scaled conjugate gradient algorithms for unconstrained optimization*. Computational Optimization and Applications **38**, 401-416 (2007)
4. Andrei, N., *Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization.* Optimization Methods and Software **22**, 561-571 (2007)
5. Andrei, N., *A scaled BFGS preconditioned conjugate gradient algorithm for unconstrained optimization*. Applied Mathematics Letters **20**, 645-650 (2007)
6. Andrei, N., *A scaled nonlinear conjugate gradient algorithm for unconstrained optimization.* Optimization. A journal of mathematical programming and operations research. **57,** 549-570 (2008).
7. Andrei, N., *A Dai-Yuan conjugate gradient algorithm with sufficient descent and conjugacy conditions for unconstrained optimization.* Applied Mathematics Letters **21**, 165-171 (2008)
8. Andrei, N., *Numerical comparison of conjugate gradient algorithms for unconstrained optimization.* Studies in Informatics and Control **16**, 333-352 (2007)
9. Andrei, N., *Acceleration of conjugate gradient algorithms for unconstrained optimization.* Applied Mathematics and Computation, **213**, 361-369 (2009).
10. Andrei, N., *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization.* Numerical Algorithms, DOI 10.1007/s11075 009-9321-0
11. Birgin, E. and Martínez, J.M., *A spectral conjugate gradient method for unconstrained optimization*, Applied Math. and Optimization **43**, 117-128 (2001)
12. Bongartz, I, Conn, A.R., Gould, N.I.M. and Toint, P.L., *CUTE: constrained and unconstrained testing environments*, ACM Trans. Math. Software **21**, 123-160 (1995)
13. Y.H. Dai, *New properties of a nonlinear conjugate gradient method.* Numer. Math., 89 (2001), pp.83-98.
14. Dai, Y.H. and Liao, L.Z., *New conjugate conditions and related nonlinear conjugate gradient methods*, Appl. Math. Optim. **43**, 87-101 (2001)
15. Dai, Y.H. and Yuan, Y., *Convergence properties of the Beale-Powell restart algorithm*, Science in China (series A) **41**, 1142-1150 (1998)
16. Dai, Y.H. and Yuan, Y., *Global convergence of the method of shortest residuals.* Numerische Mathematik, 83, 581-598 (1999)
17. Dai, Y.H. and Yuan, Y., *An efficient hybrid conjugate gradient method for unconstrained optimization*, Ann. Oper. Res. **103**, 33-47 (2001)
18. Daniel, J.W., *The conjugate gradient method for linear and nonlinear operator equations.* SIAM J. Numer. Anal. **4**, 10-26 (1967)
19. Dolan, E.D. and Moré, J.J., *Benchmarking optimization software with performance profiles*, Math. Programming **91**, 201-213 (2002)
20. Fletcher, R. and Reeves, C.M., *Function minimization by conjugate gradients* Comput. J. **7**, 149-154 (1964)
21. Gilbert, J. Ch. and Nocedal, J., *Global convergence properties of conjugate gradient methods for optimization.* SIAM J. Optimization, **2**, 21-42 (1992)
22. Hager, W.W. and Zhang, H., *A new conjugate gradient method with guaranteed descent and an efficient line search*, SIAM Journal on Optimization **16**, 170-192 (2005)
23. Hager, W.W. and Zhang, H., *Algorithm 851: CG-DESCENT, A conjugate gradient method with guaranteed descent* ACM Trans. Math. Software **32**, 113-137 (2006)
24. Hager, W.W. and Zhang, H., *A survey of nonlinear conjugate gradient methods*. Pacific Journal of Optimization, **2**, pp.35-58 (2006)
25. Hestenes, M.R. and Stiefel, E., *Methods of conjugate gradients for solving linear systems*, J. Research Nat. Bur. Standards Sec. B. **48**, 409-436 (1952)
26. Liu, D.C. and Nocedal, J., *On the limited memory BFGS method for large scale optimization methods.* Mathematical Programming **45**, 503-528 (1989)
27. Nash, S.G., *Preconditioning of truncated-Newton methods.* SIAM J. on Scientific and Statistical Computing **6**, 599-616 (1985)

28. Nocedal, J., *Conjugate gradient methods and nonlinear optimization*. In Linear and nonlinear Conjugate Gradient related methods, L. Adams and J.L. Nazareth (eds.), SIAM, 9-23 (1996)
29. Oren, S.S. and Luenberger, D.G., *Self-scaling variable metric algorithm. Part I*, Management Sci. **20**, 845-862 (1976)
30. Oren, S.S. and Spedicato, E., *Optimal conditioning of self-scaling variable metric algorithms*, Math. Programming **10**, 70-90 (1976)
31. Perry, J.M., 1977, *A class of conjugate gradient algorithms with a two step variable metric memory*, Discussion paper 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University (1977)
32. Polak E. and Ribière, G., *Note sur la convergence de methods de directions conjugres*, Revue Francaise Informat. Reserche Opérationnelle **16**, 35-43 (1969)
33. Polyak, B.T., *The conjugate gradient method in extreme problems*. USSR Comp. Math. Math. Phys. **9**, 94-112 (1969)
34. Powell, M.J.D., *Some convergence properties of the conjugate gradient method*. Math. Programming **11**, 42-49 (1976)
35. Powell, M.J.D., *Restart procedures for the conjugate gradient method*, Math. Programming **12**, 241-254 (1977)
36. Pytlak, R., *On the convergence of conjugate gradient algorithms*. IMA J. Numerical Analysis, **14**, 443-460 (1994)
37. Pytlak, R., *Conjugate gradient algorithms in nonconvex optimization.* Springer, Heidelberg, 2009.
38. Raydan, M., *The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*, SIAM J. Optim. **7**, 26-33 (1997)
39. Shanno, D.F., *Conditioning of quasi-Newton methods for function minimization.* Math. Computation, **24**, 647-657 (1970)
40. Shanno, D.F., *Conjugate gradient methods with inexact searches*, Mathematics of Operations Research **3**, 244-256 (1978)
41. Shanno, D.F., *On the convergence of a new conjugate gradient algorithm*, SIAM J. Numer. Anal. **15**, 1247-1257 (1978)
42. Shanno, D.F. and Phua, K.H., *Algorithm 500, Minimization of unconstrained multivariate functions*, ACM Trans. on Math. Soft. **2**, 87-94 (1976)
43. Shanno, D.F. and Phua, K.H., Matrix conditioning and nonlinear optimization. Mathematical Programming, **14**, 149-160, (1978)
44. Wolfe, P., *Convergence conditions for ascent methods*, SIAM Rev. **11**, 226-235 (1969)
45. Wolfe, P., *Convergence conditions for ascent methods II: some corrections*, SIAM Rev. **13**, 185-188 (1971)
46. Zhang, J.Z., Deng, N.Y. and Chen, L.H., *New quasi-Newton equation and related methods for unconstrained optimization.* J. Optim. Theory Appl. **102**, 147-167 (1999)

July 31, 2009