# A numerical study on efficiency and robustness of some conjugate gradient algorithms for large-scale unconstrained optimization

**Neculai Andrei**[1]

**Abstract**. A numerical evaluation and comparisons using performance profiles of some representative conjugate gradient algorithms for solving a large variety of large-scale unconstrained optimization problems are carried on. In this intensive numerical study we selected eight known conjugate gradient algorithms: Hestenes and Stiefel (HS), Polak-Ribière-Polyak (PRP), CONMIN, ASCALCG, CG-DESCENT, AHYBRIDM, THREECG and DESCON. These algorithms are different in many respects. However, they have a lot of concepts in common, which give the numerical comparisons sense and confident expectations. The initial search direction in all algorithms is the negative gradient computed in the initial point and the step length is computed by the Wolfe line search conditions. Excepting CONMIN and CG-DESCENT, all the algorithms from this numerical study implement an acceleration scheme which modifies the step length in a multiplicative manner to improve the reduction of the functions values along the iterations. The numerical study is based on a set of 800 artificially large-scale unconstrained optimization test functions of different complexity and with different structures of their Hessian matrix. A detailed numerical evaluation based on performance profiles is applied to the comparisons of these algorithms showing that all of them are able to solve difficult large-scale unconstrained optimization problems. However, comparisons using only artificially test problems are weak and dependent by arbitrary choices concerning the stopping criteria of the algorithms and on decision of whether an algorithm found a solution or not. To get definitive conclusions using this sort of comparisons based only on artificially test problems is an illusion. However, using some real unconstrained optimization applications we can get a more confident conclusion about the efficiency and robustness of optimization algorithms considered in this numerical study.

**Key words:** Large scale unconstrained optimization. Conjugate gradient algorithms. Numerical comparisons. Benchmarking, Performance profiles, Data profiles, MINPACK-2 applications.

## 1. Introduction

Conjugate gradient method represents an important computational innovation for continuously differentiable large scale unconstrained optimization with strong local and global convergence properties and very modest and predictable memory requirements. This family of algorithms includes a lot of variants and extensions with important convergence properties and numerical efficiency. Different from the Newton or quasi-Newton methods (including here the limited-memory quasi-Newton methods), the descent condition plays a crucial role in convergence of the conjugate gradient algorithms. As a characteristic the searching directions in conjugate gradient algorithms are selected in such a way that, when applied to minimize a strongly quadratic convex function, two successive directions are conjugate, subject to the Hessian of the quadratic function. Therefore, to minimize a convex quadratic function in a subspace spanned by a set of mutually conjugate directions is equivalent to minimize this function along each conjugate direction in turn. This is a very good and productive idea, leading us to many variants of conjugate gradient algorithms, but the performance of these algorithms is strongly dependent on the accuracy of the line search.

For solving the nonlinear unconstrained optimization problem:

$$\min_{x\in R^n} f(x), \tag{1}$$

[1] Research Institute for Informatics, Center for Advanced Modeling and Optimization, 8-10, Averescu Avenue, Bucharest 1, Romania. E-mail: nandrei@ici.ro
Dr. Neculai Andrei is member of Academy of Romanian Scientists, Splaiul Independenței Nr. 54, Sector 5, Bucharest, Romania.

where $f : R^n \rightarrow R$ is a continuously differentiable function, bounded from below, starting from an initial guess $x_0$, a nonlinear conjugate gradient algorithm generates a sequence of points $\{x_k\}$, according to the following recurrence formula

$$x_{k+1} = x_k + \alpha_k d_k , \tag{2}$$

where $\alpha_k$ is the step length, usually obtained by Wolfe line search [51, 52],

$$f(x_k + \alpha_k d_k) - f(x_k) \le \rho \alpha_k g_k^T d_k , \tag{3a}$$

$$g_{k+1}^T d_k \ge \sigma g_k^T d_k , \tag{3b}$$

with $0 < \rho < 1/2 \le \sigma < 1,$ and the directions $d_k$ are computed as:

$$d_{k+1} = -g_{k+1} + \beta_k s_k , \quad d_0 = -g_0 . \tag{4}$$

Here $\beta_k$ is a scalar known as the *conjugate gradient parameter*, $g_k = \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$. In the following $y_k = g_{k+1} - g_k$. Even that the conjugate gradient algorithms correspond to different choices for the parameter $\beta_k$, often they are designed in a specific manner in such a way that the search direction $d_k$ satisfies the sufficient descent condition $g_k^T d_k \le -c \|g_k\|^2$, for some arbitrary positive constant $c > 0$. In these algorithms, the conjugacy condition, or the modified conjugacy condition, is $d_{k+1}^T y_k = 0,$ or $d_{k+1}^T y_k = -t(g_{k+1}^T s_k),$ where $t \ge 0$ is a scalar. When applied to general nonlinear functions, often, the searching directions in conjugate gradient algorithms are computed using some formulas which do not satisfy the conjugacy condition. However, by extension we call they conjugate gradient algorithms.

The elaboration of nonlinear optimization software using nonlinear conjugate gradient algorithms is a very active field of research. On one hand, many conjugate gradient algorithms have achieved a maturity stage and are frequently used for solving a wide range of real applied problems in a large variety of areas. On the other hand, plenty of conjugate gradient algorithms are continuously elaborated and therefore their efficiency and robustness need to be established. The development of different versions of nonlinear conjugate gradient algorithms can be presented as follows. *Classical conjugate gradient algorithms*: Hestenes and Stiefel [33], Fletcher and Reeves [26], Daniel [21], Polak and Ribière [42] and Polyak [43], conjugate descent by Fletcher [27], Liu and Storey [36] and Day and Yuan [22]. *Hybrid conjugate gradient algorithms using projections*: hybrid Dai-Yuan [23], Gilbert and Nocedal [28], Hu and Storey [34], Touati-Ahmed and Storey [50], hybrid Liu and Storey [36], and *hybrid conjugate gradient algorithms using the concept of convex combination of classical schemes*: convex combination of Hestenes-Stiefel and Dai-Yuan with Newton direction [3, 4, 8], convex combination of Polak-Ribière-Polyak and Dai-Yuan with conjugacy condition [7]. *Scaled BFGS preconditioned conjugate gradient algorithms* by Shanno [47, 48], Birgin and Martínez [18] and Andrei [2, 9]. *Conjugate gradient algorithms with guaranteed descent and guaranteed conjugacy conditions* by Hager and Zhang [32] and Andrei [12]. *Three-term conjugate gradient algorithms* [10, 11].

The purpose of this paper is to study the performance of some conjugate gradient algorithms in a controlled numerical environment to highlight the main differences among them and to indicate the developer of algorithms and practitioner the best algorithms and the types of problems that are well suited to each algorithm. Therefore, we are interested to see the *efficiency* and *robustness* of some conjugate gradient algorithms for solving a large class of large-scale unconstrained optimization problems. For this purposes from the above classes of algorithms we selected a number of eight conjugate gradient algorithms, which seem to be the most representative: Hestenes and Stiefel, (HS) [33], Polak-Ribière-Polyak (PRP) [42, 43], CONMIN [47-49], ASCALCG [2, 9], CG-DESCENT [32], AHYBRIDM [3, 8], THREECG [10] and DESCON [12]. For a numerical evaluation of these algorithms the performance profiles [24] or the data profiles [37] are now standards for presenting efficiency and robustness as well as the numerical comparisons. Besides, the collection of unconstrained optimization test problems used in evaluation may have a great influence on the conclusions of the numerical study of these algorithms. In order to see the performances of these algorithms we assembled a collection of 800 large-scale unconstrained optimization test problems of a large variety and of different complexity and different structures of their Hessian matrix. The comparisons among algorithms are presented using the performance profiles. Besides, a number of five applications from MINPACK-2 collection [14] have been used to see the performances of the conjugate gradient algorithms considered in this numerical study. Our study is limited by this collection of test problems we used. However, we have tried to consider a test set of considerable diversity. Some of the artificially test problems are quadratic or nearly quadratic, while others are cubic or cubic perturbed with quadratic and linear. Some are combinations of quadratics including *exp*, *sin* or *cos* functions. There are varying degrees of nonlinearity and ill-conditioning. The functions are expressed in extended or generalized form as a sum or difference of element functions [31]. It is worth saying that the Hessian of the functions from this collection has different structure: diagonal, block-diagonal, tri-diagonal or penta-diagonal, bounded-diagonal, bounded-block-diagonal, etc. or full Hessian. The numerical conclusions concerning the efficiency and robustness of algorithms are based on this sample of functions, but we hope that they may be more generally useful for both the developer of algorithms for unconstrained optimization or practitioners faced with solving practical applications.

All these eight Fortran codes, which implement the conjugate gradient algorithms considered in this numerical study, are not new. The oldest is CONMIN, the 1978 version written by Shanno and Phua [49]. CG-DESCENT is version 1.4, (2005) written by Hager and Zhang [32]. The most recent are ASCALCG (2010) [9], AHYBRIDM (2010) [8], THREECG (2013) [10] and DESCON (2013) [12], all written by Andrei. In our numerical experiments we do not try to tune the algorithms to a particular set of test problems, and a single fixed version of each algorithm with fixed parameters was used.

As a general conclusion of this numerical study we can indicate that the conjugate gradient software analyzed in this numerical study is able to solve a very large diversity of unconstrained optimization problems of different complexity and with different structures of the Hessian matrix. At least for this set of artificially test problems, concerning the efficiency, CG-DESCENT is slightly more efficient, followed by DESCON and followed by THREECG. Subject to robustness by far DESCON is the most robust, followed by THREECG and followed by ASCALCG. It seems that the conjugate gradient algorithms implementing both the sufficient descent condition and the conjugacy condition are the best. However, this is not a definitive conclusion. In front of us there are an infinite number of artificially unconstrained optimization test problems and it is always possible to assemble a set of problems for which the efficiency and robustness of the considered algorithms are completely different. However, in order to have a true conclusion at all we compared the above algorithms on five applications from MINPACK-2 collection with $10^6$ variables. In this case DESCON proved to be the fastest and the most reliable algorithm.

The structure of the paper is as follows. In section 2 the main characteristics of unconstrained optimization test problems considered in this numerical study are presented. A detailed presentation of the comparison framework including the performance profiles and the data profiles, their advantages and weakness, and the efficiency and the robustness of an algorithm is given in section 3. In section 4 we present the conjugate gradient algorithms considered in this numerical study insisting on their definition and convergence properties. Section 5 is devoted to present the numerical experiments and comparisons using the performance profiles. In section 6 some discussions are given including some comparisons among algorithms for solving problems with different structures of the Hessian, the weakness of the numerical experiments and comparisons using artificially test problems and some results and comparisons for solving five MINPACK-2 applications. Conclusions are drawn in the last section.

## 2. Unconstrained optimization test problems considered in this numerical study

In this numerical study, we have considered 80 large-scale unconstrained optimization test functions, in extended or generalized form we presented in [5], some of them being taken from Cuter collection [19]. Each problem was tested 10 times for a gradually increasing number of variables: $n = 1000, 2000, \ldots, 10000$. Therefore, we obtained a set of 800 unconstrained optimization test problems of different complexity and with different structures of their Hessian. The problems considered in this numerical study are in generalized or extended form as a sum or difference of element functions [31] of different nonlinear complexity. The structure of the Hessian matrix of the generalized functions is tri-diagonal or multi-diagonal. The structure of the Hessian matrix of the extended function is block-diagonal. Some functions are highly nonlinear and ill-conditioned.

In [38] Nash and Nocedal suggested some criteria to classify the test problems used in numerical studies. For the various function characteristics that are relevant to the convergence theory or computational performances of algorithms they selected the following criteria: deviation from quadratic (degree of nonlinearity), condition number of the Hessian, convexity, eigenvalue structure, cost of evaluating the function and its gradient, etc. None of these criteria is operational for large-scale unconstrained optimization. For example, the deviation from quadratic involves the computation of the Hessian which is a very difficult task for functions with a large number of variables. Probably, the most important criterion is the eigenvalue structure of the Hessian. The eigenvalue distribution greatly affects the performance of conjugate gradient algorithms. However, in case of large-scale optimization computation of eigenvalue structure of the Hessian is not tractable. Also, convexity, an important concept in optimization, is difficult to be established. Hence, we do not classify the problems according to these criteria because we believe that they are not relevant for our purpose and, besides, there is not a clear conclusion concerning the performances of algorithms subject to the criteria considered in [38].

However, we can classify the problems according to the structure of their Hessian. Knowing the analytical expression of the gradient it is very easy to get the structure of the Hessian. In this numerical study, out of 80 functions for 10 of them the Hessian is a diagonal matrix, for 19 the Hessian is a block-diagonal matrix, for 22 the Hessian is tri-diagonal (or penta-diagonal) and finally for 16 of them the Hessian is a full matrix. Therefore, in the last section of the paper we present some comments about the performances of the above conjugate gradient algorithms for solving problems with different structures of the Hessian.

## 3. Comparison framework

### 3.1. *Performance profiles versus data profiles*

Both performance profiles [24] and data profiles [37] are common standards for presenting the numerical comparisons among algorithms. In the following we shall present them insisting both on their importance and the main differences. Let us consider a number $m$ of methods $M_1, \ldots, M_m$ used for solving $p$ problems $P_1, \ldots P_p$ and let $t_{ij}$ be a metric representing the effort method $M_i$ made for solving problem $P_j$ in order to get a point in which the value of the function is $f_{ij}$. We assume that the metric $t_{ij}$ is such that the smaller its value, the higher the performance of the method $M_i$ for solving the problem $P_j$. Consider $t_j^{\min}$ the smallest value among all the $t_{ij}$ required by each method $M_i$ that get a solution for problem $P_j$. With these elements let us define the performance profile of method $M_i$ as:

$$\Gamma_i(\tau) = \frac{\#\left\{ j \in \{1, \ldots, p\} : M_i \text{ } \mathit{found\ a\ solution\ for}\ P_j\ \mathit{with}\ t_{ij} \le \tau t_j^{\min} \right\}}{p}, \tag{5}$$

where $\# A$ is the cardinality of the set $A$. Observe that the performance profiles, as defined in (5), represent a curve very useful for graphical representation of comparisons among several methods for solving large sets of problems. Mainly, $\Gamma_i(\tau)$ represents the fraction of problems a method $M_i$ solved within a prescribed limit on its performance measurement like, for example, the number of iterations, or the number of functions evaluation or the CPU time. The main characteristic of the performance profile is that for each problem, the imposed limit is a proportion $\tau \geq 1$ of the performance measurement of the most efficient method for solving this particular problem. Therefore, for a given method $M_i$, $\Gamma_i(\tau = 1)$ represents the fraction of problems for which the method was the most efficient over all methods. On the other hand, $\Gamma_i(\tau = \infty)$ represents the fraction of problems solved by method $M_i$, irrespective of the required effort. In this context, $\Gamma_i(\tau = 1)$ is associated to the *efficiency* of method $M_i$, while $\Gamma_i(\tau = \infty)$ is associated to the *robustness* of method $M_i$.

It is worth saying that the performance profile gives the *same importance* both to the problems *easy* to be solved and to the problems *hard* to be solved, where by easy we understand that the problem can be solved without a consistent effort (number of iterations, or number of functions evaluation or CPU time).

In order to make a difference between the easy problems and the hard ones the data profiles has been introduced as:

$$\overline{\Gamma}_i(\tau) = \frac{\#\{j \in \{1,\ldots,p\} : M_i \textit{ found a solution for } P_j \textit{ with } t_{ij} \leq \tau\}}{p}. \quad (6)$$

As we can see, $\overline{\Gamma}_i(\tau)$ represents the fraction of problems method $M_i$ is able to solve within a prescribed limit on its performance measurement like the number of iterations, or the number of function evaluations, or CPU time. Observe that in this case the limit is independent of the performances of the other methods considered in a numerical study.

The difference between these two profiles is major and there is not a clear answer which one from these two to prefer. However, in this paper we select the performance profiles as the main instrument for comparing the algorithms. The motivation behind this selection is that we consider the easy and the hard problem have the same importance within a set of test problems.

### 3.2. *Solving an unconstrained optimization problem and comparison framework*

In this numerical study by solving an unconstrained optimization problem we understand that the methods $M_1,\ldots,M_m$ determine *local solutions* of the problems $P_1,\ldots P_p$. For a given problem it is quite possible that two different methods determine two different local minimizers with different function values. There is a great discussion whether all these problems should be removed from the performance evaluation process or not. In our analysis all the problems for which two different methods found different function values are removed. The motivation behind this selection is that we are interested to compare algorithms which find the same function values (in a given tolerance) to see the main characteristics of the optimization processes concerning the number of iterations, the number of function and its gradient evaluations and the CPU running time.

In case of the unconstrained optimization the quality of solutions can be very simple evaluated by comparing only the values of function to be minimized. Since we are working in floating-point arithmetic we must compare two function values using relative errors, as follows. Let us consider that when the methods $M_1,\ldots,M_m$ are applied for solving a particular problem, the following function values $f_1,\ldots,f_m$ are obtained. Let $f^{\min} = \min\{f_1,\ldots,f_m\}$ and consider

$$\varepsilon_i = \frac{f_i - f^{\min}}{\max\{1, \left|f^{\min}\right|\}}, \quad i = 1,\ldots,m. \tag{7}$$

Therefore, for a given tolerance $\varepsilon^f > 0,$ we say that *the method* $M_i$ *found a solution* if

$$\varepsilon_i \le \varepsilon^f, \tag{8}$$

i.e. in comparisons when $\left|f^{\min}\right| \le 1$ we consider "*small absolute errors*", and "*small relative errors*" otherwise. It is quite clear that using relative errors in this manner, the question is the value of the threshold parameter $\varepsilon^f$. Arbitrary small or large value choices of this parameter will have some influence in the comparison of algorithms. Since we do not have any possibility to fix a "good" value for $\varepsilon^f,$ in this numerical study we compare the algorithms using the performance profiles $\Gamma(1)$ (efficiency) and $\Gamma(\infty)$ (robustness) for 6 different values of $\varepsilon^f$ : $\varepsilon^f = 10^{-3},\ldots,10^{-8}.$ From our intensive numerical experiments we observed that the value of the threshold parameter $\varepsilon^f$ does not have a great influence on the performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of the algorithms. In order to have a better understanding of the efficiency and the robustness of algorithms we present the performance profiles for $\tau = 1,\ldots,16$ and $\varepsilon^f = 10^{-3}.$

## 4. Conjugate gradient algorithms considered in this numerical study

In this work we focus on unconstrained optimization software which implements conjugate gradient algorithms. The eight solvers considered in this numerical study include: the classical conjugate gradient algorithms Hestenes and Stiefel (HS) ($\beta_k^{HS} = g_{k+1}^T y_k / y_k^T s_k$) [33] and Polak-Ribière-Polyak (PRP) ($\beta_k^{PRP} = g_{k+1}^T y_k / g_k^T g_k$) [42, 43]; the BFGS preconditioned conjugate gradient algorithms CONMIN [49] and ASCALCG [9]; a conjugate gradient algorithm with guaranteed descent CG-DESCENT [32]; a hybrid conjugate gradient algorithm as a convex combination of HS, and Day and Yuan conjugate gradient algorithms AHYBRIDM [3, 8]; a simple three-term conjugate gradient algorithm which satisfy both the descent and the conjugacy conditions THREECG [10], and a conjugate gradient algorithm for which both the descent and the conjugacy conditions are guaranteed with modified second Wolfe line search condition DESCON [12]. In this study we are interested to see the *efficiency* and the *robustness* of these algorithms and to compare their performances subject to a large class of artificially test problems and real unconstrained optimization applications.

Intensive numerical experiments proved that in conjugate gradient algorithms the step length may differ from 1 in a very unpredictable manner. They can be larger or smaller than 1 depending on how the problem is scaled. This is in very sharp contrast to the Newton and quasi-Newton algorithms, including the limited memory quasi-Newton algorithms, which accept the unit step length most of the time along the iterations, thus requiring only few function evaluations per search direction. Therefore, excepting CONMIN and CG-DESCENT, all the algorithms from this numerical study implement an acceleration scheme which modifies the step length in a multiplicative manner to improve the reduction of the functions values along the iterations [1, 6]. The initial search direction in all algorithms is $d_0 = -g_0$ and the step length is computed by the Wolfe line search conditions implemented in the same manner. The initial guess of the step length at the first iteration is $\alpha_0 = 1/\left\|g_0\right\|.$ At the following iterations, in all algorithms, the starting guess for step $\alpha_k$ is computed as $\alpha_{k-1}\left\|d_{k-1}\right\|_2 / \left\|d_k\right\|_2.$ This strategy proved to be one of the best selection of the initial guess of the step length.

The HS and PRP conjugate gradient algorithms are very well known in literature. Both of them have $g_{k+1}^T y_k$ in numerator and possess a built-in restart feature that directly addresses the jamming, which is a very important property. When the step $s_k = x_{k+1} - x_k$ is small, the factor $y_k$ in the numerator of $\beta_k^{HS}$ and $\beta_k^{PRP}$ tends to zero. Therefore, $\beta_k^{HS}$ and $\beta_k^{PRP}$ become small and the direction $d_{k+1}$ corresponding to HS or PRP algorithms is essentially the steepest descent direction $-g_{k+1}$. Hence, the HS and PRP methods automatically adjust the parameter $\beta_k$ to avoid jamming. The HS method has the property that the conjugacy condition $y_k^T d_{k+1} = 0$ always holds, independent by the line search. If $f$ is uniformly convex and the line search is exact, Polak and Ribière [42] proved the global convergence of PRP algorithm. On the other hand, for general nonlinear function Powell [45] proved the global convergence of PRP algorithm if the step size $s_k$ tends to zero, the line search is exact and the gradient is Lipschitz continuous. Using an exact line search, $\beta_k^{HS} = \beta_k^{PRP}$. Therefore, the convergence properties of the HS algorithm should be similar to the convergence properties of the PRP algorithm. But, for general nonlinear functions the convergence of the PRP method is uncertain. The classical Powell's example shows that when the function is not strongly convex, the PRP algorithm may not converge, even with an exact line search [44]. Hence, the HS algorithm with an exact line search may not converge for general nonlinear functions. However, although HS and PRP may not converge in general, they often perform better than some other conjugate gradient algorithms like Fletcher and Reeves (FR) ($\beta_k^{FR} = \left\|g_{k+1}\right\|^2 / \left\|g_k\right\|^2$) [26], Dai and Yuan (DY) ($\beta_k^{DY} = \left\|g_{k+1}\right\|^2 / y_k^T s_k$) [22] and the Fletcher's algorithm CD ($\beta_k^{CD} = \left\|g_{k+1}\right\|^2 / (-g_k^T s_k)$) [27]. The line search in both HS and PRP is based on the Wolfe conditions implemented in the same manner.

Both CONMIN and ASCALCG belong to the same class of conjugate gradient algorithms based on scaled BFGS preconditioning. CONMIN elaborated by Shanno and Phua [49] incorporates two nonlinear optimization algorithms, a conjugate gradient algorithm and a variable metric BFGS one. The conjugate gradient algorithm in CONMIN, we consider in our numerical study, is the Beale restarted memoryless BFGS updating algorithm, which in fact is a modification of Perry algorithm [41]. Shanno [47] observed that the conjugate gradient algorithms are exactly the BFGS quasi-Newton algorithm where the approximation to the inverse Hessian is restarted as the scaled identity matrix at every step, as no additional storage is used to develop a better approximation to the inverse Hessian. The scaling factor is computed as $y_k^T d_k / \left\|y_k\right\|^2$. The algorithm implemented in CONMIN is a composite conjugate gradient algorithm in which the same philosophy used in BFGS of modifying the negative gradient with a positive definite matrix which best estimates the inverse Hessian without adding anything to storage requirements [47] is implemented at restarting, i.e. when the Beale-Powell restart criterion is satisfied. The linear search uses Davidon's cubic interpolation to find a step-length satisfying the Wolfe line search conditions. Shanno [47] proved that under the assumptions on $f(x)$ that $u^T \nabla^2 f(x) u \le M \left\|u\right\|^2$, $M > 0$ and $f(x) \ge L$ uniformly in $x$, either $\lim_{k\to\infty} \left\|g_k\right\| = 0$ or $\lim_{k\to\infty} \left\|s_k\right\| > 0$. Under the further assumption that for any $R > 0$ the level set $S = \{x : f(x) \le R\}$ is bounded, then the sequence $\{x_k\}$ generated by the algorithm converges to a point $x^*$ at which $\left\|g(x^*)\right\| = 0$, or the sequence cycles.

On the other hand, ASCALCG elaborated by Andrei [9] is an accelerated scaled conjugate gradient algorithm which combines the scaled memoryless BFGS algorithm and the preconditioning technique. The preconditioner, which is also a scaled memoryless BFGS matrix, is reset when the Beale-Powell restart criterion [16] holds. The parameter scaling the gradient is selected as a spectral gradient: $\theta_{k+1} = s_k^T s_k / y_k^T s_k$. The search direction in ASCALCG is computed as a double quasi-Newton update scheme:

$$d_{k+1} = -v + \frac{(g_{k+1}^T s_k)w + (g_{k+1}^T w)s_k}{y_k^T s_k} - \left(1 + \frac{y_k^T w}{y_k^T s_k}\right)\frac{g_{k+1}^T s_k}{y_k^T s_k} s_k, \tag{9}$$

where $v = H_{r+1}g_{k+1}$ and $w = H_{r+1}y_k$ and $H_{r+1}$ is the BFGS approximation to the inverse Hessian initialized with the identity matrix and scaled by the scalar $\theta_{r+1}$ at the $r-$th iteration where the Beale-Powell restart test is satisfied:

$$H_{r+1} = \theta_{r+1}I - \theta_{r+1}\frac{y_r s_r^T + s_r y_r^T}{y_r^T s_r} + \left(1 + \theta_{r+1}\frac{y_r^T y_r}{y_r^T s_r}\right)\frac{s_r s_r^T}{y_r^T s_r}. \tag{10}$$

The restart direction is computed as $d_{k+1} = -Q_{k+1}^* g_{k+1}$, where $Q_{k+1}^*$ is exactly the BFGS quasi-Newton matrix, and at every step the approximation of the inverse Hessian is the identity matrix multiplied by the scalar $\theta_{k+1}$, i.e.

$$d_{k+1} = -\theta_{k+1}g_{k+1} + \theta_{k+1}\left(\frac{g_{k+1}^T s_k}{y_k^T s_k}\right)y_k - \left[\left(1 + \theta_{k+1}\frac{y_k^T y_k}{y_k^T s_k}\right)\frac{g_{k+1}^T s_k}{y_k^T s_k} - \theta_{k+1}\frac{g_{k+1}^T y_k}{y_k^T s_k}\right]s_k. \tag{11}$$

For the step-length computation the algorithm implements the Wolfe line search conditions in the same manner as in CONMIN. The global convergence of ASCALCG was established by Babaie-Kafaki [15]. For uniformly convex function if the gradient of the function $f$ is Lipschitz continuous on the level set $S = \{x : f(x) \le f(x_0)\}$, then the search directions generated by ASCALCG satisfy the sufficient descent condition. For general nonlinear functions under exact line search and if the gradient of the function $f$ is Lipschitz continuous, then the algorithm satisfies the sufficient descent condition, i.e. it is globally convergent.

CG-DESCENT algorithm was elaborated by Hager and Zhang [32] in order to ensure sufficient descent, independent of the accuracy of the line search. In CG_DESCENT the search direction $d_{k+1} = -g_{k+1} + \beta_k^{HZ} s_k$, where

$$\beta_k^{HZ} = \left(y_k - 2\frac{\|y_k\|^2}{y_k^T s_k} s_k\right)^T \frac{g_{k+1}}{y_k^T s_k}, \tag{12}$$

satisfies the sufficient descent condition $g_k^T d_k \le -(7/8)\|g_k\|^2$. If the function $f$ is a quadratic and the line search is exact, then CG_DESCENT reduces to HS. In fact, CG_DESCENT is a modification of HS algorithm. However, in CG_DESCENT the search directions do not satisfy the conjugacy condition. Again, when iterates jam the expression $(\|y_k\|^2 (s_k^T g_{k+1}))/(y_k^T s_k)^2$ in the above formulation of $\beta_k^{HZ}$ becomes negligible. This modification of the HS scheme makes CG_DESCENT to perform better than HS [32]. In order to obtain global convergence for general nonlinear functions, the algorithm truncates the parameter $\beta_k^{HZ}$ in a manner similar to PRP+ [28]. It is not clearly known that this truncation mechanism is benefic in the economy of the algorithm. Under standard assumptions the algorithm that satisfies the Wolfe line search is convergent in the sense that either $g_k = 0$ for some $k$ or $\liminf_{k\to\infty}\|g_k\| = 0$ [32]. A very simple restart scheme is implemented in CG-DESCENT: when the number of iterations is a multiple of $n$, then the searching direction is reset to the negative gradient. However, for the vast majority of problems the number of iterations for solving a problem is much smaller that its dimension. Therefore, the restart iterations are very seldom used.

AHYBRIDM, elaborated by Andrei [3, 8], is an accelerated hybrid conjugate gradient algorithm in which the parameter $\beta_k$ is computed as a convex combination of $\beta_k^{HS}$ and $\beta_k^{DY}$ where the parameter $\theta_k$ in the convex combination is computed in such a way the direction corresponding to the conjugate gradient algorithm is the best direction we know, i.e. the Newton direction, while the pair $(s_k, y_k)$ satisfies the modified secant given by Li et al. [35] $B_{k+1}s_k = z_k$, where $z_k = y_k + (\eta_k / \|s_k\|^2)s_k$ and $\eta_k = 2(f_k - f_{k+1}) + (g_k + g_{k+1})^T s_k$. Therefore,

$$\beta_k = (1-\theta_k)\beta_k^{HS} + \theta_k \beta_k^{DY} \tag{13}$$

where

$$\theta_k = \frac{\left(\dfrac{\delta\eta_k}{s_k^T s_k} - 1\right) s_k^T g_{k+1} - \dfrac{y_k^T g_{k+1}}{y_k^T s_k}\delta\eta_k}{g_k^T g_{k+1} + \dfrac{g_k^T g_{k+1}}{y_k^T s_k}\delta\eta_k}. \tag{14}$$

$\delta$ is a scalar parameter ($\delta = 1$ in our numerical experiments). In [8] we have the computational evidence that AHYBRIDM, as a convex combination of HS and DY conjugate gradient algorithms, is top performer versus the hybrid conjugate gradient algorithms obtained by projections, like hybrid Dai and Yuan [23] ($\beta_k^{hDY} = \max\{c\beta_k^{DY}, \min\{\beta_k^{HS}, \beta_k^{DY}\}\}$, where $c = (1-\sigma)/(1+\sigma)$), Gilbert and Nocedal [28] ($\beta_k^{GN} = \max\{-\beta_k^{FR}, \min\{\beta_k^{PRP}, \beta_k^{FR}\}\}$) or hybrid Liu and Storey [36] ($\beta_k^{LS-CD} = \max\{0, \min\{\beta_k^{LS}, \beta_k^{CD}\}\}$, $\beta_k^{LS} = -(g_{k+1}^T y_k)/(g_k^T s_k)$). This is the reason we selected AHYBRIDM in this numerical study on the efficiency and robustness of conjugate gradient algorithms. The step-length is computed using the Wolfe line search. An acceleration scheme is implemented by modifying the step-length in order to improve de reduction of function values along the iterations. Under classical assumptions, both for uniformly convex functions and for general nonlinear functions the algorithm with strong Wolfe line search is globally convergent [8].

THREECG is a simple three-term conjugate gradient algorithm developed by Andrei [10]. The algorithm is a modification of the HS algorithm or of CG-DESCENT in such a way that the search direction is descent and it satisfies the conjugacy condition. These properties are independent of the line search. Also, the algorithm could be considered as a very simple modification of the memoryless BFGS quasi-Newton method. The search direction is computed as:

$$d_{k+1} = -g_{k+1} - \delta_k s_k - \eta_k y_k, \tag{15}$$

where

$$\delta_k = \left(1 + \frac{\|y_k\|^2}{y_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T g_{k+1}}{y_k^T s_k}, \tag{16}$$

$$\eta_k = \frac{s_k^T g_{k+1}}{y_k^T s_k}. \tag{17}$$

The new approximation of the minimum is obtained by the general Wolfe line search conditions and the acceleration technique. For uniformly convex functions, under standard assumptions, the algorithm is globally convergent [10].

Finally DESCON [12] is a conjugate gradient algorithm for which both the descent and the conjugacy conditions are guaranteed. The search direction is selected as

$$d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k, \tag{18}$$

where

$$\theta_k = \frac{a_k}{y_k^T g_{k+1}}\left[1 + \frac{(y_k^T s_k)\|g_{k+1}\|^2}{\bar{\Delta}_k}\right] - \frac{b_k}{\bar{\Delta}_k}, \tag{19}$$

$$\beta_k = \frac{y_k^T g_{k+1}}{y_k^T s_k}\left(1 - \frac{b_k}{\bar{\Delta}_k}\right) + a_k \frac{\|g_{k+1}\|^2}{\bar{\Delta}_k} \tag{20}$$

and

$$a_k = v(s_k^T g_{k+1}) + y_k^T g_{k+1}, \tag{21}$$

$$b_k = w\|g_{k+1}\|^2 (y_k^T s_k) + (y_k^T g_{k+1})(s_k^T g_{k+1}), \tag{22}$$

$$\bar{\Delta}_k = (y_k^T g_{k+1})(s_k^T g_{k+1}) - \|g_{k+1}\|^2 (y_k^T s_k). \tag{23}$$

$w \geq 0$ and $v \geq 0$ are arbitrary positive constants which specify the sufficient descent condition $g_{k+1}^T d_{k+!} \leq -w\|g_{k+1}\|^2$ and the conjugacy condition $d_{k+1}^T y_k = -v(g_{k+1}^T s_k),$ respectively. The algorithm introduces the modified Wolfe line search conditions, in which the parameter in the second Wolfe condition is modified at every step as:

$$\sigma_k = \|g_{k+1}\|^2 / (|y_k^T g_{k+1}| + \|g_{k+1}\|^2). \tag{24}$$

The algorithm implements the acceleration scheme. Both for uniformly convex functions and for general nonlinear functions, the algorithm with strong Wolfe line search generates directions bounded away from infinite. Therefore, the algorithm is globally convergent [12].

HS, PRP, AHYBRIDM, THREECG and DESCON use the Beale-Powell [16, 45] restart mechanism: if $|g_{k+1}^T g_k| \geq 0.2\|g_{k+1}\|^2$ is satisfied, then $d_{k+1} = -g_{k+1}$. In our numerical experiments we noticed that for solving a problem this test is used in many iterations representing a sort of relaxation of the algorithm.

Excepting CG-DESCENT all the algorithms considered in this numerical study use exactly the same implementation of the Wolfe line search (3). This is an advanced implementation with Davidon's cubic interpolation and different safeguards to ensure that the search procedure cannot be stuck or attempt to move away past a local maximum to a more distant local minimum.

## 5. Numerical experiments and comparisons

In the present numerical experiments we considered 800 large-scale unconstrained optimization test problems of the form $\min\{f(x) : x \in R^n\}$. The stopping criterion associated with successful convergence of the algorithms, very used in large-scale optimization, was

$$\|g_k\|_\infty \leq \varepsilon^g, \tag{25}$$

where $\|.\|_\infty$ is the maximum absolute component of a vector. Concerning the threshold parameter $\varepsilon^g$ there is not a clear rule to establish its value. However, in order to achieve small values of the sup-norm of the gradient we selected $\varepsilon^g = 10^{-6}$ in (25). We see that for problems with $n = 10^4$ variables (25) implies that $\|g_k\|_2 \leq 10^{-4}$. In all algorithms considered in this numerical study, for the step length $\alpha_k$ computation, the same implementation of the Wolfe line searches conditions (3) is used, where $\rho = 10^{-4}$ and $\sigma = 0.8$. In DESCON the parameter $\sigma$ is computed at every step as in (24). At the same time, even CG-DESCENT has two procedures for the step length $\alpha_k$ computation, the classical Wolfe line search (3) and the approximate Wolfe line search, in our numerical experiments we have considered only the classical Wolfe conditions. In all cases we preserved the software's default parameters. Software was compiled with Fortran 77, option –O4. The numerical experiments were executed on a Workstation Intel Pentium 4 with 1.8 GHz. Excepting CONMIN, all algorithms use the loop unrolling of depth 5.

In all the numerical comparisons we selected to use the performance profiles [24] to present the results of the numerical experiments. This is motivated by the fact that using a large set of test problems, the easy-to-solve problems have the same importance as the harder ones. Besides, the performance profiles illustrate both the efficiency and the robustness of a method versus some other methods considered in this numerical study. The performance profiles correspond to the CPU time metric in which all the problems that do not satisfy the criterion (8) have been ignored. It is worth saying that the performance profiles refer to a comparative analysis of eight conjugate gradient algorithms using only two algorithms each time.

5.1. *DESCON versus classical conjugate gradient algorithms HS and PRP*

Considering $\varepsilon^f = 10^{-3}$ in (8), in Figures 1 and 2 we present the performance profile of DESCON versus HS and PRP subject to CPU time metric, respectively. Observe that the best performance, relative to the CPU time metric, was obtained by DESCON, the solid top curve in Figures 1 and 2. The figures indicate that relative to CPU time metric, DESCON is fastest. Since all these three codes use the same line search based on Wolfe conditions (implemented in exactly the same manner), these codes only differ in their choice of the search direction.

Comparing, for example, DESCON versus HS (see Figure 1), subject to the number of iterations, we see that DESCON was better in 610 problems (i.e. it achieved the minimum number of iterations in 610 problems). HS was better in 84 problems and they achieved the same number of iterations in 62 problems, etc. Out of 800 problems, only for 756 problems does the criterion (8) holds with $\varepsilon^f = 10^{-3}$. Therefore in comparison with HS (see Figure 1) and PRP (see Figure 2) DESCON appear to generate the best search direction on average.

Tables 1 and 2 present the efficiency and robustness rates of DESCON versus HS and of DESCON versus PRP respectively, for $\varepsilon^g = 10^{-6}$ and $\varepsilon^f \in \{10^{-3},\ldots,10^{-8}\}$. From these Tables we have the computational evidence that DESCON is the most efficient and the most robust method for every value of $\varepsilon^f$ in the set: $\{10^{-3},\ldots,10^{-8}\}$. For example, for $\varepsilon^f = 10^{-3}$, from Table 1 we see that DESCON is 8.069% more efficient than HS and 12.037% more robust. Concerning PRP, DESCON is 9.272% more efficient and 12.848% more robust. In Tables 1 and 2 Nrp is the number of problems, out of 800 used in these numerical studies that satisfy criterion (8). Again, from Tables 1 and 2 observe that the value of the threshold parameter $\varepsilon^f$ does not have a great influence on performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus HS and PRP. All these three algorithms use the Beale-Powell restart procedure. If $\left|g_{k+1}^T g_k\right| \geq 0.2\left\|g_{k+1}\right\|^2$ is satisfied, then $d_{k+1} = -g_{k+1}$, i.e. the current direction is the negative gradient. This is an important ingredient in conjugate gradient algorithms representing a sort of relaxation of iterations. It is worth seeing the restart iterations in these algorithms for a particular problem. Let us consider the problem #3 (extended Rosenbrock function). This problem was chosen because it illustrates the typical performance that we observed in numerical experiments. For $n = 1000$ to get the optimal solution, DESCON needs 57 iterations out of which 21 are restart iterations, i.e. 36.842% are restart iterations. On the other hand, HS needs 79 iterations, out of which 46 are restart iterations, i.e. 58.227% are restart iterations. In the same context, PRP needs 78 iterations, 42 from these being restart iterations, i.e. 53.846% are restart iterations that use the negative gradient as the searching direction. Observe that DESCON needs the least number of restart iterations, and this is the reason of its efficiency and robustness in comparisons with HS and PRP.

HS and PRP conjugate gradient algorithms use the Wolfe line search conditions (3) where $\rho = 10^{-4}$ and $\sigma = 0.8$. On the other hand, DESCON again use the Wolfe line search (3), but at every step modifies $\sigma_k$ as in (24).

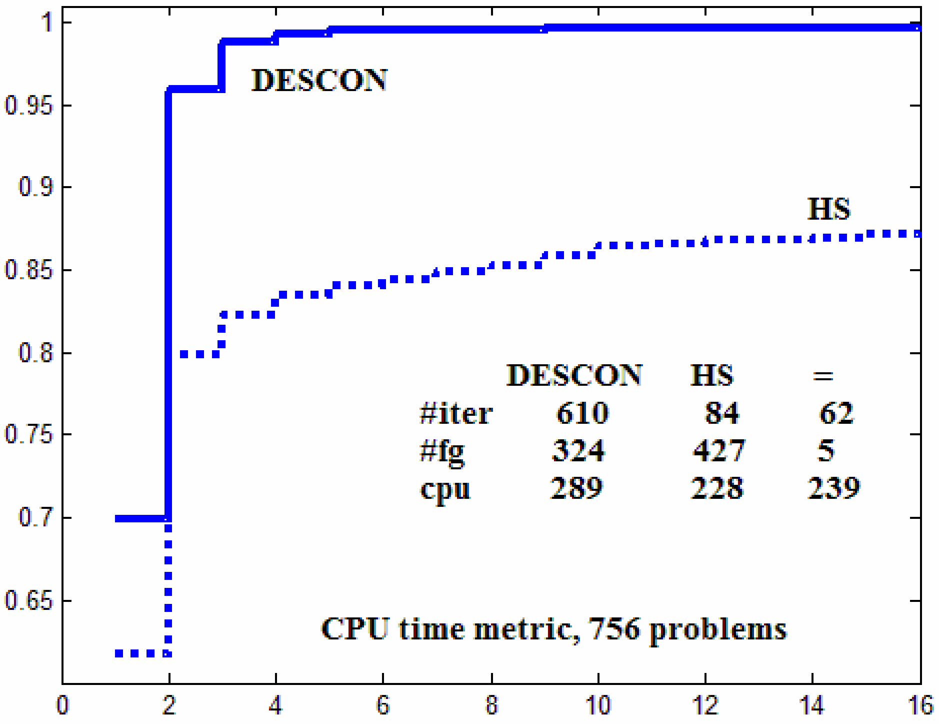

**Fig. 1.** DESCON versus HS. ($\varepsilon^f = 10^{-3}$)

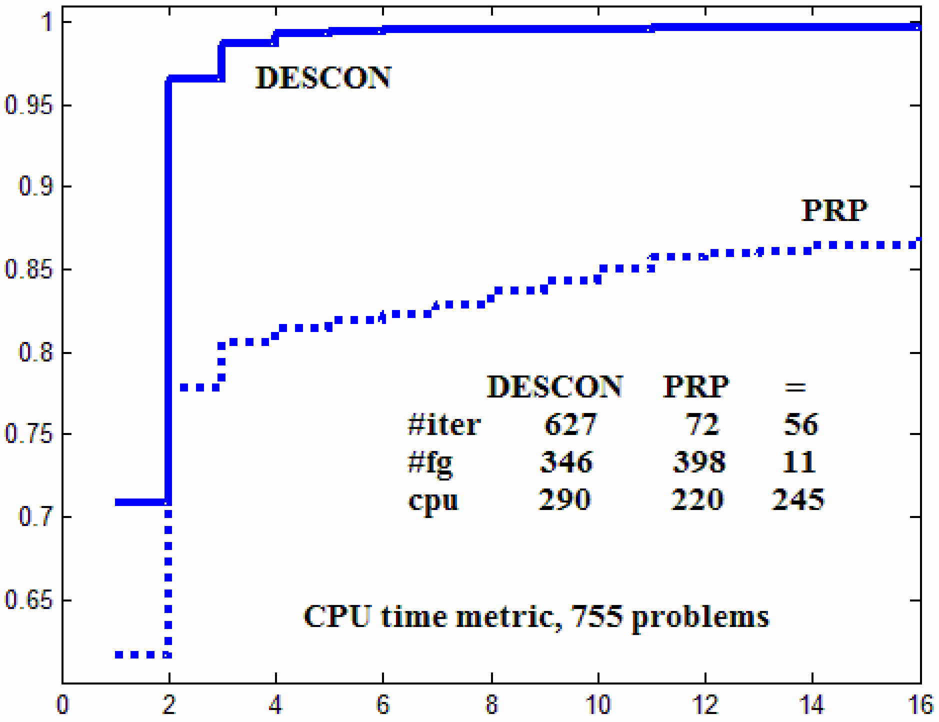

**Fig. 2.** DESCON versus PRP. ($\varepsilon^f = 10^{-3}$)

**Table 1.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus HS.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | HS | DESCON | HS |
| $10^{-3}$ | 756 | 0.69841 | 0.61772 | 0.99735 | 0.87698 |
| $10^{-4}$ | 727 | 0.69876 | 0.62173 | 0.99725 | 0.87208 |
| $10^{-5}$ | 700 | 0.70143 | 0.62857 | 0.99714 | 0.86714 |
| $10^{-6}$ | 677 | 0.70901 | 0.62925 | 0.99705 | 0.86263 |
| $10^{-7}$ | 647 | 0.71252 | 0.63679 | 0.99691 | 0.85781 |
| $10^{-8}$ | 620 | 0.71613 | 0.64677 | 0.99677 | 0.85806 |

**Table 2.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus PRP.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | PRP | DESCON | PRP |
| $10^{-3}$ | 755 | 0.70861 | 0.61589 | 0.99735 | 0.86887 |
| $10^{-4}$ | 724 | 0.71685 | 0.61326 | 0.99724 | 0.86326 |
| $10^{-5}$ | 697 | 0.71736 | 0.62267 | 0.99713 | 0.85796 |
| $10^{-6}$ | 673 | 0.72511 | 0.62407 | 0.99703 | 0.85290 |
| $10^{-7}$ | 629 | 0.72019 | 0.64865 | 0.99682 | 0.85056 |
| $10^{-8}$ | 604 | 0.72351 | 0.64404 | 0.99669 | 0.85099 |

### 5.2. *DESCON versus CONMIN and ASCALCG*

For $\varepsilon^f = 10^{-3}$ Figures 3 and 4 present the performance profiles of DESCON versus CONMIN and ASCALCG, respectively. The best performance, relative to the CPU time metric, again was obtained by DESCON, the solid top curve in Figures 3 and 4. We see that out of 800 problems used in these numerical experiments in case of CONMIN only 730 satisfy criterion (8). In case of ASCALCG only 743 problems satisfy the same criterion. Tables 3 and 4 present the performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus CONMIN and ASCALCG, respectively. From Tables 3 and 4, for $\varepsilon^f = 10^{-3}$, observe that DESCON is 23.287% more efficient than CONMIN and 35.262% more efficient than ASCALCG. Concerning the robustness, from the same tables we see that DESCON is 3.151% more robust than CONMIN and 1.076% more robust than ASCALCG. Observe that DESCON is more efficient and more robust versus CONMIN and ASCALCG, respectively, for every value of $\varepsilon^f$ in the considered set $\{10^{-3},\dots,10^{-8}\}$.

It is worth saying that CONMIN uses the second order information as the BFGS update initialized with the identity matrix at every step. On the other hand, ASCALCG uses the second order information as the BFGS update in which, at every step, the approximate of the inverse Hessian is restarted as $\theta_{k+1} I$, where $\theta_{k+1} = s_k^T s_k / y_k^T s_k$. ($y_k^T s_k > 0$ by Wolfe line search conditions.) Both CONMIN and ASCALCG satisfy the sufficient descent condition, but do not satisfy the conjugacy condition. On the other hand, DESCON satisfies both the sufficient descent and the conjugacy conditions. Besides, DESCON uses the second order information by using the modified conjugacy condition $d_{k+1}^T y_k = -v(g_{k+1}^T s_k)$ with $v = 0.05$.

ASCALCG, is more elaborated than CONMIN. Therefore, subject to robustness, it is closer to DESCON than CONMIN is (see Figure 4).

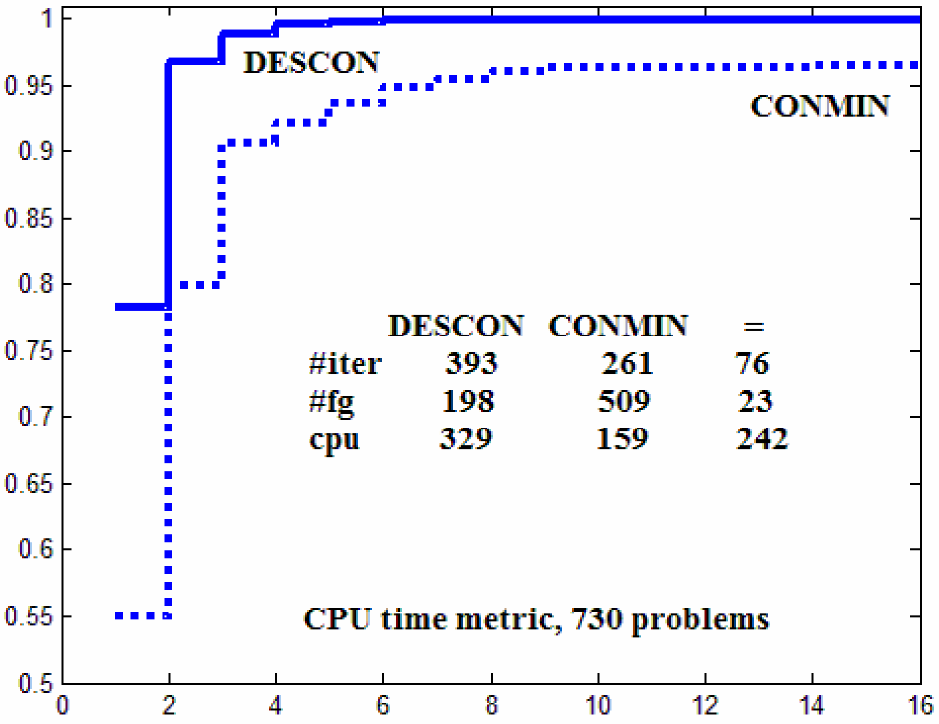

**Fig. 3.** DESCON versus CONMIN. ($\varepsilon^f = 10^{-3}$)

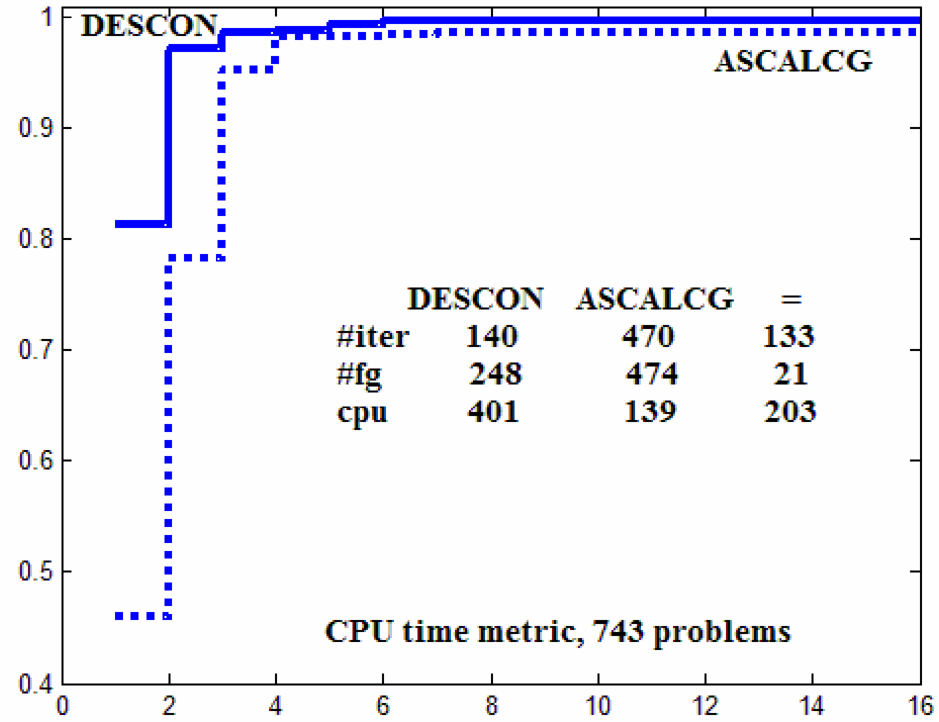

**Fig. 4.** DESCON versus ASCALCG. ($\varepsilon^f = 10^{-3}$)

**Table 3.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus CONMIN.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | CONMIN | DESCON | CONMIN |
| $10^{-3}$ | 730 | 0.78219 | 0.54932 | 0.99863 | 0.96712 |
| $10^{-4}$ | 720 | 0.77917 | 0.55417 | 0.99861 | 0.96667 |
| $10^{-5}$ | 682 | 0.77273 | 0.57918 | 0.99853 | 0.96481 |
| $10^{-6}$ | 660 | 0.77273 | 0.59091 | 0.99848 | 0.96515 |
| $10^{-7}$ | 614 | 0.78013 | 0.60098 | 1 | 0.96254 |
| $10^{-8}$ | 592 | 0.77365 | 0.61318 | 1 | 0.96284 |

**Table 4.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus ASCALCG.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | ASCALCG | DESCON | ASCALCG |
| $10^{-3}$ | 743 | 0.81292 | 0.46030 | 0.99596 | 0.98520 |
| $10^{-4}$ | 721 | 0.81415 | 0.46186 | 0.99584 | 0.98474 |
| $10^{-5}$ | 671 | 0.81669 | 0.46796 | 0.99553 | 0.98510 |
| $10^{-6}$ | 603 | 0.81758 | 0.45439 | 0.99502 | 0.98673 |
| $10^{-7}$ | 518 | 0.80888 | 0.45946 | 0.99421 | 0.99035 |
| $10^{-8}$ | 478 | 0.80335 | 0.45816 | 0.99372 | 0.99163 |

There are problems for which ASCALCG takes very few Beale-Powell restart iterations. For example, for problem #13 (Hager), with $n = 1000$, ASCALCG needs 42 iterations, out of which 1 iteration is a restart one. On the other hand, DESCON needs 44 iterations, out of which 2 iterations are restart iterations. But, for some other problems (not so many in this collection) ASCALCG takes a large number of Beale-Powell restart iterations. For example, for problem #32 (White & Holst), with $n = 1000$, ASCALCG needs 3111 iterations, out of which 1281 (41.176%) are Beale-Powell restart iterations. In this case, DESCON takes only 148 iterations, out of which 12 (8.108%) are restart iterations. This is the weakness of ASCALCG.

### 5.3. *DESCON versus CG-DESCENT*

These two algorithms differ in many respects. CG-DESCENT was designed to guarantee the sufficient descent condition $g_k^T d_k \le -(7/8)\|g_k\|^2$. On the other hand, DESCON is more elaborated it uses the second order information by satisfying the modified conjugacy condition $d_{k+1}^T y_k = -v(g_{k+1}^T s_k)$ with $v = 0.05$ and the sufficient descent condition $g_{k+1}^T d_{k+!} \le -w\|g_{k+1}\|^2$ with $w = 7/8.$ Observe that we consider $w = 7/8$ in DESCON as in CG-DESCENT. Intensive numerical studies and sensitivity analysis [12] proved that DESCON is very little sensitive to the numerical values of $v$ and $w.$ Besides, DESCON is equipped with an acceleration scheme very efficient for improving the values of the minimizing function.

Figure 5 presents the performance profile of DESCON versus CG-DESCENT for $\varepsilon^f = 10^{-3}$. Observe that DESCON is more efficient and more robust versus CG-DESCENT. Out of 800 unconstrained optimization problems considered in this numerical study, only for 774 problems does the criterion (8) holds. Table 5 presents the performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus CG-DESCENT. From Table 5, for $\varepsilon^f = 10^{-3}$, we see that

DESCON is 6.589% more efficient than CG-DESCENT and 4.522% more robust. For $\varepsilon^f = 10^{-8}$, DESCON is 1.540% more efficient than CG-DESCENT and 3.236% more robust. In this case out of 800, only for 649 problems the criterion (8) holds. Observe that DESCON is more efficient and more robust versus CG-DESCENT for every value of $\varepsilon^f$ in the set $\{10^{-3},\ldots,10^{-8}\}$. In DESCON two important ingredients have been implemented: the acceleration and the Beale-Powell restart iterations which are responsible with the performances of it. On the other hand in CG-DESCENT the restart mechanism is very simple: when the number of iterations is a multiple of $n$, then the direction is the negative gradient. Since the number of iterations is much smaller than $n$, the restart iterations are very rare used.

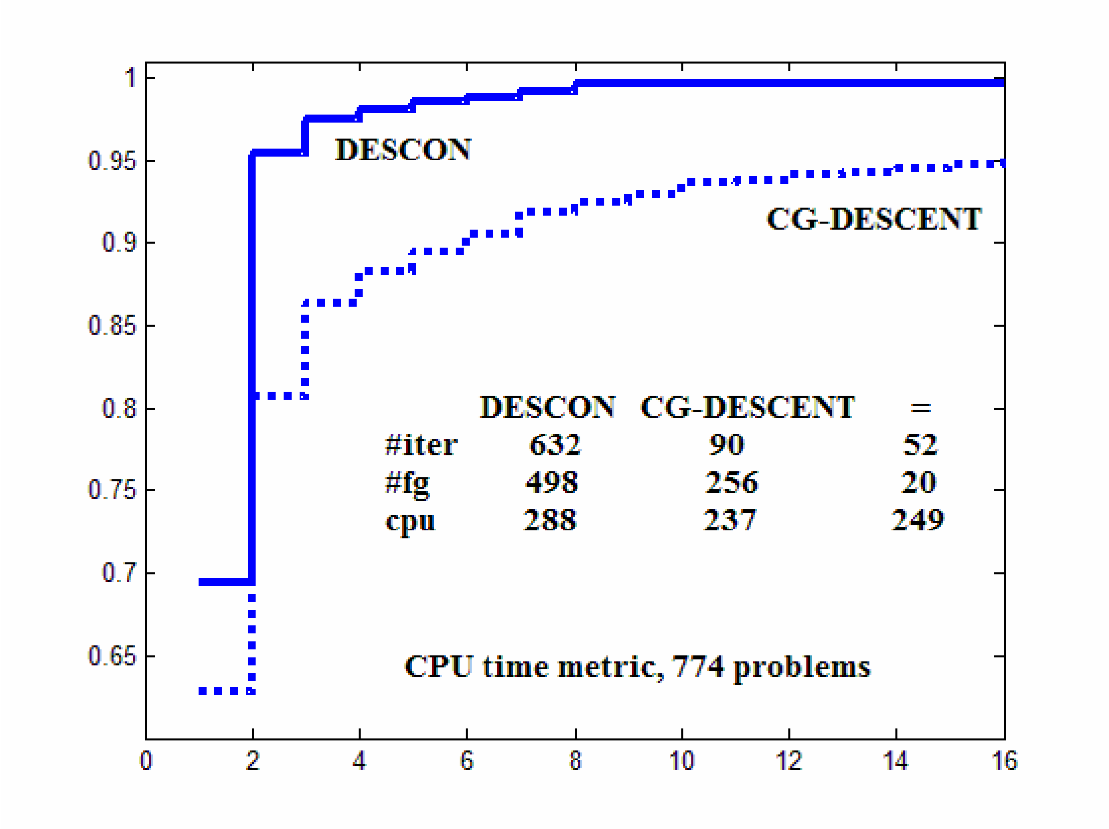

**Fig. 5.** DESCON versus CG-DESCENT. ($\varepsilon^f = 10^{-3}$)

**Table 5.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus CG-DESCENT.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | CG-DES | DESCON | CG-DES |
| $10^{-3}$ | 774 | 0.69380 | 0.62791 | 0.99612 | 0.95090 |
| $10^{-4}$ | 762 | 0.69160 | 0.63517 | 0.99606 | 0.96063 |
| $10^{-5}$ | 729 | 0.68861 | 0.65295 | 0.99588 | 0.95885 |
| $10^{-6}$ | 705 | 0.68511 | 0.66525 | 0.99574 | 0.95887 |
| $10^{-7}$ | 686 | 0.68076 | 0.66910 | 0.99563 | 0.95918 |
| $10^{-8}$ | 649 | 0.69183 | 0.67643 | 0.99538 | 0.96302 |

Besides, we must emphasize that as $\rho$ approaches 0 and $\sigma$ approaches 1, the Wolfe line search terminates quicker. Therefore, the chosen values in CG-DESCENT $\rho = 0.1$ and $\sigma = 0.9$ represent a compromise between the desire for rapid termination of line search and the desire to improve the function value. On the other hand, in DESCON in subroutine for line search we chosen $\rho = 0.0001$ and we limited the number of line search iterations to 3. To improve the function values the acceleration scheme is used which involves only one function evaluation. These are the rationales DESCON is top performer versus CG-DESCENT in Figure 5.

5.4. *DESCON versus the hybrid conjugate gradient algorithm AHYBRIDM*

Figure 6 presents the performance profiles of DESCON versus AHYBRIDM. It is worth saying that AHYBRIDM is based on the concept of hybridization by convex combination of HS and DY conjugate gradient algorithms in order to exploit their attractive features. On one side DY has strong convergence properties and HS in numerical experiments performs better than some other conjugate gradient algorithms, on the other side.

We see that out of 800 unconstrained optimization problems only for 773 problems does the criterion (8) holds. Table 6 presents the performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus AHYBRIDM. From Table 6 we see that DESCON is 34.670% more efficient than AHYBRIDM and 1.164 % more robust.

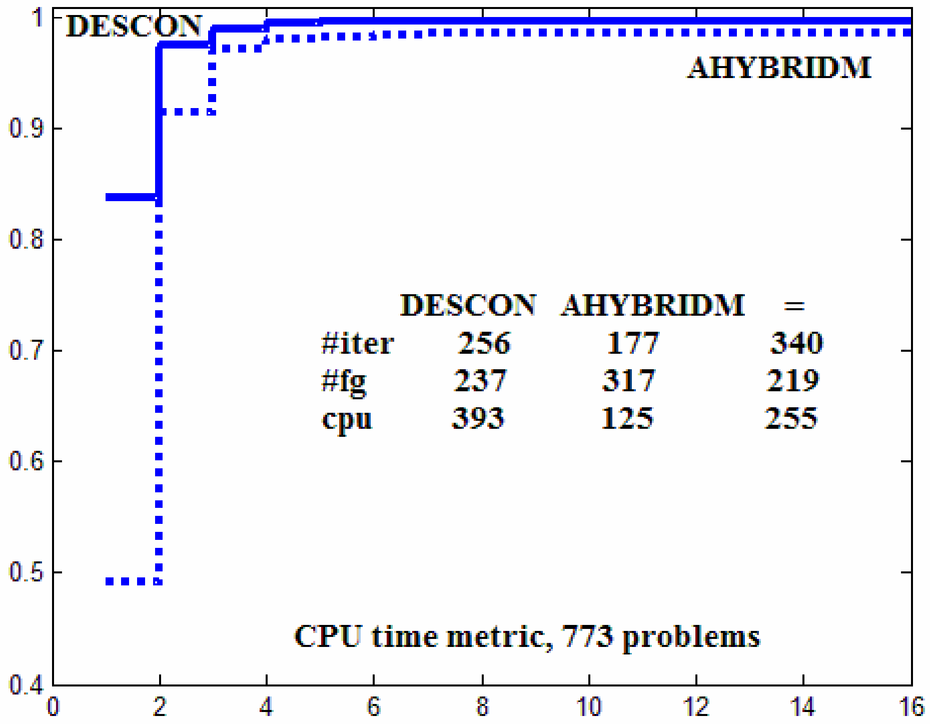

**Fig. 6.** DESCON versus AHYBRIDM. ($\varepsilon^f = 10^{-3}$)

**Table 6.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus AHYBRIDM.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | AHYBRIDM | DESCON | AHYBRIDM |
| $10^{-3}$ | 773 | 0.83829 | 0.49159 | 0.99741 | 0.98577 |
| $10^{-4}$ | 766 | 0.83943 | 0.48825 | 0.99739 | 0.98564 |
| $10^{-5}$ | 746 | 0.83914 | 0.49732 | 0.99732 | 0.98660 |
| $10^{-6}$ | 722 | 0.84211 | 0.50277 | 0.99723 | 0.99169 |
| $10^{-7}$ | 706 | 0.84136 | 0.50992 | 0.99717 | 0.99292 |
| $10^{-8}$ | 693 | 0.83838 | 0.51804 | 0.99711 | 0.99423 |

Observe that DESCON is more efficient and more robust versus AHYBRIDM for every value of $\varepsilon^f$ in the set $\{10^{-3},\ldots,10^{-8}\}$. In numerical experiments we noticed that in AHYBRIDM the iterations often trigger between HS and DY and their convex combination very seldom are used. Rephrased, the performance profile of AHYBRIDM is a little higher than the corresponding profiles of HS and DY. This is the reason why DESCON is far away more efficient than AHYBRIDM.

5.5. *DESCON versus the three-term conjugate gradient algorithm THREECG*

In Figure 7 we have the performance profiles of DESCON versus THREECG. Out of 800 unconstrained optimization test problems only for 778 problems does the criterion (8) holds. Even that there is a discrepancy concerning the efficiency, the algorithms are very close subject to robustness. In Table 7 we see the performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus THREECG.

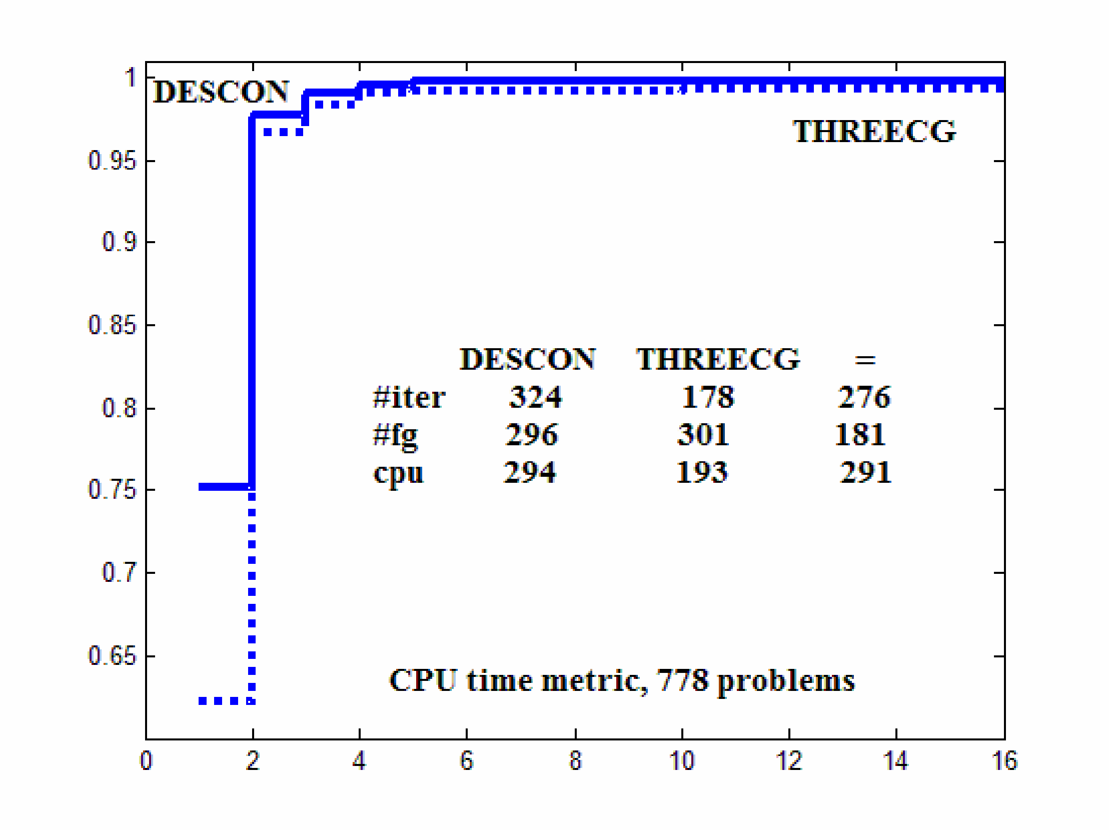

**Fig. 7.** DESCON versus THREECG. ($\varepsilon^f = 10^{-3}$)

**Table 7.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of DESCON versus THREECG.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | DESCON | THREECG | DESCON | THREECG |
| $10^{-3}$ | 778 | 0.75193 | 0.62211 | 0.99743 | 0.99229 |
| $10^{-4}$ | 771 | 0.75097 | 0.61868 | 0.99741 | 0.99222 |
| $10^{-5}$ | 760 | 0.75395 | 0.62105 | 0.99737 | 0.99474 |
| $10^{-6}$ | 737 | 0.75984 | 0.62551 | 0.99729 | 0.99729 |
| $10^{-7}$ | 727 | 0.76754 | 0.62173 | 0.99725 | 0.99725 |
| $10^{-8}$ | 704 | 0.76562 | 0.63636 | 0.99716 | 0.99858 |

For $\varepsilon^f = 10^{-3}$ DESCON is 12.982% more efficient than THREECG and 0.514% more robust. For $\varepsilon^f = 10^{-8}$ we see that THREECG is slightly more robust than DESCON. Both algorithms use the acceleration of iterations, the Beale-Powell restart and the same implementation of the Wolfe line search conditions, but with parameter $\sigma_k$ in DESCON modified at every step. Concerning the restart iterations, we noticed that the number of restart iterations for both algorithms, are very close for all the problems considered in this numerical study. For example, for problem #39 (BDQRTIC) with $n = 1000,$ to get the minimum DESCON needs 77 iterations, out of which 12 (15.584%) are Beale-Powell restart iterations. For the same problem THREECG needs again 77 iterations out of which this time 10 (12.987%) are Beale-Powell restart iterations. Observe that THREECG (15)-(17) is simpler than DESCON (18)-(24), and this could be an advantage. However, both the sufficient descent and the conjugacy conditions are natural properties of the THREECG. On the other hand, in DESCON these properties are imposed through the values of the parameters $v$ and $w,$ which may have a positive influence on its performances.

### 5.6. *ASCALCG versus CG-DESCENT, AHYBRIDM and THREECG*

These algorithms differ in many respects. For example, ASCALCG, AHYBRIDM and THREECG use the acceleration scheme, the same line search procedure based on Wolfe conditions, but different restart procedures. In both algorithms AHYBRIDM and THREECG when the restart Beale-Powell test $\left|g_{k+1}^T g_k\right| \geq 0.2\left\|g_{k+1}\right\|^2$ is satisfied then $d_{k+1} = -g_{k+1},$ the negative gradient. On the other hand, ASCALCG is an accelerated scaled conjugate gradient algorithm, BFGS preconditioned, using an advanced restarting procedure. When the Beale-Powell restart test is satisfied then the restart direction in ASCALCG is computed using again a BFGS preconditioned scheme, which is time consuming in some cases. In our numerical tests we observed that the number of restart iterations is completely unpredictable. For example, for problem #29 (full Hessian) with $n = 1000,$ to get the minimum point ASCALCG needs 480 iterations. The number of restart iterations in ASCALCG is 21 which represent 4.375% out of the total number of iterations. On the other hand, in AHYBRIDM the number of restart iteration to get the solution is 689, out of which 50 (7.256%) are restart iterations. Finally, THREECG needs 610 iterations, out of which 52 (8.524%) are restart iterations. There are some other problems for which the restart iterations are more frequent. For example, for problem #3 (extended Rosenbrock) with $n = 1000$ the number of iterations to get the minimum point is 49, out of which 42, i.e. 85.714%, are restart iterations. On the other hand AHYBRIDM needs 55 iterations for solving this problem. The number of restart iterations in this case is 20 which represent 36.363% out the total number of iterations. THREECG takes 61 iterations to solve this problem. From these 22 are restart iterations which represent 36.065%. In any case, in conjugate gradient algorithms restart is an important ingredient representing a relaxation of algorithm.

In Figures 8, 9 and 10 we have the performance profiles of ASCALCG versus CG-DESCENT, AHYBRIDM and THREECG, respectively. Observe that all these three conjugate gradient algorithms CG-DESCENT, AHYBRIDM and THREECG are more efficient than ASCALCG. For example, from Tables 8, 9 and 10, for $\varepsilon^f = 10^{-3},$ CG-DESCENT is 14.577% more efficient than ASCALCG; AHYBRIDM is 20.755% more efficient than ASCALCG and finally THREECG is again 31.241% more efficient than ASCALCG. Observe that the great discrepancy concerning efficiency is between ASCALCG and THREECG.

Concerning robustness, ASCALCG is more robust than CG-DESCENT (0.817%) and AHYBRIDM (0.135%). However, from Table 10 we see that THREECG is 0.267% more robust than ASCALCG. The direction in ASCALCG is more complicated (more time consuming) and this is the reason why its efficiency is smaller than the corresponding efficiency of the algorithms we compare with.

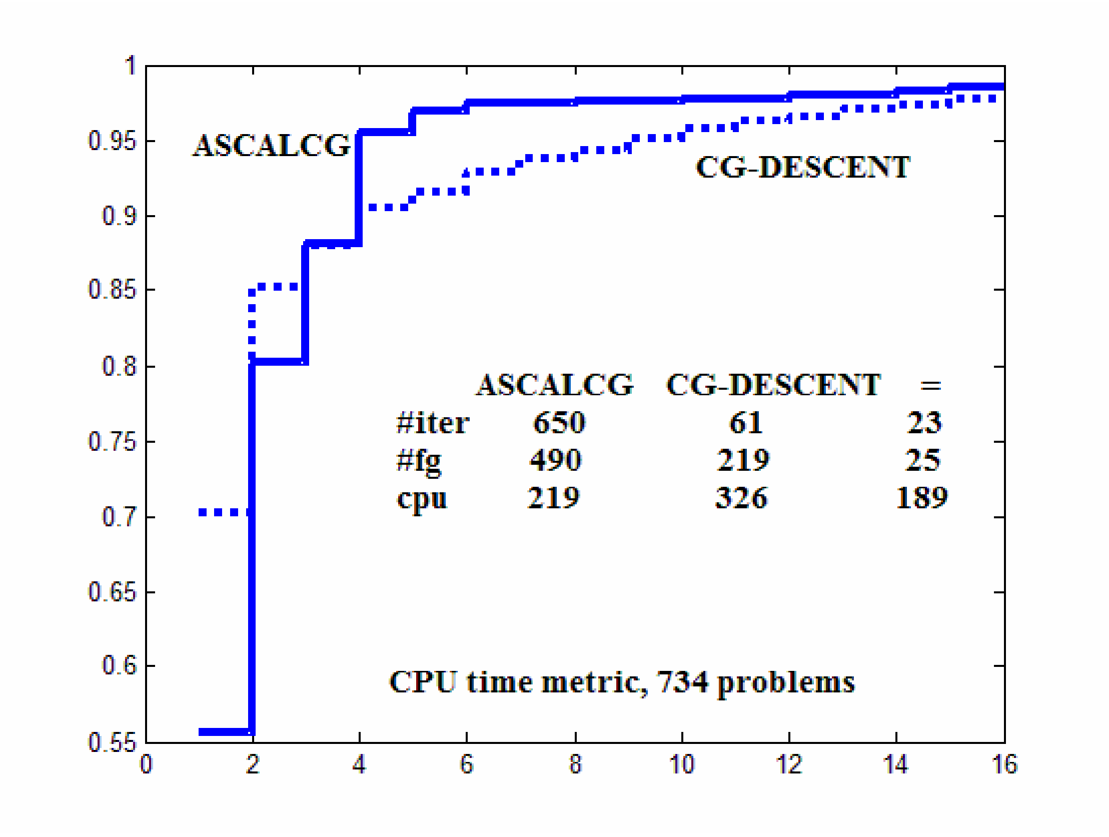

**Fig. 8.** ASCALCG versus CG-DESCENT. ($\varepsilon^f = 10^{-3}$)

**Table 8.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of ASCALCG versus CG-DESCENT.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | ASCALCG | CG-DESCENT | ASCALCG | CG-DESCENT |
| $10^{-3}$ | 734 | 0.55586 | 0.70163 | 0.98501 | 0.97684 |
| $10^{-4}$ | 715 | 0.56364 | 0.69930 | 0.98462 | 0.97622 |
| $10^{-5}$ | 674 | 0.56528 | 0.69733 | 0.98516 | 0.97478 |
| $10^{-6}$ | 595 | 0.54622 | 0.70252 | 0.98655 | 0.97311 |
| $10^{-7}$ | 511 | 0.54795 | 0.69863 | 0.99022 | 0.97065 |
| $10^{-8}$ | 453 | 0.54967 | 0.69978 | 0.99117 | 0.98455 |

Besides the restart and the acceleration, the other profitable ingredient used in AHYBRIDM is the second order information given by the modified secant condition. In contrast to ASCALCG the second order information in AHYBRIDM is used in a very direct and simple manner. Therefore, the algebraic expression of the direction is not complicated, this being very easy to be computed. If the line search is exact, the direction reduces to a minor modification of the HS algorithm. On the other hand, THREECG is a very simple three-term conjugate gradient algorithm which uses the second order information as a minor modification of the BFGS updating formula. In this algorithm the BFGS formula is restarted with the identity matrix at every step and the sign in front of the $y_k s_k^T$ term is changed in order to get the descent property. Again observe that THREECG is a modification of HS algorithm. This modification is dependent by $s_k^T g_{k+1}$ which is going to zero along the iterations. Apparently, this contribution to the HS direction determines a better direction in THREECG. Concerning the robustness, observe that ASCALCG and THREECG are bunched closer together. This is because ASCALCG in its essence is a more complicated three-term conjugate gradient algorithm (see (9)-(11)).

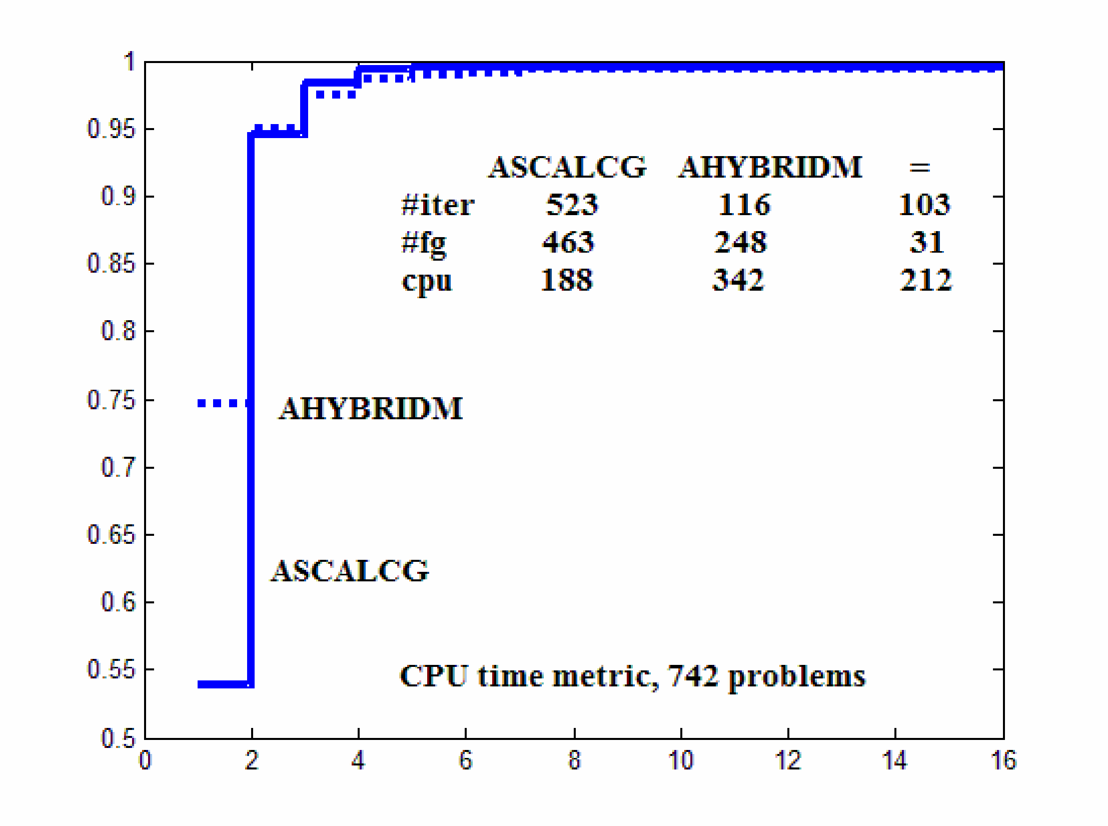

**Fig. 9.** ASCALCG versus AHYBRIDM. ($\varepsilon^f = 10^{-3}$)

**Table 9.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of ASCALCG versus AHYBRIDM.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | ASCALCG | AHYBRIDM | ASCALCG | AHYBRIDM |
| $10^{-3}$ | 742 | 0.53908 | 0.74663 | 0.99461 | 0.99326 |
| $10^{-4}$ | 719 | 0.54242 | 0.74548 | 0.99444 | 0.99305 |
| $10^{-5}$ | 670 | 0.55224 | 0.74179 | 0.99552 | 0.99552 |
| $10^{-6}$ | 608 | 0.53947 | 0.74836 | 0.99507 | 0.99671 |
| $10^{-7}$ | 517 | 0.53772 | 0.74855 | 0.99807 | 0.99613 |
| $10^{-8}$ | 471 | 0.54140 | 0.74310 | 0.99788 | 0.99788 |

Together, Figures 8, 9 and 10 seem to imply that ASCALCG is the least efficient algorithm in comparison to CG-DESCENT, AHYBRIDM and THREECG. The linear algebra in ASCALCG code to update the search direction is more time consuming than the linear algebra in the algorithms we compare with. Therefore, the CPU time of ASCALCG is dominated by the cost of linear algebra. On the other hand, the number of iterations in line search in ASCALCG, AHYBRIDM and THREECG is limited to 3 to get an acceptable step length. Using the acceleration technique in these algorithms this value of the step length is modified to reduce the value of the minimizing function. Also, it is interesting to observe that the performances of these algorithms are dependent on this number, which limits the iterations in line search. It seems that the smaller the number of line search iterations, the more efficient the algorithms are. In CG-DESCENT, in the line search, more function evaluations are needed to get the stopping criterion. However, the value of the step length in

CG-DESCENT is more accurate, and this is the rationale concerning the efficiency of this algorithm.

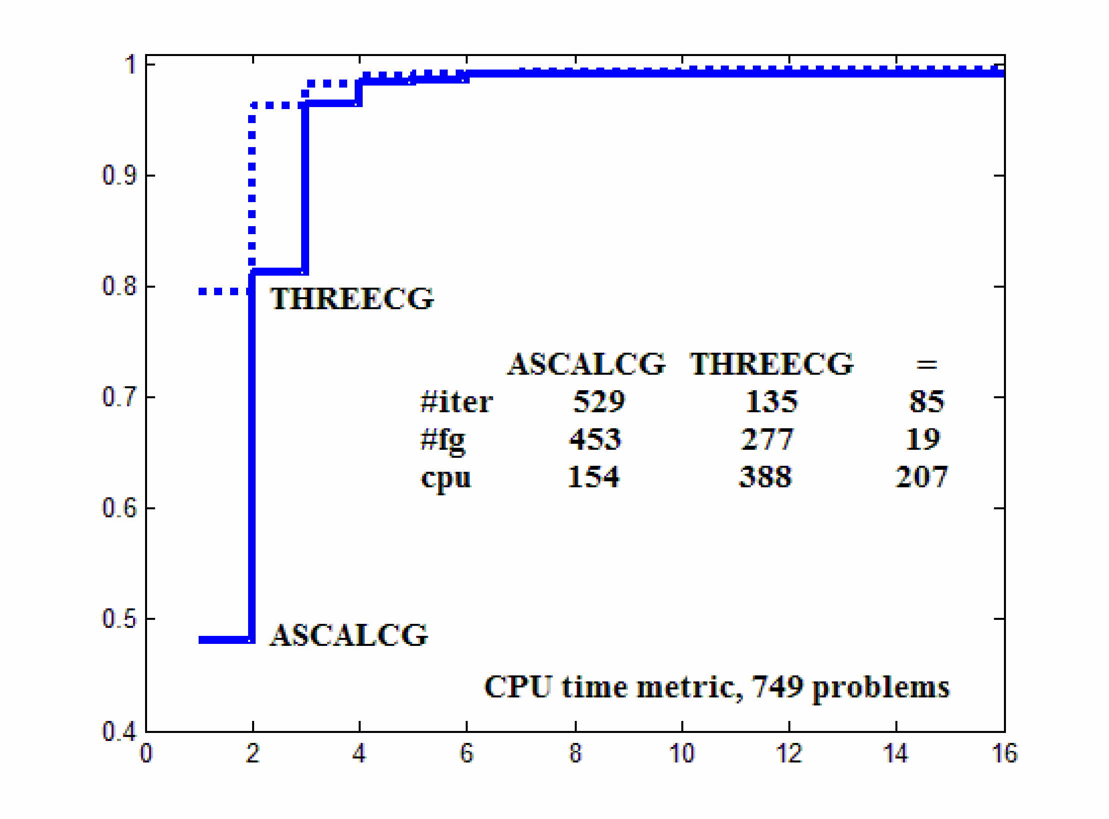

**Fig. 10.** ASCALCG versus THREECG. ($\varepsilon^f = 10^{-3}$)

**Table 10.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of ASCALCG versus THREECG.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | ASCALCG | THREECG | ASCALCG | THREECG |
| $10^{-3}$ | 749 | 0.48198 | 0.79439 | 0.99199 | 0.99466 |
| $10^{-4}$ | 726 | 0.48623 | 0.79614 | 0.99174 | 0.99449 |
| $10^{-5}$ | 672 | 0.49405 | 0.79911 | 0.99256 | 0.99554 |
| $10^{-6}$ | 608 | 0.48191 | 0.80263 | 0.99342 | 0.99671 |
| $10^{-7}$ | 521 | 0.48369 | 0.80230 | 0.99616 | 0.99616 |
| $10^{-8}$ | 475 | 0.48421 | 0.79368 | 0.99789 | 0.99789 |

### 5.7. *CG-DESCENT versus AHYBRIDM and THREECG*

For $\varepsilon^f = 10^{-3}$, in Figures 11 and 12 we present the performance profiles of CG-DESCENT versus AHYBRIDM and THREECG, respectively. Observe that, at least for this set of problems, CG-DESCENT is more efficient than AHYBRIDM, and the difference is significant, about 14.305%. However, in comparison with THREECG, observe that this time THREECG is very little more efficient, about 1.428%. Concerning the robustness, both AHYBRIDM and THREECG are more robust than CG-DESCENT. From Tables 11 and 12 we see that this characteristic of robustness of AHYBRIDM and THREECG is maintained for all values of $\varepsilon^f$ in the set $\{10^{-3},\ldots,10^{-8}\}$. For example, for $\varepsilon^f = 10^{-3}$, from Table 11 we see that AHYBRIDM is about 0.918% more robust than CG-DESCENT. Again, from Table 12 observe that THREECG is 1.948% more robust than CG-DESCENT.

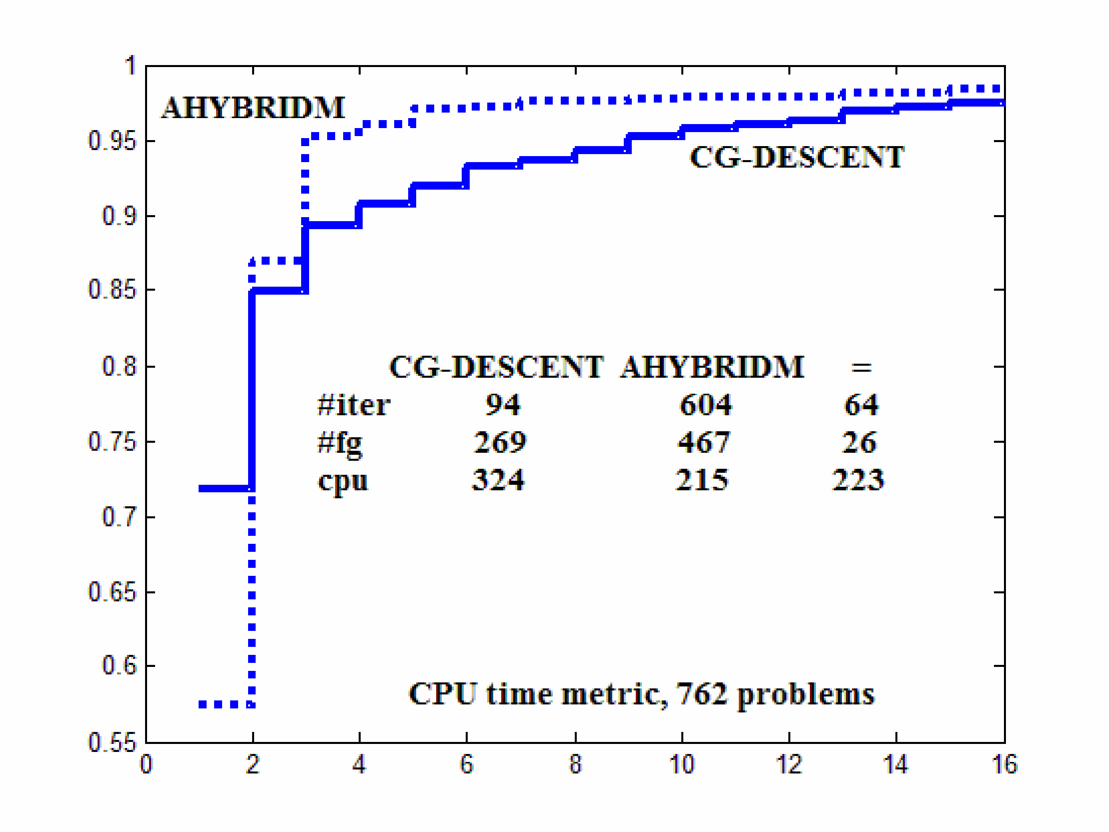

**Fig. 11.** CG-DESCENT versus AHYBRIDM. ($\varepsilon^f = 10^{-3}$)

**Table 11.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of CG-DESCENT versus AHYBRIDM.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | CG-DESCENT | AHYBRIDM | CG-DESCENT | AHYBRIDM |
| $10^{-3}$ | 762 | 0.71785 | 0.57480 | 0.97507 | 0.98425 |
| $10^{-4}$ | 754 | 0.72281 | 0.57162 | 0.97878 | 0.98408 |
| $10^{-5}$ | 731 | 0.73598 | 0.56772 | 0.97811 | 0.98632 |
| $10^{-6}$ | 703 | 0.74964 | 0.56330 | 0.97724 | 0.99147 |
| $10^{-7}$ | 680 | 0.75294 | 0.56324 | 0.97794 | 0.99118 |
| $10^{-8}$ | 643 | 0.75894 | 0.57543 | 0.98289 | 0.99222 |

We emphasize that in CG-DESCENT the mechanism of restarting the iterations is very simple. Since $n$ is often larger than the number of iterations needed for solving a problem, it follows that the algorithm performs no restarts in the vast majority of test runs. This is the reason why the CG-DESCENT is more efficient than AHYBRIDM. The iterations in CG-DESCENT are not relaxed as in AHYBRIDM and THREECG where the Beale-Powell restart test is used.

As we know, AHYBRIDM is an accelerated hybrid conjugate gradient algorithm obtained as a convex combination of HS and DY conjugate gradient algorithms. In our intensive numerical experiments we observed that the AHYBRIDM have the propensity to use the HS algorithm along the iterations. Besides, we notice that AHYBRIDM often triggers between HS and DY, while their convex combination is seldom used. For example for problem #3 (extended Rosenbrock) with $n = 1000$, AHYBRIDM needs 55 iterations. Out of these, the HS algorithm is used in 39 (70.92%) iterations, the DY algorithm is used in 14 (25.45%), and the convex combination of HS and DY is used in 2 (3.63%) iterations. This is a typical behavior of AHYBRIDM. As we said, CG-DESCENT also is a modification of HS algorithm, but this modification is always used along the iterations without any possibility to change it at least as the negative gradient in case of restart. This is the rationale for a better robustness of AHYBRIDM versus CG-DESCENT.

From (12) and (16) observe that both CG-DESCENT and THREECG have in common the expression $\|y_k\|(s_k^T g_{k+1})/(y_k^T s_k)^2$. When iterates jam, $y_k$ becomes tiny while $\|g_k\|$ is bounded away from zero. Therefore, in CG-DESCENT when iterates jam, this expression becomes negligible, i.e. $\beta_k^{HZ} = \beta_k^{HS}$. However, in case of jamming, in THREECG there is the third component $-\eta_k y_k$ (see 15) which compensate the lost of robustness of CG-DESCENT.

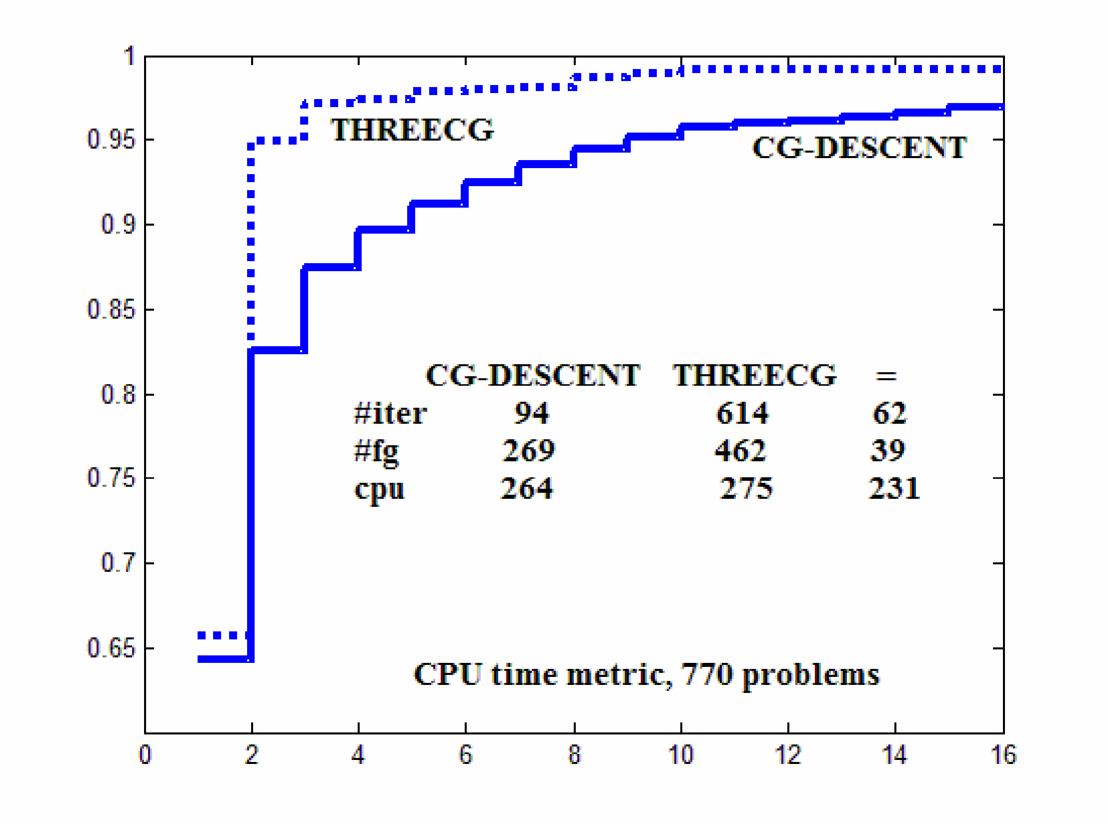

**Fig. 12.** CG-DESCENT versus THREECG. ($\varepsilon^f = 10^{-3}$)

**Table 12.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of CG-DESCENT versus THREECG.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | CG-DESCENT | THREECG | CG-DESCENT | THREECG |
| $10^{-3}$ | 770 | 0.64286 | 0.65714 | 0.97143 | 0.99091 |
| $10^{-4}$ | 762 | 0.64567 | 0.65748 | 0.97507 | 0.99081 |
| $10^{-5}$ | 729 | 0.66255 | 0.65432 | 0.97394 | 0.99314 |
| $10^{-6}$ | 704 | 0.67756 | 0.64631 | 0.97301 | 0.99574 |
| $10^{-7}$ | 682 | 0.68182 | 0.64370 | 0.97507 | 0.99560 |
| $10^{-8}$ | 642 | 0.68692 | 0.65732 | 0.97975 | 0.99688 |

### 5.8. *AHYBRIDM versus THREECG*

In Figure 13 we have the performance profiles of these algorithms for $\varepsilon^f = 10^{-3}$. Out of 800 problems only for 782 problems the criterion (8) holds. Observe that THREECG is about 30.563% more efficient and about 0.511% more robust then AHYBRIDM. Besides, from

Table 13 we see that this characteristic of these algorithms is invariant at the variation of $\varepsilon^f$ in the set $\{10^{-3},\ldots,10^{-8}\}$. As we have already seen these algorithms are different in many respects. Even that AHYBRIDM often triggers between HS and DY trying to exploit the attractive features of these algorithms, THREECG is more efficient and more robust showing the importance of three-term concept in conjugate gradient paradigm. On the other hand, the algebraic expression of the search direction in THREECG is simpler then the search direction in AHYBRIDM. This makes THREECG more efficient that AHYBRIDM.

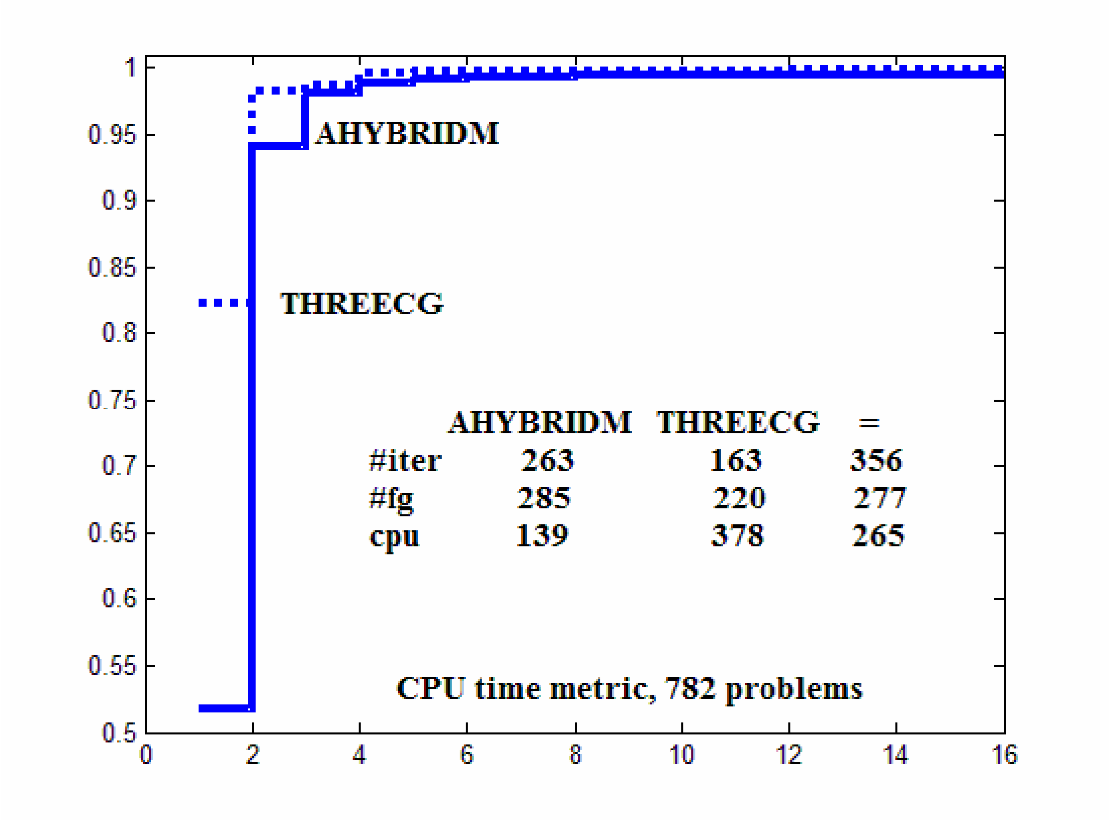

**Fig. 13.** AHYBRIDM versus THREECG. ($\varepsilon^f = 10^{-3}$)

**Table 13.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of AHYBRIDM versus THREECG.

| $\varepsilon^f$ | Nrp | $\Gamma(1)$ | | $\Gamma(\infty)$ | |
|---|---|---|---|---|---|
| | | AHYBRIDM | THREECG | AHYBRIDM | THREECG |
| $10^{-3}$ | 782 | 0.51662 | 0.82225 | 0.99361 | 0.99872 |
| $10^{-4}$ | 782 | 0.51662 | 0.82225 | 0.99361 | 0.99872 |
| $10^{-5}$ | 763 | 0.52425 | 0.82307 | 0.99345 | 0.99869 |
| $10^{-6}$ | 741 | 0.52632 | 0.82861 | 0.99595 | 1 |
| $10^{-7}$ | 726 | 0.53444 | 0.82782 | 0.99725 | 1 |
| $10^{-8}$ | 704 | 0.54403 | 0.82812 | 0.99716 | 1 |

### 5.9. *The performance profiles of all algorithms*

Firstly, in this section we present the performance profile of all eight algorithms considered in this numerical study for $\varepsilon^f = 10^{-3}$. The top solid curve in Figure 14 corresponds to DESCON, the top performer among these algorithms. In Table 14 we can see the efficiency $\Gamma(1)$ and the robustness $\Gamma(\infty)$ of these algorithms, relative to the CPU time metric. Concerning the efficiency CG-DESCENT is top performer. The second place is taken by

DESCON and the third by HS. Concerning the robustness on the first place is DESCON, followed by THREECG and followed by ASCALCG.

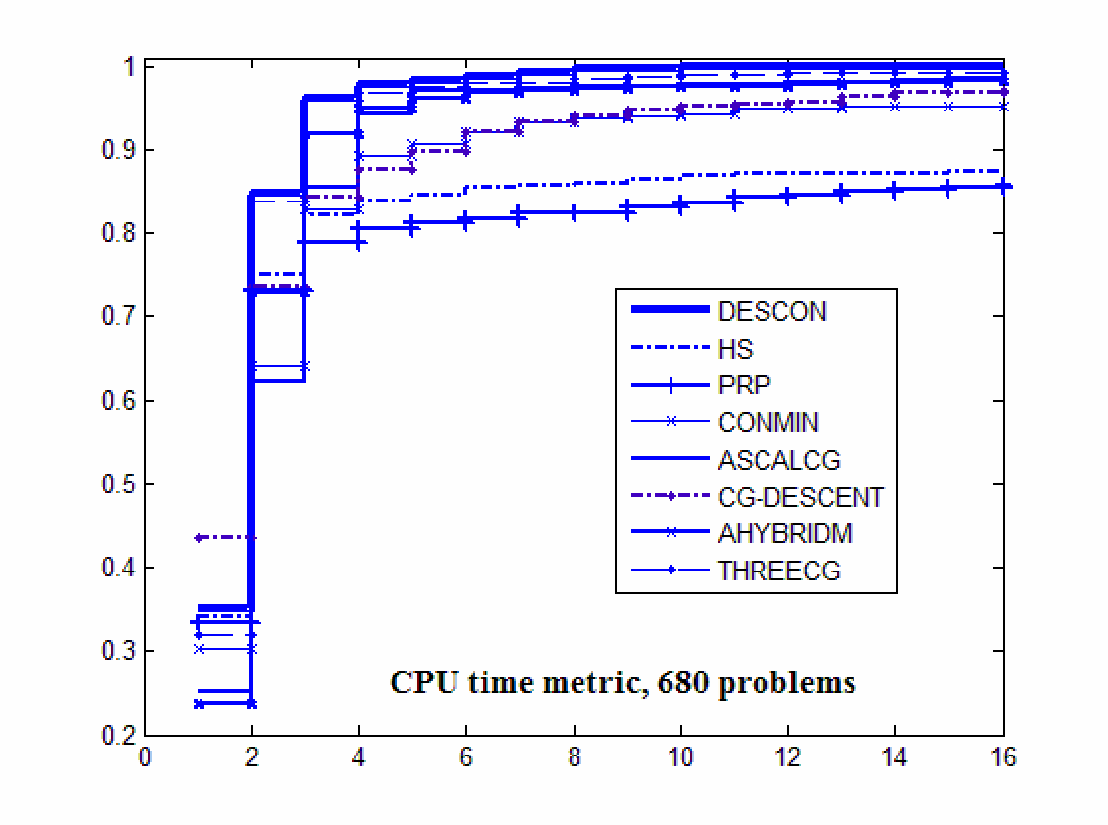

**Fig. 14.** Performance profiles of all algorithms for $\varepsilon^g = 10^{-6}$ and $\varepsilon^f = 10^{-3}$.

**Table 14.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of all algorithms. The **first,** the *second* and the <u>third</u> places of algorithms are shown in **bold**, *italic* and <u>underline</u>, respectively.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| DESCON | *0.35000* | **0.99853** |
| HS | <u>0.34118</u> | 0.87353 |
| PRP | 0.33382 | 0.85735 |
| CONMIN | 0.30294 | 0.95294 |
| ASCALCG | 0.25147 | <u>0.98529</u> |
| CG-DESCENT | **0.43529** | 0.97059 |
| AHYBRIDM | 0.23676 | 0.98382 |
| THREECG | 0.32059 | *0.99118* |

In Figure 14 observe that HS and PRP have the most reduced performance profiles. Therefore, in Figure 15 we present the performance profiles of five algorithms for $\varepsilon^f = 10^{-3}$. Observe in Figure 15 that concerning the robustness the algorithms are grouped, but subject to efficiency they are more dispersed, slightly fastest being CG-DESCENT. Again, the top solid curve in Figure 15 corresponds to DESCON. Subject to the efficiency, from Table 15, we see that CG-DESCENT is slightly faster, followed by DESCON and followed by THREECG. Concerning the robustness, the DESCON is the most robust, followed by THREECG and followed by ASCALCG. Since all these algorithms use the same line search procedure, based on the Wolfe conditions, DESCON appears to generate the best search direction, on average.

In Figure 15, we have the computational evidence that these five algorithms are the best conjugate gradient algorithms able to solve a large variety of large-scale unconstrained optimization problems of different structures of their Hessian. Excepting CG-DESCENT all the algorithms considered in Figure 15 implement an acceleration procedure which proves to be very efficient in reducing the values of the function values. On the other hand, these algorithms contain in a way or another, the second order information which improve in a certain way the computation of the search direction.

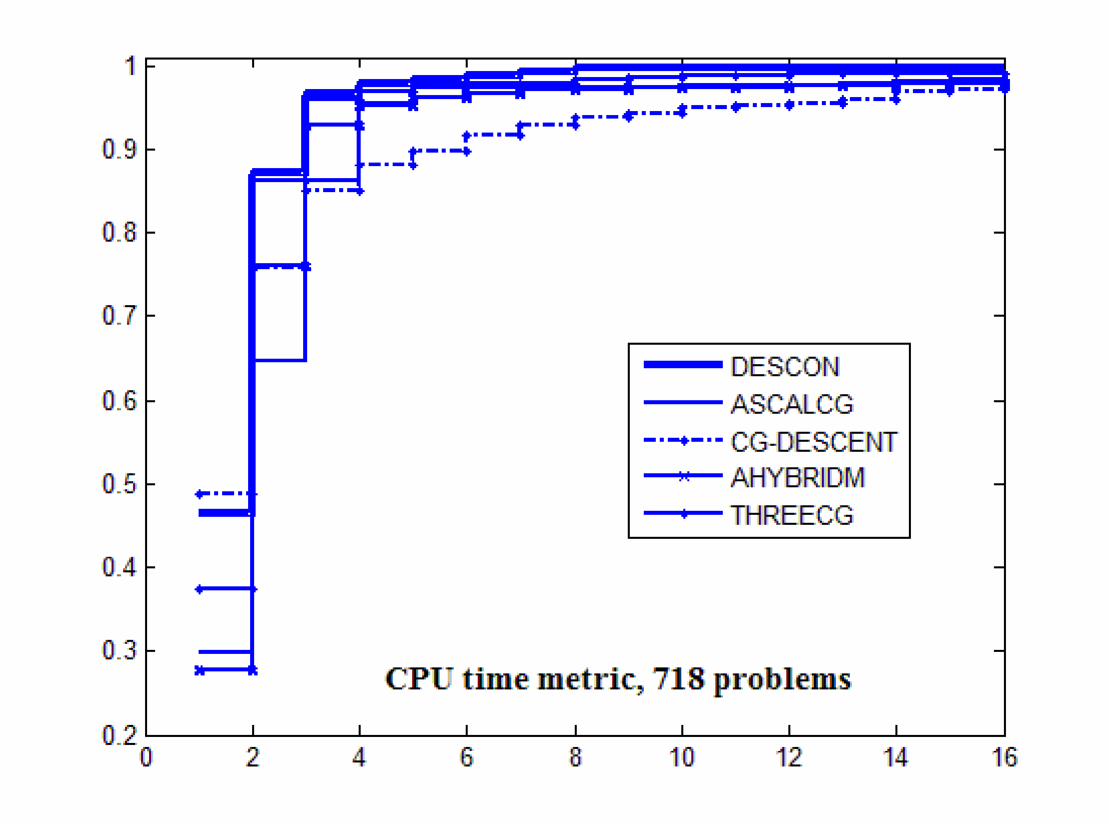

**Fig. 15.** Performance profiles of 5 algorithms for $\varepsilon^g = 10^{-6}$ and $\varepsilon^f = 10^{-3}$.

**Table 15.** Performance profiles $\Gamma(1)$ and $\Gamma(\infty)$ of 5 algorithms.
The **first,** the *second* and the third places of algorithms are shown in **bold**, *italic* and underline, respectively.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| DESCON | *0.46379* | **0.99582** |
| ASCALCG | 0.29805 | 0.98329 |
| CG-DESCENT | **0.48886** | 0.97214 |
| AHYBRIDM | 0.27716 | 0.98050 |
| THREECG | 0.37326 | *0.98886* |

## 6. Discussion

6.1. *Comparisons among algorithms for solving problems with different structure of the Hessian matrix*

In this numerical study we classified the problem according to the structure of their Hessian matrix. Hence, out of 800 unconstrained optimization test problems, considered in this paper, for 100 of them the Hessian is a diagonal matrix, for 190 the Hessian is a block-diagonal matrix, for 220 the Hessian is tri-diagonal (or penta-diagonal) and for 160 the Hessian is a full matrix. The rest of the problems have a bounded diagonal or a bounded block-diagonal structure of the Hessian matrix we do not consider in our analysis. In this section we present a comparison of AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG conjugate gradient algorithms for solving these four classes of unconstrained optimization test problems. The below tables present the efficiency and the robustness of these algorithms.

**Table 16.** The efficiency and robustness of algorithms for solving 100 test problems with diagonal Hessian matrix. $\varepsilon^g = 10^{-6}$, $\varepsilon^f = 10^{-3}$.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| AHYBRIDM | 0.17582 | 0.97802 |
| ASCALCG | 0.17582 | 0.96703 |
| CG-DESCENT | **0.89011** | 1 |
| DESCON | *0.26374* | 1 |
| THREECG | 0.26374 | 1 |

**Table 17.** The efficiency and robustness of algorithms for solving 190 test problems with block-diagonal Hessian matrix. $\varepsilon^g = 10^{-6}$, $\varepsilon^f = 10^{-3}$.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| AHYBRIDM | 0.31138 | 1 |
| ASCALCG | 0.27545 | 1 |
| CG-DESCENT | **0.55090** | 0.99401 |
| DESCON | 0.41317 | 1 |
| THREECG | *0.48503* | 1 |

**Table 18.** The efficiency and robustness of algorithms for solving 220 test problems with tri-diagonal or penta-diagonal Hessian matrix. $\varepsilon^g = 10^{-6}$, $\varepsilon^f = 10^{-3}$.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| AHYBRIDM | 0.14894 | 0.99468 |
| ASCALCG | 0.06915 | 0.99468 |
| CG-DESCENT | **0.52128** | 1 |
| DESCON | *0.43085* | 1 |
| THREECG | 0.25000 | 0.99468 |

**Table 19.** The efficiency and robustness of algorithms for solving 160 test problems with full Hessian matrix. $\varepsilon^g = 10^{-6}$, $\varepsilon^f = 10^{-3}$.

| | $\Gamma(1)$ | $\Gamma(\infty)$ |
|---|---|---|
| AHYBRIDM | 0.27451 | 0.94118 |
| ASCALCG | 0.35948 | 0.94771 |
| CG-DESCENT | *0.36601* | 0.92810 |
| DESCON | **0.43137** | **0.98039** |
| THREECG | 0.31373 | 0.95425 |

Observe that CG-DESCENT is the most efficient algorithm for solving problems with structured Hessian. On the other hand, DESCON is the most efficient and the most robust algorithm for solving problems with full Hessian.

It is worth seeing the behavior of these algorithms for solving these four classes of problems subject to the CPU time metric. In Table 20 we present the total CPU time for solving these classes of problems with the Hessian matrix structured as: diagonal (DD), block-diagonal (BD), tri-diagonal or penta-diagonal (TP) and full Hessian (FH). Observe that for solving 100 unconstrained optimization problems with Hessian a diagonal matrix all algorithms need a grand total of 958.85 seconds. AHYBRIDM needs 286.45 seconds. Therefore, for AHYBRIDM, in average one problem in this class needs 286.45/100=2.8645 seconds. Observe that the fastest algorithm for solving problems in this class is CG-DESCENT. For solving one problem, in average this algorithm needs 49.51/100=0.4951 seconds. For solving this class of problems DESCON is on the second place (in italics).

From Table 20 we have the computational evidence that all algorithms considered in this study are fastest for solving problems whose Hessian is a bloc-diagonal matrix. For solving 190 problems, with Hessian a block-diagonal matrix, all algorithms need a grand total of 40.82 seconds. We see that, in average, for solving one problem, for which the Hessian is bloc-diagonal, THREECG needs 7.22/190=0.038 seconds, this algorithm being the fastest among all the algorithms considered in this numerical study. Observe that DESCON again is on the second place, etc.

**Table 20.** CPU time (seconds) for solving unconstrained optimization test problems classified as: DD, BD, TP and FH. $\varepsilon^g = 10^{-6}$, $\varepsilon^f = 10^{-3}$.
The **first**, the *second* places of algorithms are shown in **bold** and *italic*, respectively.

| | DD | BD | TP | FH |
|---|---|---|---|---|
| | 100 | 190 | 220 | 160 |
| AHYBRIDM | 286.45<br>(2.8645) | 8.26<br>(0.0434) | 517.18<br>(2.3508) | 594.77<br>(3.7173) |
| ASCALCG | 299.96<br>(2.9996) | 8.42<br>(0.0443) | 795.11<br>(3.6141) | 406.94<br>(**2.5433**) |
| CG-DESCENT | 49.51<br>(**0.4951**) | 9.39<br>(0.0492) | 394.88<br>(1.7949) | 1455.53<br>(9.0970) |
| DESCON | 153.00<br>(*1.53*) | 7.53<br>(*0.0396*) | 363.78<br>(**1.6535**) | 442.69<br>(*2.7668*) |
| THREECG | 169.93<br>(1.6993) | 7.22<br>(**0.038**) | 379.18<br>(*1.7235*) | 452.63<br>(2.8289) |
| TOTAL | 958.85 | 40.82 | 2450.13 | 3352.56 |

Concerning the 220 problems with Hessian a tri-diagonal or a penta-diagonal matrix all algorithms need a grand total of 2450.13 seconds. The fastest algorithm for solving the problems from this class is DESCON. In average, it needs 363.78/220=1.6535 seconds. The second place is taken by THREECG.

Subject to CPU time metric, the most difficult problems seem to be the problems with full Hessian. For solving 160 problems with full Hessian all algorithms need a grand total of 3352.56 seconds, ASCALCG being the fastest for solving these problems. Again DESCON is on the second place.

As we know the convergence of conjugate gradient algorithms is very dependent by the entire spectrum of the Hessian. Suppose that the Hessian is a positive definite matrix. If the eigenvalues of the Hessian matrix are contained in, let say, $m$ disjoint intervals of very small length on the real axis, then the conjugate gradient algorithms will produce very small gradients after at most $m$ steps. In case of functions with Hessian a block-diagonal matrix the eigenvalues of Hessian are clustered in a number of disjoint intervals. Therefore, for these sorts of functions all the algorithms considered in this numerical study are faster versus functions with some other structures of the Hessian.

6.2. *The weakness of numerical experiments and comparisons using artificially test problems*
From the above numerical experiments and comparisons we have the computational evidence that the conjugate gradient algorithms considered in this numerical study are able to solve a large variety of large-scale unconstrained optimization problems of different nonlinear complexity and with different structures of their Hessian matrix. This is the main remark of this numerical study.

Apparently some algorithms are more efficient, or more robust, or faster than others. For example, from Figures 14 and 15, it seems that the algorithms DESCON and THREECG, for which both the sufficient descent condition and the conjugacy condition are satisfied, are the best in this class of algorithms. But this is not a definitive conclusion. This behavior is obtained by means of a relatively large collection of artificially unconstrained optimization test problems we have used in our numerical study. It is quite clear that in front of us we have an infinite number of artificially unconstrained optimization test problems from which it is always possible to assemble a set of problems for which completely different conclusions about the efficiency and robustness of the algorithms considered in this numerical study are obtained. This is the weakness of numerical studies using artificially optimization test problems, even that they are of different nonlinear complexity and with different structures of their Hessian matrix.

Therefore, in order to get a true conclusion at all the real unconstrained optimization applications must be used in numerical experiments and comparisons. The main characteristic of real optimization applications is that their mathematical model is written on the basis of the conservation laws. In this respect the Noether theorem [39] shows that the conservation laws are direct consequences of symmetries. But, in any time and any place we are surrounded by concepts that appear in dual-symmetric pairs. Therefore, the conservation laws have very solid fundamentals which are directly transmitted to the mathematical models of real applications. For example [25] and [46] present plenty of optimization mathematical models of real applications. This is the main reason why the real optimization applications give true insights on behavior of optimization algorithms.

### 6.3. *Solving MINPACK-2 applications*

Now, we present comparisons between AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG conjugate gradient algorithms for solving five applications from MINPACK-2 test problem collection [14]. In Table 21, we present these applications, as well as the values of their parameters.

**Table 21.** Applications from MINPACK-2 collection.

| | |
|---|---|
| **A1** | *Elastic-Plastic Torsion* [29, pp. 41-55], $c = 5$. |
| **A2** | *Pressure Distribution in a Journal Bearing* [20], $b = 10$, $\varepsilon = 0.1$. |
| **A3** | *Optimal Design with Composite Materials* [30], $\lambda = 0.008$. |
| **A4** | *Steady-State Combustion* [13, pp. 292-299], [17], $\lambda = 5$. |
| **A5** | *Minimal Surfaces with Enneper conditions* [40, pp. 80-85]. |

The infinite-dimensional version of these problems is transformed into a finite element approximation by triangulation. The discretization steps are *nx* = 1000 and *ny* = 1000, thus obtaining minimization problems with 1,000,000 variables. Considering $\varepsilon^g = 10^{-6}$, then the number of iterations (#iter), or the number of function and its gradient evaluation (#fg), or the CPU time (seconds), required by AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG conjugate gradient algorithms, for solving all these applications, is given in Tables 22-24.

**Table 22.** Performances of AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG algorithms for solving applications A1 and A2. $\varepsilon^g = 10^{-6}$. CPU seconds.

| | **A1** | | | **A2** | | |
|---|---|---|---|---|---|---|
| | #iter | #fg | CPU | #iter | #fg | CPU |
| AHYBRIDM | 1113 | 1114 | 378.14 | 2845 | 2873 | 1209.13 |
| ASCALCG | 1110 | 1142 | 485.26 | 2842 | 2871 | 1473.58 |
| CG-DESCENT | 1145 | 2291 | 476.12 | 3370 | 6741 | 1838.77 |
| DESCON | 1113 | 2257 | **347.25** | 2845 | 5718 | **1122.64** |
| THREECG | 1111 | 2253 | *352.60* | 2845 | 5718 | *1140.19* |

**Table 23.** Performances of AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG algorithms for solving applications A3 and A4. $\varepsilon^g = 10^{-6}$. CPU seconds.

| | **A3** | | | **A4** | | |
|---|---|---|---|---|---|---|
| | #iter | #fg | CPU | #iter | #fg | CPU |
| AHYBRIDM | 4701 | 4738 | 2876.92 | 1413 | 1451 | *2050.96* |
| ASCALCG | 4701 | 4854 | 3362.16 | 1412 | 1451 | 2192.64 |
| CG-DESCENT | 4814 | 9630 | 3960.59 | 1802 | 3605 | 3796.39 |
| DESCON | 4693 | 9425 | *2715.07* | 1413 | 2864 | **2003.78** |
| THREECG | 4478 | 9045 | **2641.22** | 1413 | 2864 | 2059.80 |

**Table 24.** Performances of AHYBRIDM, ASCALCG, CG-DESCENT, DESCON and THREECG algorithms for solving application A5. $\varepsilon^g = 10^{-6}$. CPU seconds.

| | **A5** | | |
|---|---|---|---|
| | #iter | #fg | CPU |
| AHYBRIDM | 1265 | 1293 | 600.54 |
| ASCALCG | 1280 | 1323 | 729.97 |
| CG-DESCENT | 1225 | 2451 | 756.21 |
| DESCON | 1277 | 2576 | **568.06** |
| THREECG | 1298 | 2619 | *582.29* |

Subject to the CPU time metric the **first,** the *second* and the third places of algorithms in Tables 22-24 are shown in **bold**, *italic* and underline, respectively. The first place is gained by DESCON being the fastest algorithm for applications A1, A2, A4 and A5.

## 7. Conclusions

Conjugate gradient algorithms have been subjected to intensive theoretical and computational developments for over 60 years. The main ingredients used in these developments include: scaled memoryless BFGS preconditioning (Perry [41], Shanno [47], Andrei [9]); restarting the iterations (Beale [16], Powell [45], Birgin and Martínez [18]); acceleration of iterations (Andrei [6]); hybridization by convex combination of classical conjugate gradients (Andrei [8]); guaranteed sufficient descent condition and conjugacy conditions (Hager and Zhang [32], Andrei [12]).

In this paper we have presented a comprehensive numerical study on efficiency and robustness of the most well-known eight conjugate gradient algorithms for solving large-scale nonlinear unconstrained optimization problems of different complexities and structures of the Hessian matrix. Both the artificially test problems and real nonlinear optimization applications have been included in this study. While the artificially test problems lead to partial conclusions, the real nonlinear optimization applications give more true insights on performances of optimization algorithms.

Detailed and meticulous numerical evaluation based on the performance profiles was applied to the comparisons of the algorithms showing that all of them are able to solve a large variety of large-scale unconstrained optimization problems. In our analysis all the problems for which two different algorithms found different function values are removed. We have the computational evidence that the threshold parameter $\varepsilon^f$ deciding that an algorithm found a solution or not does not have a great influence of the performance profiles of efficiency or robustness.

At least for this collection of 800 artificially unconstrained optimization test problems the CPU time performance profile for DESCON was higher than those of HS, PRP, ASCALCG, CONMIN, AHYBRIDM, CG-DESCENT and THREECG. The second best performance in the time metric was achieved by THREECG. It seems that the conjugate gradient algorithms satisfying both the sufficient descent condition and the conjugacy condition are the best. Apparently, introducing of the second order information in conjugate gradient algorithms like CONMIN, ASCALCG and AHYBRIDM does not have too much significance in efficiency or robustness of these algorithms. Additionally, hybridization by convex combination of classical conjugate gradient algorithms does not lead us to more efficient or more robust algorithms. Concerning the efficiency, due to its highly accurate procedure for step length computation, CG-DESCENT is the best conjugate gradient algorithm, especially for solving large-scale unconstrained optimization problems with structured Hessian matrix. The second place is taken by DESCON. For solving problems for which the Hessian matrix is full (unstructured), DESCON remains to be the best both subject to efficiency and robustness. Concerning the robustness DESCON is by far the most robust, followed by THREECG and followed by ASCALCG. For solving large-scale real nonlinear unconstrained optimization applications, DESCON is the fastest conjugate gradient algorithm.

All in all we can conclude that conjugate gradient algorithms represent one of the most important mathematical optimization technologies able to solve both structured and unstructured large-scale unconstrained optimization problems and applications.

## References

[1] Andrei, N., *An acceleration of gradient descent algorithm with backtracking for unconstrained optimization.* Numerical Algorithms, 42 (2006), 63-73.

[2] Andrei, N., *Scaled conjugate gradient algorithms for unconstrained optimization.* Computational Optimization and Applications, 38 (2007), 401-416.

[3] Andrei, N., *A hybrid conjugate gradient algorithm for unconstrained optimization as a convex combination of Hestenes-Stiefel and Dai-Yuan.* Studies in Informatics and Control**,** vol.17, no.1, March 2008, pp. 55-70.

[4] Andrei, N., *Another hybrid conjugate gradient algorithm for unconstrained optimization.* Numer. Algorithms 47 (2008) 143-156.

[5] Andrei, N., *An unconstrained optimization test functions collection.* Advanced Modeling and Optimization, 10 (2008), pp. 147-161.

[6] Andrei, N., *Acceleration of conjugate gradient algorithms for unconstrained optimization.* Applied Mathematics and Computation, 213 (2009), 361-369.

[7] Andrei, N., *A hybrid conjugate gradient algorithm for unconstrained optimization.* J. Optim. Theory Appl. 141 (2009) 249-264.

[8] Andrei, N., *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization.* Numerical Algorithms, 54 (2010), 23-46.

[9] Andrei, N., *Accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization.* European Journal of Operational Research, 204 (2010) 410-420.

[10] Andrei, N., *A simple three-term conjugate gradient algorithm for unconstrained optimization.* Journal of Computational and Applied Mathematics, 241 (2013), 19-29.

[11] Andrei, N., *On three-term conjugate gradient algorithms for unconstrained optimization.* Applied Mathematics and Computation, 219 (2013) 6316-6327.

[12] Andrei, N., *Another conjugate gradient algorithm with guaranteed descent and conjugacy conditions for large-scale unconstrained optimization.* Journal of Optimization Theory and Applications, DOI: 10.1007/s10957-013-0285-9

[13] Aris, R., *The mathematical theory of diffusion and reaction in permeable catalysts.* Oxford, 1975.

[14] Averick, B.M., Carter, R.G., Moré, J.J., Xue, G.L. *The MINPACK-2 test problem collection.* Mathematics and Computer Science Division, Argonne National Laboratory, Preprint MCS-P153-0692, June 1992.

[15] Babaie-Kafaki, S., *A note on the global convergence theorem of the scaled conjugate gradient algorithm proposed by Andrei,* Computational Optimization and Applications, 52 (2012) 409-414.

[16] Beale. E.M.L., *A derivation of conjugate gradients.* In F.A. Lootsma (Ed.), Numerical Methods for Nonlinear Optimization, Academic Press, London, 1972, 39-43.

[17] Bebernes, J., Eberly, D., *Mahematical problems from combustion theory.* Applied Mathematical Sciences 83, Springer-Verlag, 1989.

[18] Birgin, E., Martínez, J.M., *A spectral conjugate gradient method for unconstrained optimization.* Appl. Math. Optim. 43 (2001) 117-128.

[19] Bongartz, I., Conn, A.R., Gould, N.I.M., Toint, Ph.L., *CUTE: constrained and unconstrained testing environment.* ACM Trans. Math. Softw. 21 (1995), 123-160.

[20] Cimatti, G., *On a problem of the theory of lubrication governed by a variational inequality.* Appl. Math. Potim., 3 (1977) 227-242.

[21] Daniel, J.W., *The conjugate gradient method for linear and nonlinear operator equations.* SIAM J. Numer. Anal., 4 (1967) 10-26.

[22] Dai, Y.H., Yuan, Y., *A nonlinear conjugate gradient method with a strong global convergence property*, SIAM J. Optim., 10 (1999), pp. 177-182.

[23] Dai, Y.H., Yuan, Y., *An efficient hybrid conjugate gradient method for unconstrained optimization.* Ann. Oper. Res. 103 (2001), 33-47.

[24] Dolan, E.D., Moré, J.J., *Benchmarking optimization software with performance profiles*, Math. Programming **91**, (2002), 201-213

[25] Filip. F.G. *Sisteme suport pentru decizii.* Editura Tehnică, Bucharest, 2004. (Editia II completată şi revizuită, 2007.)

[26] Fletcher, R. and Reeves, C.M., *Function minimization by conjugate gradients* Comput. J. **7**, 149-154 (1964)

[27] Fletcher, R., *Practical Methods of Optimization, vol. 1: Unconstrained Optimization*, John Wiley & Sons, New York, 1987.

[28] Gilbert, J.C. Nocedal, J., *Global convergence properties of conjugate gradient methods for optimization,* SIAM J. Optim., 2 (1992), pp. 21-42.

[29] Glowinski, R., *Numerical Methods for Nonlinear Variational Problems.* Springer-Verlag, Berlin, 1984.

[30] Goodman, J., Kohn, R., Reyna, L., *Numerical study of a relaxed variational problem from optimal design.* Comput. Methods Appl. Mech. Engrg., 57, 1986, pp.107-127.

[31] Griewank, A., Toint, Ph.L., *Partitioned variable metric update for large structured optimization problems.*Numer. Math., 39 (1982), 119-137.

[32] Hager, W.W. and Zhang, H., *A new conjugate gradient method with guaranteed descent and an efficient line search*, SIAM Journal on Optimization, Vol. 16, pp. 170-192, 2005.

[33] Hestenes, M.R., Stiefel, E.L.: Methods of conjugate gradients for solving linear systems. J. Research Nat. Bur. Standards 49 (1952), 409-436.

[34] Hu, Y.F., Storey, C., *Global convergence result for conjugate gradient methods.* J. Optim. Theory Appl. 71 (1991) 399-405.

[35] Li, G., Tang, C., Wei, Z., *New conjugacy condition and related new conjugate gradient methods for unconstrained optimization.* J. Comput. Appl. Math. 202 (2007), pp.523-539.

[36] Liu, Y., Storey, C., *Efficient generalized conjugate gradient algorithms, Part 1: Theory.* J. Optim. Theory Appl., 69 (1991) 129-137.

[37] Moré, J.J., Wild, S.M., Benchmarking derivative-free optimization algorithms. SIAM J. Optim. 20, (2009), 172-191.

[38] Nash, S.G., Nocedal, J., *A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization.* SIAM J. Optimization, 1 (1991), pp.358-372.

[39] Noether, E., *Invariante Varlationsprobleme.* Nach. D. Könighche Gesellschaft der Wissenschaften zu Göttingen, Math-phys. Klasse, 1918, 235-257.

[40] Nitsche, J.C.C. *Lectures on minimal surfaces.* Vol.1, Cambridge University Press, 1989.

[41] Perry, A., *A class of conjugate gradient algorithms with a two step variable metric mamory.* Discussion Paper No. 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University, 1977.

[42] Polak, E., Ribière, G., *Note sur la convergence de directions conjuguée,* Rev. Francaise Informat Recherche Operationelle, 3e Année 16 (1969) 35-43.

[43] Polyak, B.T., *The conjugate gradient method in extreme problems.* USSR Comp. Math. Math. Phys. **9**, 94-112 (1969)

[44] Powell, M.J.D., *Nonconvex minimization calculations and the conjugate gradient method.* Numerical Analysis (Dundee, 1983), Lecture Notes in Mathematics, Vol. 1066, Springer, Berlin, 1984, pp.122-141.

[45] Powell, M.J.D., *Restart procedures of the conjugate gradient method.* Mathematical Programming, 2 (1977), pp. 241-254.

[46] Rădulescu, C.Z., Rădulescu, M., *A Decision Support Tool Based on a Portfolio Selection Model for Crop Planning under Risk*, Studies in Informatics and Control, ISSN 1220-1766, vol. 21 (4), pp. 377-382, 2012.

[47] Shanno, D.F., *On the convergence of a new conjugate gradient algorithm.* SIAM J. Numer. Anal., 15 (1978), pp. 1247-1257.

[48] Shanno, D.F. *Conjugate gradient methods with inexact searches.* Mathematics of Operations Research, vol.3, No.3, (1978), 244-256.

[49] Shanno, D.F., Phua, K.H., *Algorithm 500, Minimization of unconstrained multivariate functions,* ACM Trans. on Math. Soft., 2 (1976) 87-94.

[50] Touati-Ahmed, D., Storey, C., *Efficient hybrid conjugate gradient techniques.* J. Optim. Theory Appl. 64 (1990) 379-397.

[51] Wolfe, P., *Convergence conditions for ascent methods,* SIAM Rev., Vol. 11, pp.226-235, 1968.

[52] Wolfe, P., *Convergence conditions for ascent methods, (II): some corrections.* SIAM Review 13 (1971) 185-188.

**November 7, 2013**