# Concurrent nomadic and bundle search: A class of parallel algorithms for local optimization

C. Voglis[1,2], D. G. Papageorgiou[3], and I. E. Lagaris[2]

[1] Nodalpoint Systems LTD
Athens, Greece
[2] Department of Computer Science
University of Ioannina, P.O. BOX 1186
GR–45110, Ioannina, Greece
[3] Department of Materials Science and Engineering
University of Ioannina, P.O. BOX 1186
GR–45110, Ioannina, Greece

**Abstract.** We present a family of algorithms for local optimization that exploit the parallel architectures of contemporary computing systems to accomplish significant performance enhancements. This capability is important for demanding real time applications, as well as, for problems with time–consuming objective functions. The proposed concurrent schemes namely *nomadic* and *bundle* search are based upon well established techniques such as quasi-Newton updates and line searches. The parallelization strategy consists of (a) distributed computation of an approximation to the Hessian matrix and (b) parallel deployment of line searches on different directions (bundles) and from different starting points (nomads). Preliminary results showed that the new parallel algorithms can solve problems in less iterations than their serial rivals.

**Keywords:** parallel local search, nomadic search, concurrent search, nested parallelism, line search, quasi-Newton

## 1 Introduction

Optimization has broad practical applications in engineering, science and management. Many of these may either have expensive function evaluations or require real–time response. For example we refer to aircraft design, molecular modeling, space trajectory planning, optimal sea routing etc. High performance parallel computing can provide powerful tools for solving optimization problems.
Nowadays the computing power of a single core CPU has reached a premium that is improving at a very slow pace. The direction for performance enhancement has turned from pushing the CPU clock higher up, towards creating multi-core parallel architecture systems. However these hardware developments alone cannot change the picture overnight. Suitable software must be developed based on algorithms that can exploit the parallel features of the hardware in order to reach the desired performance levels. In critical real time applications, an untimely (delayed)

result is at best useless if not costly or catastrophic. In addition, large scale tough problems with expensive objectives, left aside or abandoned as hopeless goals due to the extremely long computational times they required, are now being reconsidered in the light of parallel processing.
Parallelizing a sequential algorithm usually results in minor performance gains and often the new algorithm is not equivalent to the original. However, an algorithm designed afresh aiming to exploit parallelism, is naturally expected to attain high levels of performance. Monte–Carlo methods are inherently parallel and their implementation on parallel systems is quite straightforward. Global optimization methods are also parallelizable rather easily due to the nature of the problem itself. The case of local optimization is quite hard and indeed very few inherently parallel methods exist. Most of them are advertised as parallel by adopting a parallel linear system solver. Algorithms of that kind are not genuinely parallel optimization algorithms, since they treat in parallel only the bookkeeping operations and not the calls to the objective function. The work of Chen and Han [1], describes a method called "Parallel Quasi-Newton", which handles a special case where the objective function is partially separable and hence updating conjugate subspaces is effective for large scale sparse optimization problems. In the article of Chen, Fel and Zheng [2], a "Parallel Quasi-Newton algorithm" is presented that divides the processors in two groups that operate asynchronously and avoids updating the Hessian at every iteration. However the gain, if any, in the case of time consuming objectives is minimal, since the main cost is not in updating an $n \times n$ matrix. Byrd, Schnabel and Shultz [3], discuss a parallelization scheme for the BFGS method based on estimating the gradient and part of the Hessian by finite differencing (which lends it self to extensive parallel computation) and in addition on parallel linear algebra solvers. A similar philosophy is followed for the Newton method, in the work by Conforti and Musmanno [4]. Earlier in [5], Laarhoven presented a method that exploits parallel calls to estimate via updating, (*and not via finite differencing*) the inverse Hessian matrix. His work is based upon a *1973* NASA–report by Straeter [6], that indeed capitalizes on the capabilities offered by parallel processing systems. Phua et al. [7], describe a method where line searches are applied in parallel to several descent directions produced via different Hessian updates. Among the various schemes the SR1, BFGS and the Biggs [8] updates are being considered. An interesting review appeared in 1995 by Schnabel [9], commending on the prospects of parallel non-linear optimization in a broader framework where also the field of global optimization was considered.
In the present article we present optimization algorithms suitable for execution on parallel systems. Our main focus is to accelerate the solution process without much concern for the amount of utilized resources such as the number of processors/cores, memory size, disc space, communication switches etc. One line we pursue is based on a population of $M$ neighbouring points $x_i$ which are used to obtain, via SR1 updating, approximations to the corresponding $M$ Hessians. Next, from each of these $M$ points, a line search is started in the direction determined by:

$$h_i = -B_i^{-1} g_i \equiv -H_i g_i, \quad \forall i = 1, 2, \ldots, M$$

$B_i$, $H_i$ and $g_i$ denote a modified Hessian, its inverse and the gradient at the point $x_i$ correspondingly. From the new points that emerged after the line–searches, we pick the one with the lowest function value and repeat the process anew; i.e. we pick $M-1$ additional points in its neighbourhood, estimate the corresponding $(M)$ Hessian matrices and so on so forth, until a termination criterion prevails.
A second approach comes from noticing that trust region methods solve a modified Hessian problem $(B+aI)h=-g$, where $a$ is a parameter determined by the constraint $|h(a)|\leq\rho$, $\rho$ being a proper trust radius controlled externally according to the quality of the quadratic local fit to the objective function.
From the current point we calculate $M$ directions $h_i$, by picking $M$ values $a_i$, for the parameter $a$ inside a proper range. In the next step, a line search is started along each direction $h_i$. From the resulting new points, the best one is selected as the current point. This process is repeated until a convergence criterion is satisfied.
The detailed procedures are described in section (2). Benchmarking experiments have been performed and the results are presented in section (3). Conclusions and directions for further research are contained in section (4).

## 2 Algorithmic presentation

In this section we describe in detail the proposed algorithmic schemes. We start with the nomadic search and then we analyse the bundle search. We conclude the section with the presentation of a nested combined scheme.

### 2.1 Concurrent nomadic search

Concurrent nomadic search's main iteration step consists of four basic operations:

- Definition of a nomadic group, i.e. a set of $M$ points $x_1, x_2, \cdots, x_M$.
- Estimation of a positive definite approximation to the corresponding Hessians $B_i \approx \nabla^2 f(x_i),\ i=1,\ldots,M$.
- Solution of the linear systems: $B_i h_i = -\nabla f(x_i)$, to obtain search directions $h_i,\ i=1,\ldots,M$.
- Application of $M$ line search procedures to compute the new points as: $x_i+\alpha_i h_i,\ i=1,\ldots,M$.

The algorithm starts by creating a set of $M$ points $x_1, x_2, \cdots, x_M$ relatively close to each other and randomly chosen in the vicinity of $x_1$, which is considered to be the starting point. Each point is assigned a Hessian matrix $B_i,\quad i=1,\ldots,M$ which is estimated via SR1 updates from the rest $M-1$ points. The SR1 update formula shown in Eq. (1) has been already used for estimating a Hessian from neighboring points [10, 11].

$$B_i' = B_i + \frac{(y-B_i s)\,(y-B_i s)^\top}{(y-B_i s)^\top s} \qquad (1)$$
$$s = x_i - x_j,\quad y = \nabla f(x_i) - \nabla f(x_j)$$

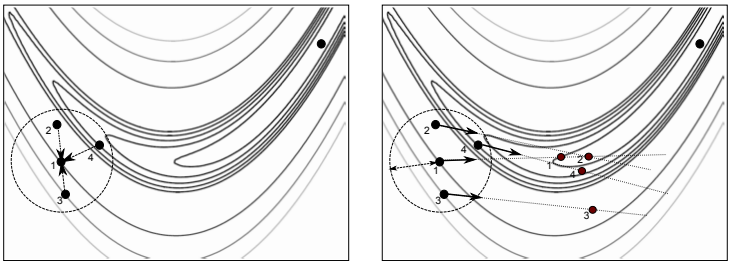
Equation (1) is applied for every point $x_i$ using information communicated by the rest $M-1$ points. This procedure can be performed concurrently at the extra communication cost of broadcasting location ($x_i$) and gradient ($\nabla f(x_i)$) information. Since the SR1 update does not maintain positive definiteness, we modify each Hessian matrix using a variant of Choleski decomposition. The Hessian is first decomposed into Choleski $LDL^T$ factors and if the diagonal matrix $D$ contains negative elements, it is modified so as to enforce positive definiteness. The search direction $h_i$, is determined by replacing the estimated Hessian $B_i$, with a convex combination $B_i' \equiv (1-\mu_i)B_i + \mu_i I$, with $\mu_i \in [0,1]$, that creates directions that are Newton–dominant for low values of $\mu_i$ and gradient–dominant for high values of $\mu_i$. Note that $\mu_i$ is calculated so as to favour a Newton–dominant direction for points with a relatively low objective value, and a gradient–dominant direction for points with a relatively high objective value, as indicated by relation (2).

$$\mu_i = \frac{f(x_i) - F_s}{F_b - F_s}, \quad \forall i = 1, 2, \cdots, M \tag{2}$$

where $F_s = \min f(x_i)$ and $F_b = \max f(x_i)$. The algorithm then performs concurrently $M$ line searches along the $h_i$ directions and computes $M$ new points $x_i + \lambda_i h_i$, where $\lambda_i$ is the step calculated by the line search procedure. From that point onwards, concurrent nomadic search can either keep these new points and repeat the procedure or maintain the best point and resample $M-1$ points anew (*periodic reset*).
In Figure (1) we provide a two dimensional illustration of the basic steps and in Algorithm (1) we lay out a detailed description. We consider that the function and gradient evaluations as well as the line searches are the time consuming parts of the algorithm. If $M$ computational resources are available the concurrent execution of the nomadic search can be performed in two steps: (i) computing the function and its gradient and (ii) performing the line search. The line search procedure contains a small (bounded) number of successive function and gradient evaluations.
Nomadic search differs from a Newton method that estimates the Hessian with SR1 updates, in that not only one, but $M$ Hessians are being estimated and $M$ line–search procedures are applied. Since the extra effort is undertaken in parallel, and assuming the availability of $M$ cores, there is no time surcharge. Note also that the Hessians are further transformed via the convex combination with the identity matrix, so that the resulting search directions are properly biased towards the gradient or the Newton direction, as dictated by the respective local values of the objective function. Another important feature of the proposed algorithm is that it does not require $O(n)$ points for approximating the Hessian,in contradistinction to Straeter's approach [6] or to numerical differentiation of the gradient.

### 2.2 Bundle search algorithm

Bundle search maintains a single point and a set of $N$ descent directions (the bundle) originating from it, along each of which a line–search is



(a) Communicating for SR1 updates

(b) $M$ line searches in parallel

**Fig. 1.** Nomadic search algorithm 2-D illustration

**Algorithm 1:** Nomadic search algorithm

**Input**: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of points: $M$; , small radius: $R$, a small number $\epsilon > 0$

**Output**: Best detected solution: $x^*$, $f(x^*)$.

Select at random $M-1$ points, $x_2, x_3, \cdots, x_M$, such that $|x_1 - x_j| \leq R, \quad \forall j = 2, \cdots, M$.

Calculate $f_i = f(x_i), \quad g_i = \nabla f(x_i) \quad \forall i = 1, 2, \cdots, M$ **(in parallel)**

**if** *termination* **then**
  $x^* = x_1 \quad f(x^*) = f_1$
  **return**
**end**

**for** $i = 1, 2, \cdots, M$ **do**
  Use SR1 updates from all other points $x_j$ with $j \neq i$, to obtain an approximate Hessian $B_i \approx \nabla^2 f(x_i)$.
  Decompose (Choleski) $B_i = L_i D_i L_i^T$, and modify $D_i$ so as to render $B_i$ positive definite.
**end**

Calculate $F_s = \min_i\{f_i\}$ and $F_b = \max_i\{f_i\}$

Calculate $\mu_i = (1-\epsilon)\dfrac{f_i - F_s}{F_b - F_s}, \quad \forall i = 1, 2, \cdots, M$ **(in parallel)**

Solve $[(1-\mu_i)B_i + \mu_i I]\, h_i = -g_i, \quad \forall i = 1, 2, \cdots, M$ **(in parallel)**

Apply a line–search $\forall i = 1, 2, \cdots, M$ as: **(in parallel)**
$\lambda_i^* = \arg\min_\lambda f(x_i + \lambda h_i)$
Set $f_i = f(x_i + \lambda_i^* h_i)$ (already calculated during the line–search).

Set: $x_i \leftarrow x_i + \lambda_i^* h_i, \ \forall i = 1, 2, \cdots, M$ **(in parallel)**

Find the index $k$ for which $f_k = \min_i\{f_i\}, \ \forall i = 1, 2, \cdots, M$

Swap $x_1$ and $x_k$

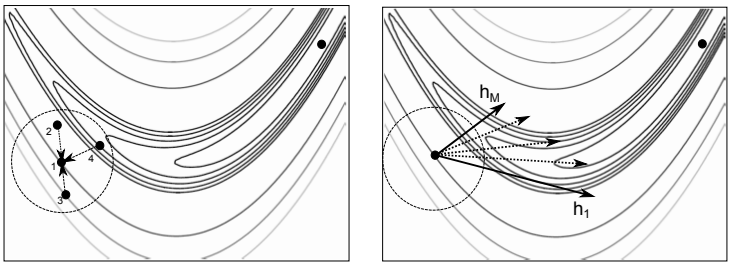
**if** *periodic reset* **then**
  Repeat from Step 1
**else**
  Repeat from Step 2
**end**

to be concurrently applied. The algorithm begins with an estimation of the Hessian matrix, using the same technique as in the case of Nomadic search. The $N$ descent directions are calculated by solving in parallel, $N$ linear systems of the form:

$$[(1-\mu_i)B + \mu_i I]\, h_i = -g, \quad \forall i = 1, 2, \ldots, N$$

where the quantities $\mu_i$ are appropriately chosen in $[0, 1]$ The bundle contains directions that are biased towards the steepest descent for large $\mu_i$ and towards the Newton direction for small $\mu_i$. The rationale behind this choice is to exploit the steepest descent if the current point is far from the minimum, and the Newton direction if it is close to it. In Figure 2 we illustrate on the left side, information communication for Hessian approximation, and on the right side the extend of the bundle. In Figure 2(b) direction $h_1$ corresponds to the Newton $(-B^{-1}g)$ while $h_N$ to

the steepest descent ($-g$) direction. In two dimensions the bundle resembles a uniform fan of descent directions between $-g$ and $-B^{-1}g$. In the case of problems of higher dimensionality, the directions of the bundle are not coplanar. After having applied the line searches, the bundle algorithm keeps the point $x' = x + \lambda_J h_J$ with the lowest function value. The next iteration involves random sampling of $M-1$ points around the kept one and estimation of the new Hessian matrix via $M-1$ SR1 updates. A complete description of the bundle search is presented in Algorithm 2. Similar to the nomadic search, bundle search performs two time consuming tasks. Function and gradient calculation for the SR1 updating of the Hessian, and the line searches. Assuming again that we do have $N$ processing units available, a single iteration of the bundle search costs as much as a single function plus gradient evaluation and a line search. The communication costs are reduced in comparison to nomadic search, since in this scheme we have to update only one Hessian matrix.




(a) Communicating for single SR1 update

(b) Following $M$ directions in parallel

**Fig. 2.** Bundle search 2-D example

**Algorithm 2:** Bundle search algorithm

**Input**: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of directions: $N$; number of points: $M$; small radius: $R$, a small number $\epsilon > 0$
**Output**: Best detected solution: $x^*$, $f(x^*)$.

**1** **if** *termination* **then**
  $x^* = x$  $f(x^*) = f$
  **return**
**end**

**2** Decompose (Choleski) $B = LDL^T$, and modify $D$ so as to render $B$ positive definite.

**3** Solve $[(1-\mu_i)B + \mu_i I]\, h_i = -g, \quad \forall i = 1, 2, \cdots, N$ **(in parallel)**
where $\mu_i \in [0, 1], \ \forall i = 1, \cdots, N$.

**4** Apply a line–search $\forall i = 1, 2, \cdots, N$ as: **(in parallel)**
$\lambda_i^* = \arg\min_{\lambda} f(x + \lambda h_i)$
Set $f_i = f(x + \lambda_i^* h_i)$ (already calculated during the line–search).

**5** Find the index $k$ for which $f_k = \min_i \{f_i\}, \ \forall i = 1, 2, \cdots, N$

**6** Set: $x \leftarrow x + \lambda_k^* h_k$.

**7** Set $f \leftarrow f_k$, and $g \leftarrow \nabla f(x_k)$

**8** Select at random $M-1$ points, $x_2, x_3, \cdots, x_M$, such that
$|x - x_j| \leq R, \quad \forall j = 2, \cdots, M$.

**9** Calculate the Hessian approximation at $x$ via SR1 using the adjacent points $x_2, x_3, \cdots, x_M$.

**10** Go to 1

### 2.3 Nested nomadic and bundle search

In order to take advantage of both nomadic and bundle search methodologies, we propose a nested scheme that involves an outer iteration following nomadic search and an inner iteration with bundles of directions. In this scheme we maintain $M$ points $x_i,\ i = 1, \ldots, M$ and perform all–to–all SR1 updates to approximate $M$ Hessian matrices $B_i,\ i = 1, \ldots, M$. From each point we then define $N$ directions by following the bundle search methodology. The points $x_i,\ i = 1, \ldots, M$ of the next iteration are taken from the results of the $N$ line searches. In this nested scheme we define $M \times N$ line search tasks in parallel (Step 8 of Algorithm 3). In Algorithm 3 we present the nested scheme.
The nested scheme is even more demanding on computational resources, but we expect to further reduce the length of execution times. Efficient implementation of nested schemes need advanced runtime support [12, 13] which is currently available and supported by OpenMP API.

**Algorithm 3:** Nested nomadic and bundle scheme

**Input**: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; number of points: $M$; number of directions $N$ small radius: $R$, a small number $\epsilon > 0$
**Output**: Best detected solution: $x^*,\ f(x^*)$.

1 Select at random $M-1$ points, $x_2, x_3, \cdots, x_M$, such that $|x_1 - x_j| \leq R, \quad \forall j = 2, \cdots, M$.

2 Calculate $f_i = f(x_i), \quad g_i = \nabla f(x_i) \quad \forall i = 1, 2, \cdots, M$ **(in parallel)**

3 **if** *termination* **then**
  $x^* = x_1 \quad f(x^*) = f_1$
  **return**
**end**

4 Estimate $B_i \approx \nabla^2 f(x_i)$ and decompose so as to render it positive definite $\forall i = 1, 2, \cdots, M$ **(in parallel)**

5 Calculate $F_s = \min_i \{f_i\}$ and $F_b = \max_i \{f_i\}$ needed in $\mu_i = (1-\epsilon)\dfrac{f_i - F_s}{F_b - F_s}$

6 Solve $\left[(1-\mu_i)B_i + \mu_i I\right] h_i = -g_i, \quad \forall i = 1, 2, \cdots, M$ **(in parallel)**
// For all points
**for** $i = 1, 2, \ldots, M$ **do**
  // Begin bundle search
  **for** $j = 1, 2, \ldots, N$ **do**
7  Solve $\left[(1-\tilde{\mu}_j)B_i + \tilde{\mu}_j I\right] h_j = -g_i$, where $\tilde{\mu}_j = (1-\epsilon)\dfrac{j-1}{N-1}$.
8  Apply a line–search $\lambda_j^* = \arg\min_{\lambda} f(x_i + \lambda h_j)$
   Set $\bar{f}_j = f(x_i + \lambda_j^* h_j)$.
  **end**
9  Find the index $k$ for which $f_k = \min_j \{\bar{f}_j\},\ \forall j = 1, 2, \cdots, M$
10  Set: $x_i \leftarrow x_i + \lambda_k^* h_k, \quad f_i \leftarrow f_k$, and $g_i \leftarrow \nabla f(x_k)$.
**end**
11 Find the index $k$ for which $f_k = \min_i \{f_i\},\ \forall i = 1, 2, \cdots, M$

12 Swap $x_1$ and $x_k$

13 **if** *fresh restart* **then**
  Repeat from Step 1
**else**
  Repeat from Step 2
**end**

## 3 Numerical experiments

We have implemented all parallel algorithms in Matlab in order to measure their effectiveness with respect to the number of iterations. It is not

a true parallel implementation, e.g using Matlab's parallel toolbox, but it is used as a proof of concept for the efficiency of the nomadic, bundle and the combined search. The comparison is made against two well known quasi–Newton sequential algorithms: BFGS and SR1, each with a line search. All methods in the comparison table share the same line search code. The basic computational cost of these methods per iteration is one function and gradient evaluation and one line search. Considering the communication costs negligible with respect to function/gradient evaluation, it is plausible to claim that one iteration of a sequential quasi–Newton algorithm and that of our proposed parallel methodologies, take the same amount of time. Hence in this study we compare the number of iterations to provide a proof–of–concept, expecting that the estimated speedup is close to that of a real parallel implementation.
We used a part of the well established Moré optimization test functions [14] and some instances from the Dixon-Szego test set. For every test function we report the number of iterations each algorithm performed in order to reach the target minimizer starting from a prespecified point. All numbers reported are averages of twenty runs with different random seeds. We have experimented with values of $M = 8, 16, 48, 64$. In Table 1 we present relative speedups, with respect to serial BFGS and DFP methods, up for the cases $M = 8, 16$ and in Table 2 for the cases $M = 48, 64$. In the last row we present the average speedup for all test functions.
By inspecting the result tables we can see that the proposed parallel algorithms can result in 6 times less iterations than their serial competitors which represent the state–of–the–art in the field on numerical optimization. A closer look reveals that in some cases (eg. Quadratic 50, 100, Rosenbrock 10, Brown and Dennis, Trigonometric) the speed up in terms of iterations is noteworthy when $M$ is greater than 16. These results indicate that with a proper implementation and a sufficiently heavy objective function evaluation, nomadic, bundle and nested concurrent searches may be used to accelerate convergence by a substantial factor. Is is obvious though that the speedup does not scale well with $M$. This can be attributed to the fact that near the minimum all directions tend to coincide with the Newton direction, hence in these last iterations the alternatives offer almost no advantage. Dynamic allocation of computational resources and batch optimization schemes may increase the overall ratio.

## 4 Conclusions

We have presented three parallel methods for the problem of local optimization with line searches. A multipoint or concurrent nomadic search, a multi–direction or concurrent bundle search, and a combination of the two. All use SR1 updates from randomly sampled points to estimate required Hessian matrices. Preliminary simulation results clearly indicate that they may significantly reduce the number of iterations, and consequently the overall computational time, needed by well established and widely used serial rival methods.

**Table 1.** Speedup in iterations for $M = 8, 16$

| | | MP (8) | | MD(8) | | Nested(4x2) | | MP (16) | | MD(16) | | Nested(4x4) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS |
| Heli | 3 | 1.40 | 1.80 | 1.75 | 2.25 | 1.11 | 1.42 | 1.62 | 2.08 | 1.50 | 1.93 | 1.40 | 1.80 |
| Gaussian | 3 | 1.86 | 1.71 | 1.63 | 1.50 | 1.86 | 1.71 | 2.17 | 2.00 | 2.17 | 2.00 | 2.17 | 2.00 |
| Var Dim. | 2 | 2.00 | 2.00 | 2.00 | 2.00 | 1.60 | 1.60 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| Watson | 2 | 1.60 | 1.80 | 1.60 | 1.80 | 1.60 | 1.80 | 1.60 | 1.80 | 1.60 | 1.80 | 1.60 | 1.80 |
| Brown | 4 | 16.00 | 55.56 | 24.00 | 83.33 | 12.00 | 41.67 | 16.00 | 55.56 | 18.00 | 62.50 | 16.00 | 55.56 |
| Gulf | 3 | 5.81 | 1.33 | 5.55 | 1.27 | 6.78 | 1.56 | 6.42 | 1.47 | 7.63 | 1.75 | 7.18 | 1.65 |
| Trigon | 2 | 1.67 | 1.17 | 2.00 | 1.40 | 1.67 | 1.17 | 2.00 | 1.40 | 2.00 | 1.40 | 2.00 | 1.40 |
| Rosen. | 2 | 3.00 | 2.54 | 2.44 | 2.06 | 1.44 | 1.22 | 2.05 | 1.74 | 4.33 | 3.67 | 2.05 | 1.74 |
| Beale | 2 | 1.44 | 1.56 | 1.63 | 1.75 | 0.57 | 0.61 | 1.86 | 2.00 | 1.86 | 2.00 | 0.65 | 0.70 |
| Wood | 4 | 0.59 | 0.68 | 0.79 | 0.89 | 0.71 | 0.81 | 0.73 | 0.83 | 0.81 | 0.93 | 0.92 | 1.04 |
| Cheby. | 2 | 1.50 | 1.50 | 1.20 | 1.20 | 0.86 | 0.86 | 1.50 | 1.50 | 2.00 | 2.00 | 1.00 | 1.00 |
| Cubic | 2 | 3.48 | 1.30 | 3.64 | 1.36 | 1.48 | 0.56 | 3.64 | 1.36 | 5.33 | 2.00 | 1.18 | 0.44 |
| De Jong1 | 3 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| De Jong2 | 2 | 1.50 | 1.31 | 1.41 | 1.24 | 0.65 | 0.57 | 1.50 | 1.31 | 1.50 | 1.31 | 0.67 | 0.58 |
| Goldstein | 2 | 1.33 | 1.44 | 2.00 | 2.17 | 1.33 | 1.44 | 1.71 | 1.86 | 2.00 | 2.17 | 1.71 | 1.86 |
| Branin | 2 | 1.00 | 1.00 | 2.00 | 2.00 | 1.00 | 1.00 | 1.20 | 1.20 | 1.50 | 1.50 | 1.00 | 1.00 |
| Shekel5 | 4 | 2.83 | 2.50 | 3.40 | 3.00 | 2.83 | 2.50 | 2.83 | 2.50 | 3.40 | 3.00 | 3.40 | 3.00 |
| Shekel10 | 4 | 1.50 | 2.00 | 1.80 | 2.40 | 1.29 | 1.71 | 1.50 | 2.00 | 1.80 | 2.40 | 1.50 | 2.00 |
| Six hump | 2 | 6.00 | 1.60 | 7.50 | 2.00 | 4.29 | 1.14 | 6.00 | 1.60 | 7.50 | 2.00 | 5.00 | 1.33 |
| Colville | 4 | 1.57 | 2.07 | 1.29 | 1.71 | 0.69 | 0.91 | 1.22 | 1.61 | 1.47 | 1.93 | 1.10 | 1.45 |
| Bazaraa | 2 | 5.50 | 4.50 | 2.20 | 1.80 | 1.69 | 1.38 | 1.83 | 1.50 | 2.20 | 1.80 | 7.33 | 6.00 |
| Quadratic | 2 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 |
| Var Dim. | 10 | 2.78 | 2.78 | 2.78 | 2.78 | 3.13 | 3.13 | 2.78 | 2.78 | 3.57 | 3.57 | 3.13 | 3.13 |
| Watson | 10 | 2.45 | 2.20 | 1.07 | 0.96 | 1.75 | 1.57 | 3.27 | 2.93 | 1.11 | 1.00 | 2.58 | 2.32 |
| Trigon | 10 | 2.80 | 3.93 | 3.23 | 4.54 | 3.23 | 4.54 | 4.67 | 6.56 | 3.82 | 5.36 | 3.50 | 4.92 |
| Rosen. | 10 | 4.86 | 1.69 | 8.95 | 3.11 | 3.94 | 1.37 | 7.88 | 2.74 | 10.94 | 3.81 | 5.12 | 1.78 |
| Quadratic | 10 | 20.00 | 8.67 | 30.00 | 13.00 | 12.00 | 5.20 | 60.00 | 26.00 | 60.00 | 26.00 | 20.00 | 8.67 |
| Var Dim. | 50 | 2.09 | 0.36 | 1.44 | 0.25 | 1.28 | 0.22 | 1.64 | 0.29 | 2.09 | 0.36 | 1.92 | 0.33 |
| Watson | 50 | 1.45 | 0.90 | 0.47 | 0.29 | 0.74 | 0.46 | 2.44 | 1.51 | 0.34 | 0.21 | 0.95 | 0.59 |
| Trigon | 50 | 2.24 | 2.55 | 3.70 | 4.22 | 2.74 | 3.13 | 3.27 | 3.73 | 3.70 | 4.22 | 4.72 | 5.39 |
| Rosen. | 50 | 1.57 | 0.91 | 1.02 | 0.59 | 1.19 | 0.69 | 2.14 | 1.23 | 2.24 | 1.29 | 1.89 | 1.09 |
| Quadratic | 50 | 15.73 | 7.82 | 15.73 | 7.82 | 6.92 | 3.44 | 28.83 | 14.33 | 43.25 | 21.50 | 12.36 | 6.14 |
| Var Dim. | 100 | 1.57 | 43.48 | 1.89 | 52.63 | 0.92 | 25.64 | 1.29 | 35.71 | 2.77 | 76.92 | 2.57 | 71.43 |
| Watson | 100 | 1.32 | 7.35 | 0.39 | 2.16 | 0.66 | 3.69 | 1.54 | 8.55 | 0.86 | 4.78 | 0.98 | 5.46 |
| Trigon | 100 | 7.93 | 1.77 | 4.79 | 1.07 | 9.49 | 2.12 | 8.34 | 1.86 | 9.68 | 2.16 | 9.88 | 2.20 |
| Rosen. | 100 | 1.84 | 0.81 | 1.00 | 0.44 | 2.00 | 0.88 | 2.40 | 1.05 | 1.53 | 0.67 | 2.02 | 0.88 |
| Quadratic | 100 | 7.90 | 6.52 | 5.03 | 4.15 | 2.44 | 2.01 | 15.09 | 12.45 | 18.44 | 15.22 | 5.03 | 4.15 |
| **Speedup:** | | **3.84** | **5.17** | **4.20** | **6.08** | **2.75** | **3.61** | **5.65** | **5.92** | **6.46** | **7.49** | **3.80** | **5.86** |

**Table 2.** Speedup in iterations for $M = 48, 64$

| | | MP (48) | | MD(48) | | Nested(12x4) | | MP (64) | | MD(64) | | Nested(16x4) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS | SR1 | BFGS |
| Heli | 3 | 1.62 | 2.08 | 1.75 | 2.25 | 1.91 | 2.45 | 1.62 | 2.08 | 1.75 | 2.25 | 1.91 | 2.45 |
| Gauss | 3 | 2.17 | 2.00 | 1.86 | 1.71 | 2.17 | 2.00 | 2.17 | 2.00 | 2.17 | 2.00 | 2.17 | 2.00 |
| VarD | 2 | 2.00 | 2.00 | 2.67 | 2.67 | 2.00 | 2.00 | 2.00 | 2.00 | 2.67 | 2.67 | 2.67 | 2.67 |
| Wats | 2 | 1.60 | 1.80 | 2.67 | 3.00 | 1.60 | 1.80 | 1.60 | 1.80 | 2.00 | 2.25 | 2.00 | 2.25 |
| Brow | 4 | 16.00 | 55.56 | 20.57 | 71.43 | 18.00 | 62.50 | 16.00 | 55.56 | 18.00 | 62.50 | 18.00 | 62.50 |
| Gulf | 3 | 7.18 | 1.65 | 10.17 | 2.33 | 8.13 | 1.87 | 7.63 | 1.75 | 12.20 | 2.80 | 8.13 | 1.87 |
| Trigon | 2 | 2.00 | 1.40 | 1.67 | 1.17 | 2.00 | 1.40 | 2.00 | 1.40 | 2.00 | 1.40 | 2.00 | 1.40 |
| Rosen. | 2 | 3.90 | 3.30 | 3.00 | 2.54 | 1.77 | 1.50 | 3.55 | 3.00 | 3.25 | 2.75 | 2.05 | 1.74 |
| Beale | 2 | 1.86 | 2.00 | 1.63 | 1.75 | 1.18 | 1.27 | 2.17 | 2.33 | 1.63 | 1.75 | 1.00 | 1.08 |
| Wood | 4 | 0.88 | 1.00 | 0.81 | 0.93 | 1.22 | 1.39 | 0.96 | 1.09 | 0.88 | 1.00 | 1.38 | 1.56 |
| Cheby. | 2 | 1.50 | 1.50 | 1.50 | 1.50 | 1.00 | 1.00 | 1.50 | 1.50 | 1.50 | 1.50 | 1.00 | 1.00 |
| Cubic | 2 | 3.81 | 1.43 | 5.33 | 2.00 | 3.81 | 1.43 | 4.44 | 1.67 | 5.33 | 2.00 | 3.64 | 1.36 |
| Jong1 | 3 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Jong2 | 2 | 1.60 | 1.40 | 1.85 | 1.62 | 1.60 | 1.40 | 1.60 | 1.40 | 1.85 | 1.62 | 0.89 | 0.78 |
| Gold | 2 | 2.00 | 2.17 | 2.40 | 2.60 | 1.71 | 1.86 | 1.71 | 1.86 | 2.40 | 2.60 | 2.00 | 2.17 |
| Branin | 2 | 1.20 | 1.20 | 1.50 | 1.50 | 1.20 | 1.20 | 1.20 | 1.20 | 1.50 | 1.50 | 1.20 | 1.20 |
| S5 | 4 | 2.83 | 2.50 | 3.40 | 3.00 | 3.40 | 3.00 | 2.83 | 2.50 | 3.40 | 3.00 | 3.40 | 3.00 |
| S10 | 4 | 1.50 | 2.00 | 1.80 | 2.40 | 1.50 | 2.00 | 1.50 | 2.00 | 1.80 | 2.40 | 1.50 | 2.00 |
| Hump | 2 | 6.00 | 1.60 | 10.00 | 2.67 | 5.00 | 1.33 | 6.00 | 1.60 | 7.50 | 2.00 | 5.00 | 1.33 |
| Colv | 4 | 1.57 | 2.07 | 1.38 | 1.81 | 1.16 | 1.53 | 1.57 | 2.07 | 1.38 | 1.81 | 1.29 | 1.71 |
| Baza | 2 | 2.75 | 2.25 | 2.20 | 1.80 | 5.50 | 4.50 | 3.67 | 3.00 | 2.20 | 1.80 | 4.40 | 3.60 |
| Quad | 2 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 | 3.00 | 9.00 |
| VarD | 10 | 4.17 | 4.17 | 4.17 | 4.17 | 3.57 | 3.57 | 4.17 | 4.17 | 4.17 | 4.17 | 3.57 | 3.57 |
| Wats | 10 | 3.50 | 3.14 | 1.58 | 1.42 | 3.27 | 2.93 | 3.50 | 3.14 | 1.48 | 1.33 | 3.50 | 3.14 |
| Trigon | 10 | 4.20 | 5.90 | 10.50 | 14.75 | 5.25 | 7.38 | 3.82 | 5.36 | 4.67 | 6.56 | 6.00 | 8.43 |
| Rose | 10 | 8.57 | 2.98 | 12.31 | 4.28 | 9.61 | 3.34 | 7.43 | 2.58 | 12.71 | 4.42 | 10.65 | 3.70 |
| Quad | 10 | 60.00 | 26.00 | 60.00 | 26.00 | 60.00 | 26.00 | 60.00 | 26.00 | 60.00 | 26.00 | 60.00 | 26.00 |
| VarD | 50 | 1.92 | 0.33 | 1.92 | 0.33 | 2.30 | 0.40 | 2.09 | 0.36 | 1.92 | 0.33 | 2.30 | 0.40 |
| Wats | 50 | 3.45 | 2.14 | 0.83 | 0.51 | 2.22 | 1.38 | 4.00 | 2.48 | 1.16 | 0.72 | 2.63 | 1.63 |
| Trigon | 50 | 4.05 | 4.62 | 3.40 | 3.88 | 5.00 | 5.71 | 3.15 | 3.59 | 3.54 | 4.04 | 4.72 | 5.39 |
| Rose | 50 | 3.36 | 1.93 | 9.09 | 5.24 | 3.33 | 1.92 | 5.38 | 3.10 | 9.26 | 5.33 | 3.70 | 2.13 |
| Quad | 50 | 57.67 | 28.67 | 86.50 | 43.00 | 28.83 | 14.33 | 173.00 | 86.00 | 173.00 | 86.00 | 34.60 | 17.20 |
| VarD | 100 | 1.71 | 47.62 | 1.50 | 41.67 | 2.57 | 71.43 | 1.71 | 47.62 | 1.33 | 37.04 | 2.57 | 71.43 |
| Wats | 100 | 1.08 | 6.02 | 1.31 | 7.30 | 2.54 | 14.08 | 1.29 | 7.19 | 1.43 | 7.94 | 2.17 | 12.05 |
| Trigon | 100 | 11.52 | 2.57 | 20.17 | 4.50 | 20.17 | 4.50 | 13.83 | 3.09 | 16.13 | 3.60 | 16.69 | 3.72 |
| Rose | 100 | 3.97 | 1.74 | 6.33 | 2.77 | 4.18 | 1.83 | 4.03 | 1.77 | 11.90 | 5.21 | 4.50 | 1.97 |
| Quad | 100 | 33.20 | 27.40 | 55.33 | 45.67 | 15.09 | 12.45 | 41.50 | 34.25 | 55.33 | 45.67 | 20.75 | 17.13 |
| Speedup: | | **7.31** | **7.19** | **9.65** | **8.81** | **6.43** | **7.53** | **10.77** | **8.99** | **11.82** | **9.46** | **6.70** | **7.72** |

For the nomadic search algorithm the selection of points for the next iteration is an issue that needs further examination. This is a practical, yet important issue, since it may affect seriously the performance of the method. For the Hessian of the maintained point (or points) one may consider to either discard the existing information and proceed as in the initialization step using the SR1 scheme, or to continue updating the existing Hessian. Another question that may arise in the second case, is whether a periodic reset is then necessary and at what frequency.

In addition, the Hessian may also be updated right after the linesearch application, for instance via a BFGS or any other Quasi-Newton formula. It is a matter of further investigation if this will enhance the methods performance or not. Most Hessian issues referred to above, need to be investigated for the case of the bundle search as well. We intend first to implement these methods in a real parallel system (MPI multicore cluster) and then we would like to address the above important issues.

## Acknowledgments

This work is co-financed by the European Union and Greece Operational Program "Human Resources Development" - NSFR 2007-2013 - European Social Fund.

## References

1. M.-Q. Chen and S.-P. Han. A parallel quasi-Newton method for partially separable large scale minimization. *Annals of Operations Research*, **14**:195–211, 1998.
2. Z. Chen, P. Fel, and H. Zheng. A parallel quasi-Newton algorithm for unconstraint optimization. *Computing*, **55**:125–133, 1995.
3. R.H. Byrd, R.B. Schnabel, and G.A. Shultz. Parallel quasi-Newton methods for unconstrained optimization. *Mathematical Programming*, **42**:273–306, 1988.
4. D. Conforti and R. Musmanno. A parallel asynchronous Newton algorithm for unconstrained optimization. *Journal of optimization theory and applications*, **77**:305–322, 1993.
5. P.J.M. van Laarhoven. Parallel variable metric algorithms for unconstrained optimization. *Mathematical Programming*, **33**:68–81, 1985.
6. T. A. Straeter. A parallel variable metric optimization algorithm. NASA Technical Note D-7329, Hampton, VA, 1973.
7. P.K.H. Phua, W.Fan, and Y.Zeng. Self-scaling parallel quasi-Newton methods. In *Fourth International Conference on Optimization: Techniques and Applications,Australia*, 1998.
8. M. C. Biggs. A note on minimization algorithms which make use of non-quadratic properties of the objective function. *J. Inst. Maths. Apllics.*, **12**:337–338, 1973.
9. Robert B. Schnabel. A view of the limitations, opportunities, and challenges in parallel nonlinear optimization. *Parallel Computing*, **21**:875–905, 1995.
10. H Fayez Khalfan, Richard H Byrd, and Robert B Schnabel. A theoretical and experimental study of the symmetric rank-one update. *SIAM Journal on Optimization*, 3(1):1–24, 1993.
11. W Tu and RW Mayne. Studies of multi-start clustering for global optimization. *International journal for numerical methods in engineering*, 53(9):2239–2252, 2002.
12. C. Voglis, P.E. Hadjidoukas, V.V. Dimakopoulos, I.E. Lagaris, and D.G. Papageorgiou. Task-parallel global optimization with application to protein folding. In *High Performance Computing and Simulation (HPCS)*, pages 186–192. IEEE, 2011.
13. P. Hadjidoukas, C. Voglis, V. Dimakopoulos, I. Lagaris, and D.G. Papageorgiou. High-performance numerical optimization on multicore clusters. *Euro-Par 2011 Parallel Processing*, pages 353–364, 2011.
14. Jorge J Moré, Burton S Garbow, and Kenneth E Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41, 1981.