Asynchronous Parallel $(1+1)$-CMA-ES for Constrained Global Optimisation

Thomas Philip Runarsson
School of Engineering and Natural Science, University of Iceland, Hjardarhagi 26, Reykjavik, Iceland

Keywords: Constrained global optimization, Evolution strategies, Parallel Computing.

Abstract: The global search performance of an asynchrounous parallel $(1+1)$ evolution strategy using the full covariance matrix adaptation for constrained optimization is presented. Although the $(1+1)$-CMA-ES may be a poor global optimizer it will be shown that within this parallel framework the global search performance can be enhanced significantly. This is achieved even when all individual $(1+1)$ strategies use the same initial search point. The focus will be on constrained global optimization using a recently developed $(1+1)$ evolution strategy for this purpose.

1 INTRODUCTION

This paper considers the general nonlinear programming problem formulated as

$$\text{minimize} \quad f(x), \quad x = (x_1, \ldots, x_n) \in \mathcal{R}^n, \qquad (1)$$

where $f(x)$ is the objective function, $x \in \mathcal{S} \cap \mathcal{F}$, $\mathcal{S} \subseteq \mathcal{R}^n$ defines the search space bounded by the box constraints

$$\underline{x}_i \leq x_i \leq \overline{x}_i, \qquad (2)$$

and the feasible region $\mathcal{F}$ is defined by

$$\mathcal{F} = \{x \in \mathcal{R}^n \mid g_j(x) \leq 0 \quad \forall\, j\}, \qquad (3)$$

where $g_j(x)$, $j = 1, \ldots, m$, are inequality constraints.

The state of the art evolution strategy (ES) is the $(\mu/\mu_w, \lambda)$-CMA-ES with full covariance matrix adaptation (CMA), where μ/μ_w denotes the weighted averaging (recombination) of μ parents. The most successful version of the $(\mu/\mu_w, \lambda)$-CMA-ES uses multiple restarts. The $(\mu/\mu_w, \lambda)$-CMA-ES will double the population size each time it restarts (Auger and Hansen, 2005). Handling boundaries and constraints using the $(\mu/\mu_w, \lambda)$-CMA-ES is problem dependent (Hansen, 2011). Typically infeasible solution are ranked lower than the worst feasible solution, this is somewhat similar to a death penalty approach to constraint handling. The infeasible solutions are then ranked according to a penalty term, which may be a function of the constraint or the distance to a known feasible solution. The approaches suggested by Hansen in his tutorial avoid computing the objective function when a solution x is infeasible. This is for many real world applications crucial since the objective can in many cases not be computed when infeasible. When objective information can be used to bias the search towards favourable feasible regions, a very simple ES can be quite effective, as illustrated by (Runarsson and Yao, 2005).

Recently a simple but effective $(1+1)$-CMA-ES for constrained optimization was proposed by (Arnold and Hansen, 2012). The technique is a very effective local optimizer, but requires a feasible starting point. In some cases an initial solution may be supplied by the user, otherwise random sampling of the search space must be performed until a feasible search point is found. Depending on the number of constraints, their computational complexity and the size of the feasible region, this may be a time consuming procedure. Nevertheless, this is the basis for the asynchrounous parallel global optimizer presented here. In essence it is proposed that one consider multiple competing $(1+1)$-CMA-ES to produce a powerful global optimizer, opposed to extending the technique proposed by (Arnold and Hansen, 2012) to the multi-parent the $(\mu/\mu_w, \lambda)$-CMA-ES version, as they suggest and are currently working towards.

The remainder of the paper is organized as follows. Section 2 describes the $(1+1)$-CMA-ES applied, which is based on (Suttorp et al., 2009) but uses the modification in (Arnold and Hansen, 2012) for the case when an infeasible individual is generated. Section 3 introduces then the asynchrounous parallel search strategy, which is comprised of a set of $(1+1)$-CMA-ESs. The computational performance

Runarsson T..
Asynchronous Parallel (1+1)-CMA-ES for Constrained Global Optimisation.
DOI: 10.5220/0005084502660272
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 266-272
ISBN: 978-989-758-052-9

of the parallel algorithm is illustrated on two different hardware configurations in section 4. The strategy is then illustrated on some unconstrained and constrained global optimization problems in section 5. Finally, section 6 concludes the paper with a brief summary and some remarks.

2 (1+1)-CMA-ES

The (1+1) covariance matrix adaptation evolutionary strategy is a single *parent* search strategy. A single iterations of this search strategy will now be described, but the reader is referred to (Suttorp et al., 2009) for a more complete description. The parent x is replicated (imperfectly) to produce an offspring $y = x + \sigma\mathcal{N}(0,C)$, where σ is a global step size and C the covariance matrix of the zero mean Gaussian distribution. The replication is implemented as follows:

$$z = \mathcal{N}(0,I) \quad (4)$$

$$s = Az \quad (5)$$

$$y = x + \sigma s \quad (6)$$

where the covariance matrix has been decomposed into Cholesky factors $AA^\intercal$. The normally distributed random vector z is sampled from the standard normal distribution $\mathcal{N}(0,I)$. The success probability of this replication is updated by

$$\bar{p}_{succ} \leftarrow (1-c_p)\bar{p}_{succ} + c_p \mathbb{1}\left[f(y) \leq f(x)\right] \quad (7)$$

where $f(\cdot)$ if the objective (fitness) function which will be minimized. Here $\mathbb{1}[\cdot]$ is the indicator function and takes the value one if its argument is true otherwise zero. The parameter c_p is the learning rate ($0 < c_p \leq 1$) and is set to $1/12$. The initial value for $\bar{p}_{succ} = 2/11$ which is also the target success probability p^t_{succ}. Following the evaluation of the success probability the global step size is updated by

$$\sigma \leftarrow \sigma \exp\left(\frac{\bar{p}_{succ} - p^t_{succ}}{d(1-p^t_{succ})}\right) \quad (8)$$

where $d = 1 + n/2$ and n the number of objective variables. The initial global step size will be problem dependent but should cover the intended search space. All of these default parameter setting are discussed in (Suttorp et al., 2009).

If the replication was successful, that is $f(y) \leq f(x)$, then y will replace the parent search point x. Furthermore, the Cholesky factors A will be updated. Initially A and A^{-1} are set to the identity matrix and s set to 0. The update is then as follows (Suttorp et al., 2009):

1. If $\bar{p}_{succ} < p^t_{succ}$ then $s \leftarrow (1-c)s + \sqrt{c(2-c)}Az$ and set $\alpha \leftarrow (1-c_{cov})$ else $s \leftarrow (1-c)s$ and set $\alpha \leftarrow 1 - c^2_{cov}c(2-c)$.
2. Compute $w \leftarrow A^{-1}s$ and set $a = \sqrt{1 + c_{cov}\|w\|^2/\alpha}$
3. $A \leftarrow \sqrt{\alpha}A + \sqrt{\alpha}(a-1)sw^\intercal/\|w\|^2$
4. $A^{-1} \leftarrow \frac{1}{\sqrt{\alpha}}A^{-1} - \frac{1}{\sqrt{\alpha}\|w\|^2}\left(1 - 1/a\right)w\left[w^\intercal A^{-1}\right]$.

The default setting for the covariance weight factor $c_{cov} = 2/(n^2+6)$ and $c = 2/(2+n)$. A^{-1} requires $\Theta(n^2)$ time, whereas a factorization of the covariance matrix requires $\Theta(n^3)$ time. The Cholesky version of the (1+1)-CMA is therefore computationally more efficient.

Recently Arnold and Hansen provided an elegant solution to constraint handling for the (1+1)-CMA-ES. The procedure requires a feasible starting point and when the replication produces a feasible point the algorithm is equivalent to the one described above. However, when the replication produces an infeasible point the mutation strength in the directions of the normal vectors of constraint boundaries is reduced (Arnold and Hansen, 2012). This is achieved by maintaining a so-called constraint vector v_j for all constraints violated. This includes the box-constraints (upper and lower bounds on x), i.e. $j = 1,\ldots,m+2n$. The vectors for the violated constraints are updated as follows:

$$v_j \leftarrow (1-c_v)v_j + c_c Az \quad (9)$$

and initialized to zero. The parameter c_c is set by default to $1/(n+2)$ and controls the decay of information contained in the constraint vectors. For each constraint j violated the Cholesky factors are updated as follows:

$$A \leftarrow A - \frac{\beta}{n_v}\frac{v_j w_j^\intercal}{w_j^\intercal w_j^\intercal} \quad (10)$$

where $w_j = A^{-1}v_j$ and n_v is the total number of violated constraints. The parameter β, set by default to $0.1/(n+2)$, controls the size of the updates similarly to parameter c_{cov}. The computational cost are now increased since computing the inverse of A must now be done directly at a cost of $\Theta(n^3)$.

How a population of (1+1)-CMA-ES may be combined to make the search more effective will now be described.

3 ASYNCHRONOUS PARALLEL (1+1)-CMA-ES

Consider now a set of λ (1+1)-CMA-ES as individuals in a population. Furthermore, assume that one has n_t dedicated threads replicating and evaluating these

individuals. The thread's job is to pick out two available candidates i and j, where i is replicated and j is potentially overwritten. Here being "available" means that no other thread is using the candidate. In practice it is necessary to mark i and j as busy, or temporarily unavailable, hence stopping threads from writing to the same memory at the same time. The resulting behaviour of this new strategy is a competition between independent $(1+1)$-CMA evolution strategies.

The implementation presented here has similarities to a scheme described in (Runarsson, 2003). The idea being to replace the worst individuals in a continuous selection scheme (Runarsson and Yao, 2002). There an inferior j may be overwritten by a mutated version of parent i. The idea is in essence as follows. Let any thread capture any two freely available individuals from the population and lock them. Take then the better of the two and replicate as described by eqn. (4)–(6). If the resulting solution y is infeasible then release individual j and reduce the mutation strength of individual i by equations (9)–(10). Then compute the inverse of the Cholesky factors. If the solution y is feasible then continue with equations (7)–(8). If this replication produces a fitter feasible individual then overwrite the worse individual j with the better one, update the Cholesky factors and release them both. It is important to note that an individual is only overwritten in the case of a successful mutation. When this occurs the individual is overwritten with one complete (1+1)-CMA iteration of the parent. The parent on the other hand has only its global step-size updated and its mean success probability. In the case when the replication produced an infeasible solution, then neither the global step-size nor the success probability is updated. In this case the mutation strength is reduced for the individual i as discussed above. This procedure of *variation* followed by *selection* is repeated a number of times (*generations*) until some termination criteria is met, commonly a maximum number of function evaluations. Here number of generations can be considered a finite number of replications or function evaluations. The pseudcode for the search strategy for any given thread is illustrated in figure 1. The number of times this procedure is started would typically be equivalent to the number of cores available on the hardware.

Single thread process:

```
1   while termination criteria not satisfied do
2     i, j ← any available individuals, f(x_i) ≤ f(x_j)
3     set individuals i, j as busy
4     y ← perform one (1+1)-CMA-ES replication
      step for individual i, eqn. (4)–(6)
5     if y infeasible
6       reduce mutation strength for individual i
7     else
5       σ_i ← update the global step size for
        individual i, eqn. (7)–(8)
6       if f(y) ≤ f(x_i)
7         replace individual j with individual i and
          complete the (1+1)CMA-ES iteration on
          individual j (the four steps)
          (note that x_j ← y and x_i is unchanged).
8       fi
8     fi
9     release individuals i and j.
    od
```

Figure 1: A single thread procedure. The worse individual j is only replaced when a better solution is found, and a complete (1+1)-CMA iteration performed. The global step size of individual i is updated at all times.

4 COMPUTATIONAL PERFORMANCE AND IMPLEMENTATION

The computational performance, or speed-up, for the asynchronous and synchronous parallel (1+1)-CMA-ES algorithms are now compared for two hardware configurations. Both systems use the Linux operating system. The default GNU/C++ compiler is used and the program is implemented using standard Posix threads. They are:

1. Sun Fire X4600 using eight Quad Core AMD Opteron processors, a total of $n_c = 32$ cores, and
2. Intel *i*7-4930K hexacore with $n_c = 6$ cores and 12 hyper-threads.

According to Amdahl's law and considering an overhead $H(n_t)$ for the n_t threads, one expects a speed-up of

$$1/[(1-P)+P/n_c+H(n_t)]$$

where P is proportion of the algorithm that is parallel. The thread overhead will typically come from thread activity such as synchronization, communication and memory bandwidth limitations. On a good parallel machine the overhead is not linear.

The synchronous parallel $(1+1)$-CMA-ES is exactly the same as for the asynchronous version with one exception; each thread is dedicated to a subset of λ/n_t individuals and will *only* compete with individuals within that subset. This implies that the only sync done is when updating the function evaluation counter, this is done using `__sync_add_and_fetch` and does not create much overhead. Essentially, this algorithm is equivalent to evolving a population size of λ/n_t $(1+1)$-CMA-ES in parallel. In the case when $\lambda = n_t$ this would be equivalent to

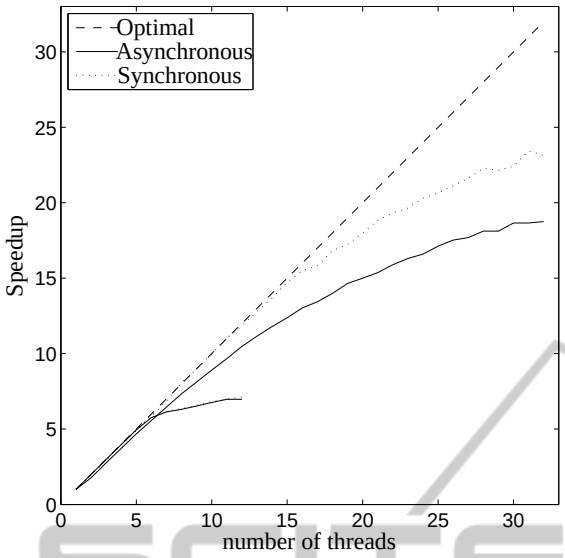

Figure 2: Computation time, normalized by that of using a single core, versus cores (threads).

running $(1+1)$-CMA-ES λ times, i.e. a multi-start. The asynchronous implementation will select competing individuals from the whole population. To check if an individual is available a `__sync_bool_compare_and_swap` is called and this will create some thread overhead. Indeed the loss in speed-up is entirely due to this call and its consequences.

The synchronous and asynchronous $(1+1)$-CMA-ES performance speed up is compared in figure 2 on the different hardware setups. The number of individuals used is $\lambda = 200$ and the total maximum number of functions calls 400.000. The average computational time is then averaged over 250 independent runs. The speed up for Intel's 6 core CPU is linear for both parallel strategies. Only marginal speed-ups are observed using the 12 hyper-threads. For the eight quad core configuration the asynchronous version creates greater overhead than the synchronous version. However, both show a decrease in speed up as the number of threads increases. The reason for this may be numerous hardware and software related issues mentioned above. The speed-up is linear up to 16 core for the synchronous version. Now the global search performance will be studied.

5 GLOBAL SEARCH PERFORMANCE

In this section search performance of the parallel $(1+1)$-CMA-ES is studies for two unconstrained multi-modal test functions and a number of multi-modal constrained test functions. The hardware configurations and implementations are described in the previous section.

5.1 Unconstrained

In order to give some indication of global search performance two unconstrained multi-modal functions are studied. The functions chosen are Ackley's and Kowalik's function. The population is initialized uniformly and randomly in $[-32,32]^{30}$ and $[-5,5]^4$ respectively. The experiments use a global step-size of $\sigma = 64/\sqrt{n}$ and $\sigma = 1/\sqrt{n}$ respectively.

Ackley

In the previous section the asynchronous and synchronous versions of the (1+1)CMA-ES were studied with respect to speed-up. A population size of $\lambda = 200$ was used and a maximum of 400.000 function evaluations. The global hit rate, for these different parallel implementations, is examined on Ackley's function:

$$\begin{aligned} \min \quad f(x) \quad = \quad & -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) \\ & -\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e \end{aligned}$$

This function has many local minima and so can be deceptive for a single $(1+1)$-CMA-ES strategy. Indeed when using 200 restarts of the $(1+1)$-CMA-ES on this function the global hit rate is zero! The global hit rate increases however with the increasing number of competing parallel (1+1)-CMA-ES. In our synchronous version the number competing individuals is λ/n_t and the global hit rate declines to around 98% once $n_t > 20$. This implies that the global search performance is quite acceptable for a population size of around $\lambda = 10$. Indeed the global performance of the (μ/μ_w)-CMA-ES is excellent for this function. The following section studies is a more difficult multi-modal function.

Kowalik

Here Kowalik's function:

$$\min \quad f(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$$

is examined where a =(.1957, .1947, .1735, .1600, .0844, .0627, .0456, .0342, .0323, .0235, .0246) and b^{-1} =(.25, .5, 1, 2, 4, 6, …, 16). The population

size and maximum function evaluations is the same as for Rastrigin's function. The parallel version studied is the asynchronous version using 32 cores/threads. The global hit performance is compared with the $(\mu/\mu_w,\lambda)$-CMA-ES. The code used is CMA-ES[1] using the default settings and a maximum of 5-restarts (Hansen, 2006). The number of global hits for 100 independent runs is given in table 1. On closer inspection it is revealed that local minima are found outside the object variable range initially specified. The global hit performance for the asynchronous parallel (AP-λ) $(1+1)$-CMA-ES is at 84/100 and more successful than the $(\mu/\mu_w,\lambda)$-CMA-ES using restarts.

Table 1: The number of global hits for Kowalik's function out of 100 runs.

Algorithm	global hits
$(4/4_w,8)$-CMA-ES	40
$(\mu/\mu_w,\lambda)$-CMA-ES 5-restart	50
AP-200-(1+1)-CMA-ES	84

5.2 Constrained

In (Arnold and Hansen, 2012) unimodal test functions `g06, g07, g09, g10` from (Runarsson and Yao, 2000) were examined with excellent performance. The multi-modal problem from this set of problems `g01, g08` and from the extended set of benchmarks `g19, g24` (Liang et al., 2006) will be studied here. Furthermore, test function from (Gomez and Levy, 1982) is included, where a constraint cuts the feasible search space into several small islands. Test functions with equivalent constraints will be ignored, as it will be almost impossible to randomly generate an initial feasible starting point for some of these problems. In this experimental study the main concern is with the global hit rate for the optimizer. Furthermore, test function `g02` was not ignored, however, finding solutions close to best known and consistently remains a challenge. The convergence speed of the $(1+1)$-CMA-ES is fast as reported by (Arnold and Hansen, 2012), the danger is that is will be too fast and so trapped in a local minima. In each of the experiments the initial global step size σ is set arbitrarily to 0.1 as in (Arnold and Hansen, 2012).

`g01`

The following test function seems harmless at first, but is indeed non-convex.

Minimize (Floundas and Pardalos, 1987):

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \tag{11}$$

subject to:

$$\begin{aligned} g_1(x) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g_2(x) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g_3(x) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g_4(x) &= -8x_1 + x_{10} \leq 0 \\ g_5(x) &= -8x_2 + x_{11} \leq 0 \\ g_6(x) &= -8x_3 + x_{12} \leq 0 \\ g_7(x) &= -2x_4 - x_5 + x_{10} \leq 0 \\ g_8(x) &= -2x_6 - x_7 + x_{11} \leq 0 \\ g_9(x) &= -2x_8 - x_9 + x_{12} \leq 0 \end{aligned} \tag{12}$$

where the bounds are $0 \leq x_i \leq 1$ $(i = 1,\ldots,9)$, $0 \leq x_i \leq 100$ $(i = 10,11,12)$ and $0 \leq x_{13} \leq 1$. The global minimum is at $x^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ where six constraints are active (g_1, g_2, g_3, g_7, g_8 and g_9) and $f(x^*) = -15$.

In an asynchronous parallel setting using 12 threads and $\lambda = 24$ individuals, all initialized using exactly the same feasible starting solution, the number of global hits is approximately 75 out of 100 independent. However, when using a single $(1+1)$CMA-ES the global hit rate is around $18/100$. If one would now run 12 single $(1+1)$CMA-ES with the same initial solution the hit rate becomes on average $67/100$.

`g08`

Maximize (Koziel and Michalewicz, 1999):

$$f(x) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)} \tag{13}$$

subject to:

$$\begin{aligned} g_1(x) &= x_1^2 - x_2 + 1 \leq 0 \\ g_2(x) &= 1 - x_1 + (x_2-4)^2 \leq 0 \end{aligned} \tag{14}$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is located at $x^* = (1.2279713, 4.2453733)$ where $f(x^*) = 0.095825$. The solution lies within the feasible region. For this reason all variations of the $(1+1)$CMA-ES found the solution with ease each time, in less than 600 function evaluation with a precision of 10^{-16}.

`g19`

(Himmelblau et al., 1972):
Maximize: $f(x) = \sum_{i=1}^{10} b_i x_i -$

[1]MATLAB version 3.61.beta https://www.lri.fr/ ~ hansen/cmaes_inmatlab.html

$\sum_{j=1}^{5}\sum_{i=1}^{5} c_{ij}x_{(10+i)}x_{(10+j)} - 2\sum_{j=1}^{5} d_j x_{(10+j)}^3$

subject to:
$g_j(x) = 2\sum_{i=1}^{5} c_{ij}x_{(10+i)} + 3d_j x_{(10+j)}^2 + e_j - \sum_{i=1}^{10} a_{ij}x_i \geq 0 \quad j = 1,\ldots,5$

where $b = [-40,-2,-.25,-4,-4,-1,-40,-60,5,1]$ and the remaining data is given in Table 2. The bounds are $0 \leq x_i \leq 10$ $(i = 1,\ldots,15)$. Again the problem is trivial for the single $(1+1)$CMA-ES and not only that as it improves on the best known solution reported in (Liang et al., 2006). The solution is $f(x^*) = -29.84240$ $x^* = (0,0,1.659232,0,5.101101,10,0,0,2.7164250,0.502679,0.422431,0.493286,0.398111,0)$. The solution can also be found easily using MATLAB's `fmincon`, which is interesting, since the winner of the CEC2006 competition (Liang et al., 2006) used this function to perform local search. Clearly `fmincon` has been improved since then.

`g24`

Minimize (Floudas and Pardalos, 1990):

$$f(x) = -x_1 - x_2$$

subject to:

$g_1(x) = -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 \leq 0$
$g_2(x) = -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \leq 0$

where the bounds are $0 \leq x_1 \leq 3$ and $0 \leq x_2 \leq 4$. The feasible global minimum is at $x^* = (2.3295, 3.17846)$ where $f(x^*) = -5.50796$.

Experiments with a single $(1+1)$-CMA-ES strategy reveals that it may in some instances be caught in the same local minima $12/100$ times. Running 12 $(1+1)$-CMA-ES strategies on the same initial solutions results in being trapped in $2/100$ times in a local minima. The 12 thread asynchronous $\lambda = 24$ $(1+1)$-CMA-ES finds consistently the global optimum.

Gomez#3

As a final test a function from (Gomez and Levy, 1982) has been included,
Minimize:

$$f(x) = \left(4 - 2.1x_1^2 + \frac{1}{3}x_1^4\right)x_1^2 + x_1x_2 + (4x_2^2 - 4)x_2^2$$

subject to:

$$g_1(x) = -\sin(4\pi x_1) + 2\sin^2(2\pi x_2);$$

where the bounds are $0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$. The optimal solution to this function is $f(x^*) = -0.9711$ at $x_1 = 0.10926$ and $x_2 = -0.623448$. Here the variables are bounded in the range -1 to 1 which contains 20 disjoint feasible regions.

Table 2: Data set for test problem g19.

j	1	2	3	4	5
e_j	−15	−27	−36	−18	−12
c_{1j}	30	−20	−10	32	10
c_{2j}	−20	39	−6	−31	32
c_{3j}	−10	−6	10	−6	−10
c_{4j}	32	−31	−6	39	−20
c_{5j}	−10	32	−10	−20	30
d_j	4	8	10	6	2
a_{1j}	−16	2	0	1	0
a_{2j}	0	−2	0	4	2
a_{3j}	−3.5	0	2	0	0
a_{4j}	0	−2	0	−4	−1
a_{5j}	0	−9	−2	1	−2.8
a_{6j}	2	0	−4	0	0
a_{7j}	−1	−1	−1	−1	−1
a_{8j}	−1	−2	−3	−2	−1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

Again all of the runs start from a single randomly generated feasible start point. In $7/100$ runs the 12 thread asynchronous $\lambda = 24$ $(1+1)$-CMA-ES fails to locate the global minima. This indicates that the mutation strength of $\sigma = 0.1$ is large enough to jump to different feasible regions, although they are not very large. The feasible regions are equally spaces, circular, and with a diameter of around $1/4$. The single $(1+1)$-CMA-ES fails to find the global solution in $55/100$ runs. Running 12 $(1+1)$-CMA-ES strategies on the same initial solutions results in being trapped in $19/100$ times in a local minima.

6 CONCLUSION

The effectiveness of the $(1+1)$-CMA-ES in a competitive asynchronous parallel framework has been demonstrated. Indeed a multi-start $(1+1)$-CMA-ES, even starting at the same point, is an effective global optimizer. Nevertheless, the competitive asynchronous parallel scheme has a slight advantage over using purely restarts.

An asynchronous algorithm is attractive as the load on the cores may be unbalanced. In essence none of the threads should be idle, unless they have trouble finding available individuals for replications and overwriting. However, when considering more costly objective function evaluations the thread overhead will become less important and the asynchronous properties of the algorithm more pertinent.

One of the more challenging test cases studied is

the non-convex quadratic programming problem `g01`. This problem is also challenging for the full $(\mu/\mu_w, \lambda)$-CMA-ES using the various constraint handling techniques discussed in (Hansen, 2011). Interestingly this test case is quite successfully solved using a very simple ES in (Runarsson and Yao, 2000), but this may be attributed to the use of the objective function in biasing the search in the infeasible regions. How to locate feasible regions remains one of the main concerns of the approach presented here. This is especially the case when considering equality constraints. Traversing infeasible regions is also of immediate concern.

Global search performance would clearly be enhanced using multiple different starting points, but it was decided to ignore this option now and limit the search to a single starting point for the constrained problems. The reason for this is that it may be difficult to find a feasible starting points in practice. It is also possible to enhance the global search performance by letting each thread be dedicated to a subset of λ/n_t individuals. Then with some probability (say ten percent) the individual j may be chosen arbitrarily from the entire set of individuals. This will slow down local convergence but enhance global search. This would serve as a mechanism for balancing exploration versus exploitation.

REFERENCES

Arnold, D. V. and Hansen, N. (2012). A (1+1)-CMA-ES for constrained optimisation. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 297–304. ACM.

Auger, A. and Hansen, N. (2005). A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE.

Floudas, C. A. and Pardalos, P. M. (1990). *A collection of test problems for constrained global optimization algorithms*, volume 455. Springer.

Floundas, C. and Pardalos, P. (1987). *A Collection of Test Problems for Constrained Global Optimization*, volume 455 of *Lecture Notes in Computar Science*. Springer-Verlag, Berlin, Germany.

Gomez, S. and Levy, A. (1982). The tunnelling method for solving the constrained global optimization problem with several non-connected feasible regions. In *Numerical analysis*, pages 34–47. Springer.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer.

Hansen, N. (June 28, 2011). The CMA evolution strategy: A tutorial.

Himmelblau, D. M., Clark, B., and Eichberg, M. (1972). *Applied nonlinear programming*, volume 111. McGraw-Hill New York.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44.

Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello-Coello, C. A., and Deb, K. (2006). Technical report, Nanyang Technol. Univ., Singapore.

Runarsson, T. (2003). An asynchronous parallel evolution strategy. *International Journal of Computational Intelligence and Applications*, 3(04):381–394.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294.

Runarsson, T. P. and Yao, X. (2005). Search biases in constrained evolutionary optimization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(2):233–243.

Runarsson, T. R. and Yao, X. (2002). Continuous selection and self-adaptive evolution strategies. In *IEEE Conf. on Evolutionary Computation*, pages 279–284.

Suttorp, T., Hansen, N., and Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 75(2):167–197.