Mixture Model-CMA-ES

Nicolas Vasselin, François Fages

Mixture Model-CMA-ES 2016 internship report

Nicolas Vasselin, François Fages
Inria Saclay-Ile de France, Palaiseau France

December 20, 2016

1 Abstract

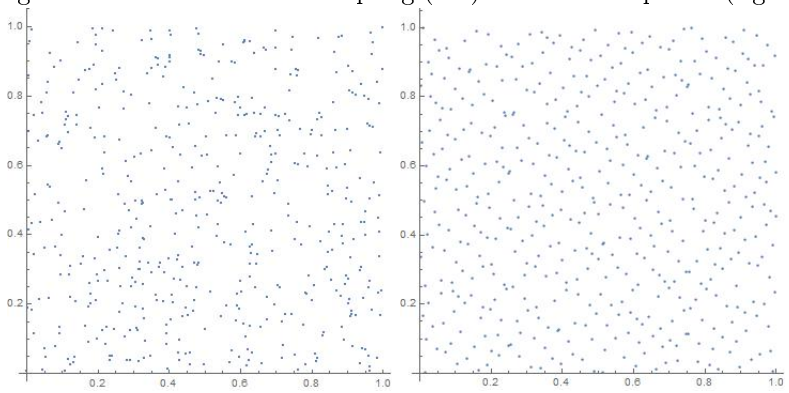
In the following, we report on our attempt to improve the CMA-ES global optimization algorithm based on two ideas: first the use of Sobol's quasi-random low discrepancy numbers instead of pseudo-random numbers, second the design of an alternative to sequential restarts to dynamically adapt the population size, using a mixture model extension of CMA-ES (MM-CMA-ES). On the standard Coco benchmark for evaluating global stochastic optimization methods, the use of Sobol numbers shows a quite uniform improvement, as was already shown by [**teytaud**] last year. On the other hand, MM-CMA-ES does not show speed-up w.r.t. CMA-ES with IPOP restart strategy, even on objective functions with many local minima such as the Rastrigin function. The reason is the overhead in the number of evaluation of the objective functions, introduced by the MM strategy, and the very subtle effect of the adaptive step size strategy of CMA-ES to escape from the covering of several local minima by one (large) normal distribution.
A first approach to the CMA-ES would be Nikolaus Hansen's The CMA Evolution Strategy: A Tutorial.

2 Sobol-CMA-ES

Sobol's low-discrepancy quasi-random sequence intuitively "fills better" the space.

In the following results (appendices B and C), the implementation of [**2**] was used, with a broader initialization allowing for sampling in larger dimensions (see appendix A). Its computational cost proved not to be significant.
We see in the results that sobol-CMA-ES slightly outperforms CMA-ES on every benchmarking function, in every dimension, both when used without restarts, or with the IPOP restart strategy as described in [**3**] (as the first restarting regime).
Similar results were observed in [**1**].

Figure 1: Uniform random sampling (left) vs Sobol's sequence (right)



3 MM-CMA-ES

CMA-ES does not adapt the size of its population in a run. This makes it perform poorly in most testbed functions, especially in higher dimensions and on functions with many local minima. The classical way to adaptively change the population size is by doing sequential, budget-restricted restarts of the CMA-ES and doubling its population size each time. MM-CMA-ES is an attempt at finding a better strategy to adapt the size of the population, by using a Mixture Model to explore the search space in a parallel way. The goal is to avoid restart-related information loss and to outperform IPOP-CMA-ES. Incidentally, it allows for greater parallelism and finding more local optima.

3.1 EM-CMA-ES

This strategy uses the Expectation-Maximization algorithm to cluster the selected points into a new number of villages. Here are its zoomed-out steps:

1. **Sampling** according to each village's law and population
2. **Ranking** all the population, keep only the better half
3. **Clustering** of what's left using EM
4. **Villages composition**: weighted mean of previous villages distributions
5. **Villages updates**: cf CMA-ES

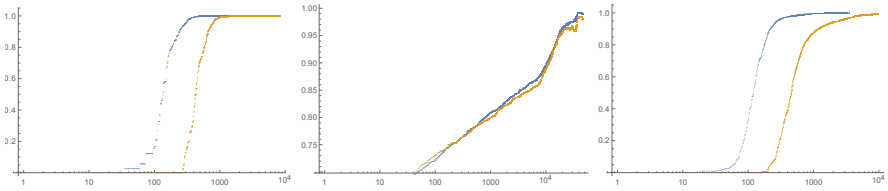
The Mathematica code is provided in D.

The results are not as good as the basic CMA-ES strategy. Figure 2 shows the best evaluation found in y-axis, with respect to the number of target function evaluation in x-axis in log scale, evaluated over a sphere, Rastrigin's, and

Rosenbrock's functions from left to right. EM-CMA-ES is significantly slower over the sphere and Rosenbrock's function, but it is almost as good as CMA-ES on Rastrigin's function.

This can be explained as follows: in EM-CMA-ES, when a village is reaching

Figure 2: EM-CMA-ES (yellow) vs CMA-ES (blue)



the bottom of a local minimum, few or none of its population will be selected to reach the next step, because the other villages will be in deeper pits of potentials. However, while the population of that village is not selected, it means that more than the half of the population of other villages will be remembered. As CMA-ES performs best when the better half of the population is remembered, the other villages will converge slower, because more of their population will be remembered and pulling them in more directions.

The clustering of EM-CMA-ES was expected to cut short to hesitation between two local optima by splitting the distribution and exploring both optima at the same time. However, CMA-ES does not waste much time when covering two local minima, because of the step-size adaptation mechanism.

The step-size adaptation uses a vector "path" variable p_c, whose update is $p_c^{(g+1)} \leftarrow (1-c_c)p_c + \alpha C^(g)^{-\frac{1}{2}} \frac{m^{(g+1)}-m^{(g)}}{\sigma^{(g)}}$, where c_c and α are some learning coefficients. This path variable p_c holds the memory of the means' movements, unaffected by the step-size nor the covariance matrix. Therefore, its norm tells whether the distribution is going forward or staying in place. If it is going somewhere, i.e the mean is always displaced towards the same direction, then $\|p_c\|$ will grow. Besides, it means the distribution is following a slope toward a region of greater interest, and so it should move faster. If, on the contrary, the means don't move much between each generation, then $\|p_c\|$ will decay; besides, it means that the distribution is hovering over a minimum, and so it should converge.

The actual step-size update is as follows: $\sigma^{(g+1)} \leftarrow \sigma^{(g)} exp(\beta(\frac{\|p_c\|}{E\|\mathcal{N}(0,\mathbf{I})\|} - 1))$. The step-size σ will quickly increase when $\|\sigma\|$ is greater than its expected value, and quickly decrease when it's smaller.

That's why CMA-ES doesn't lose much time when hovering over two minima: as the mean will stay roughly immobile between the two pits of potential, the step-size will decrease, reducing the coverage of both pits to focus in the region between them. Then, one better point in one or the other pit will drag the distribution in it. Thus, CMA-ES chooses very quickly and at random between two similar optima. Furthermore, when hovering above many local optima, CMA-ES is more likely (and even more as its population increases) to choose

the better one, skipping the coslty exploration of all of them.

3.2 poison-CMA-ES

This strategy uses a small, constant per-village population, but a variable number of villages, and so a variable global population. It follows the CMA-ES steps for each villages, with the following differences:

- After sampling, check in every village for the division criterion: $\forall (i,j) \in [\![1,\mu]\!]^2 \frac{d_{i\to j}}{d_d(i:\lambda+j:\lambda)} > d_t$, where $d_{i\to j}$ is the distance between x_i and x_j, d_d is the largest eigenvalue of the covariance matrix times the step-size σ, $i:\lambda$ and $j:\lambda$ are the ranks of x_i and x_j, and d_t is an arbitrary threshold.
- If the division criterion is met, split the village in two. Determine the distribution of the daughters using the two matching points as means (disregarding all the other points) and using a normal CMA-ES update, without the μ-rank update of the covariance matrix.
- After the distribution updates, check the fusion criterion between every villages: $\frac{\|m^i - m^j\|^2}{max(d_d^i, d_d^j)} < f_t$, where m^i is the mean of village i, and d_d^i is the defined as in the division criterion, and f_t is another arbitrary threshold.
- If met, fuse the two villages by taking the arithmetic mean of their means m, covariance matrixes C, covariance paths p_C, step-size path p_σ, the geometric mean of their step-size σ.
- Do the poison update.

Note that while the population of each village is constant and equal, the global population is variable.

The poison update is done as follows : each village, at each step, leaves behind it a drop of poison, which is formed by the quadruplet $(m, C, \sigma, fbestever)$, where $fbestever$ is the best target function evaluation done so far by this village (or its ancestors).
Then, every village v updates its toxicity level with respect to each poison drop p by adding to it the quantity $\frac{1}{(\|m^v - m^p\| * \|C^v - C^p\|)^2 * |\sigma^v - \sigma^p|}$, only if $fbestever^p < fbestever^v$. If it brings the toxicity level over the poison threshold p_t, the village is paused and stored in a sleeping queue, where the villages are stored according to their $fbestever$.
Finally, when there are no villages left, and if there are still some villages in the sleeping queue, the poison threshold p_t is doubled, and every village in the sleeping queue that are now under the new threshold are awakened. This allows to dynamically adapt the poison threshold, preventing an early end of all villages.

The results are observable in E. We can see that on the low dimensions (2,3 and 5), the two algorithm are about as efficient. poison-CMA-ES is significantly better on the function 5 Linear Slope, which tests the ability of the algorithm to quickly move a long distance by putting the optimum on the border of the domain of the interest. This is because splits allow the distributions to travel long distances significantly faster than in the standard CMA strategy.
We also observe overall better performances over the Gallagher functions (21 and 22) in low dimensions, which hints toward a better management of many local optima by poison-CMA-ES, which was the researched goal. However, this is not generalized to others such functions (Rastrigin).
The current set of parameters of poison-CMA-ES is fit only for lower dimensions. We can see its performances crumble on dimensions 10 and 20, except on function 5 Linear Slope.
Note that poison-CMA-ES under another definition of complexity adapted to its parallel nature, poison-CMA-ES may outperform CMA-ES: under the hypothesis of roughly constant evaluation cost of the target function, a village-parallel implementation of poison-CMA-ES would actually be faster than the sequential IPOP-CMA-ES.

4 Conclusion

The use of a low-discrepancy quasi-random sequence such as Sobol's is a free increase in performances over all benchmarking function and in all dimensions.
The EM-CMA-ES currently needs too large a starting population and induces a slowdown of the search of interesting regions. However, the clustering algorithm is still roughly used; maybe a finer, more local one could circumvent such issues. Another potential improvement would be a budget management between the villages, giving priority to the most promising villages while keeping exploring new regions.
The current set of parameters of poison-CMA-ES were empirically found based on low-dimension experiments. There is probably room for significant improvement on that regard, especially on higher dimensions. Another way to gain performances may be a more economical way to adapt the poison threshold.

Appendices

References

[1] Olivier Teytaud. *Quasi-random numbers improve the CMA-ES on the BBOB testbed.* Stephane Bonnevay; Pierrick Legrand; Nicolas Montmarché; Evelyne Lutton: Marc Schoenauer. Artificial Evolution (EA2015), 2015, Lyon, France. Springer Verlag, pp.13. <hal-01194542>

[2] William H. Press; Saul A. Teukolsky; William T. Vetterling; Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing.* Cambridge University Press New York, NY, USA ©1992,.SBN:0-521-43108-5

[3] Nikolaus Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. ACM-GECCO Genetic and Evolutionary Computation Conference, Jul 2009, Mon- treal, Canada. 2009. <inria-00382093>

A Sobol's quasirandom sequence C implementation

This function needs to be called once with a negative value of n in order for it to initialize. Then, it will fill the x vector, starting from index 1 (and not 0 !), with the next vector in sobol's quasi-random sequence.

```
#define MAXBIT 30
#define MAXDIM 100
#define INIT_FILE "sobolInit.txt"

void sobseq(int *n, float x[])
{
    int j,k,l;
    unsigned long i,im,ipp;
    static float fac;
    static unsigned long in,ix[MAXDIM+1],*iu[MAXBIT+1];
    static char initialized=0;
    static unsigned long mdeg[MAXDIM+1]={0};
    static unsigned long ip[MAXDIM+1]={0};
    static unsigned long iv[MAXDIM*MAXBIT+1]={0};

    if(!initialized){
        for(i=0;i<MAXDIM*MAXBIT+1;i++) iv[i]=0;
        FILE *fp = fopen(INIT_FILE,"r");
        if(!fp){
            printf("Error in sobseq initialization : %s not found\n",INIT_FILE);
            return;
        }
        fscanf(fp,"d        s        a        m_i      \n");
        for(k=1;k<=MAXDIM;k++){
            fscanf(fp,"%ld        %ld        %ld        ",&i,&mdeg[k],&ip[k]);
            for(i=0;i<mdeg[k];i++){
                fscanf(fp,"%ld ",&iv[k+i*MAXDIM]);
            }
            fscanf(fp,"\n");
        }
        initialized=1;
```

```
    }

    if(*n < 0){
        for(k=1;k<=MAXDIM;k++) ix[k]=0;
        in=0;
        if (iv[1] != 1) return;
        fac=1.0/(1L << MAXBIT);
        for (j=1,k=0;j<=MAXBIT;j++,k+=MAXDIM) iu[j] = &iv[k];

        for (k=1;k<=MAXDIM;k++) {
            for (j=1;(unsigned)j<=mdeg[k];j++) iu[j][k] <<= (MAXBIT-j);
            for (j=mdeg[k]+1;j<=MAXBIT;j++) {
                ipp=ip[k];
                i=iu[j-mdeg[k]][k];
                i ^= (i >> mdeg[k]);
                for (l=mdeg[k]-1;l>=1;l--) {
                    if (ipp & 1) i ^= iu[j-l][k];
                    ipp >>= 1;
                }
                iu[j][k]=i;
            }
        }
    } else {
        im=in++;
        for (j=1;j<=MAXBIT;j++) {
            if (!(im & 1)) break;
            im >>= 1;
        }
        if (j > MAXBIT) FATAL("MAXBIT too small in sobseq",0,0,0);
        im=(j-1)*MAXDIM;
        for (k=1;k<=intMin(*n,MAXDIM);k++) {
            ix[k] ^= iv[im+k];
            x[k]=ix[k]*fac;
        }
    }
}
```

B sobol-CMA-ES vs CMA-ES

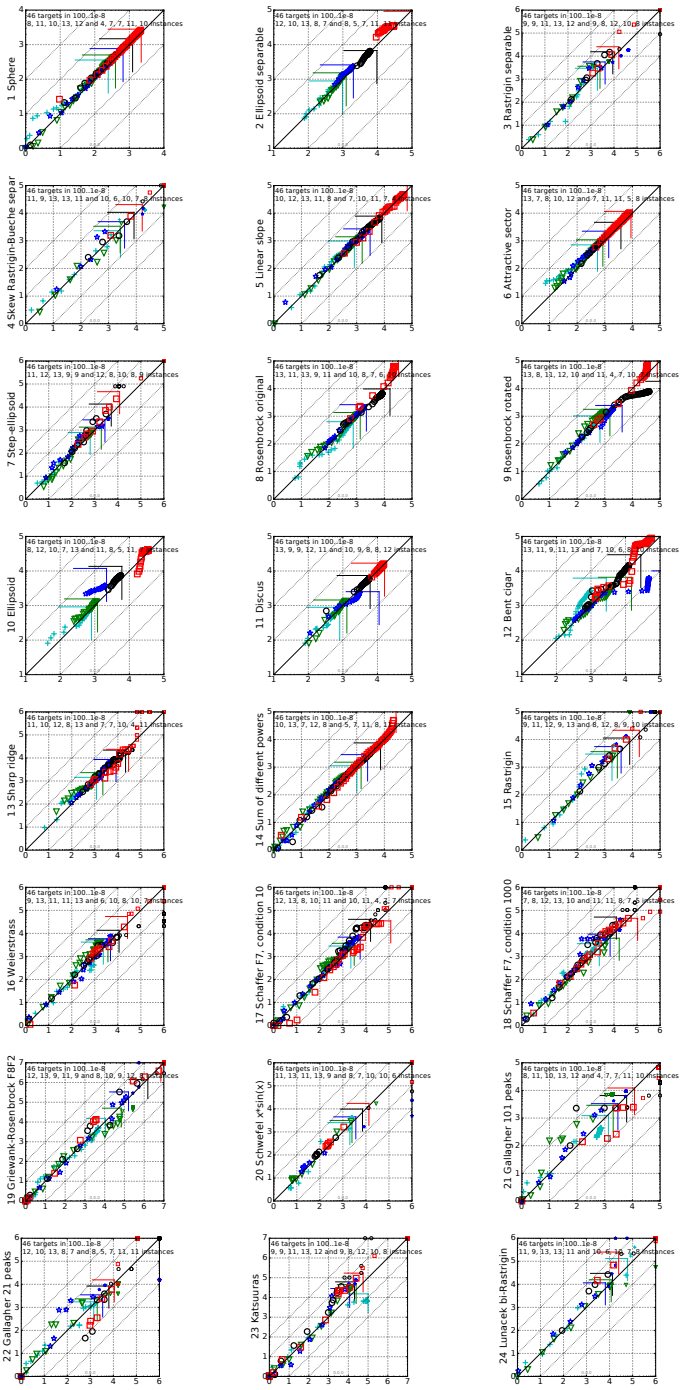


Figure 3: For each graph, corresponding to functions f1 to f24 in the Bbob/Coco framework, the x-axis is the run length for the CMA-ES in log-10 scale, whereas the y-axis is the run length in log-10 scale for the sobol-CMA-ES. When the points are above the diagonal, sobol-CMA-ES outperforms CMA-ES.

C sobol-IPOP-CMA-ES vs IPOP-CMA-ES

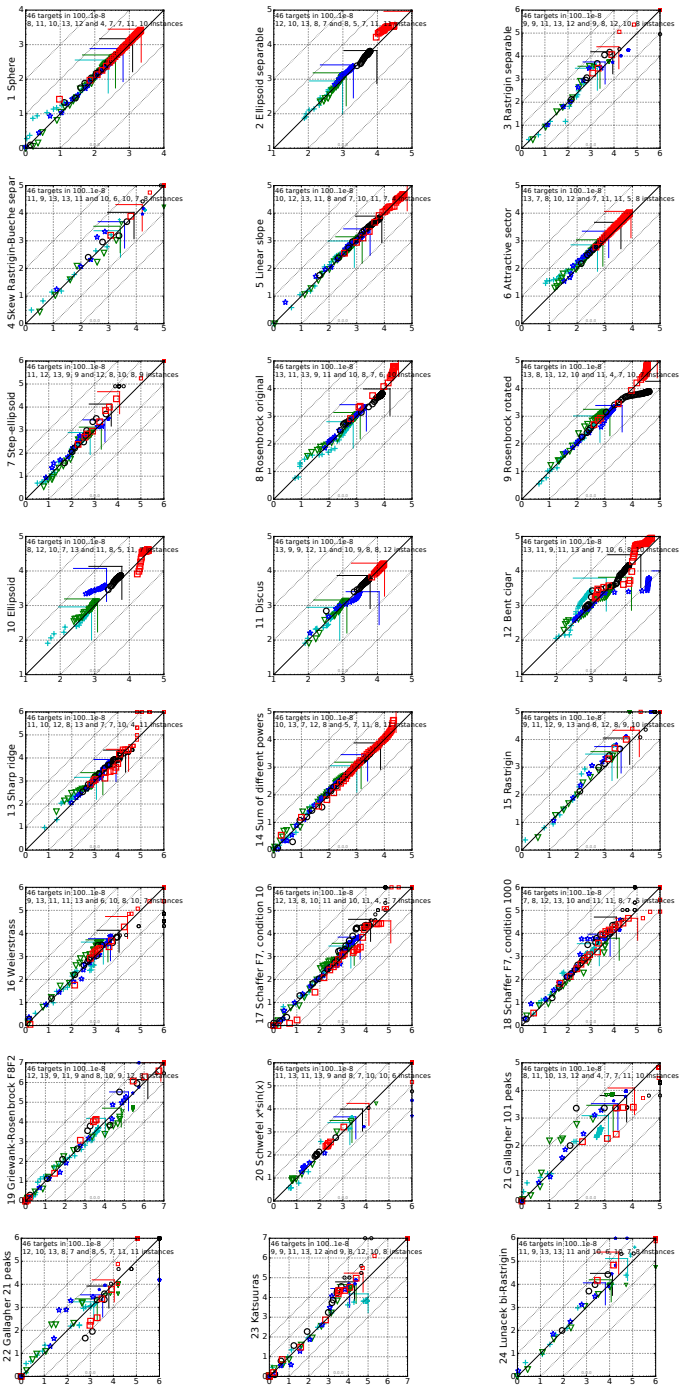


Figure 4: For each graph, corresponding to functions f1 to f24 in the Bbob/Coco framework, the x-axis is the run length for the IPOP-CMA-ES in log-10 scale, whereas the y-axis is the run length in log-10 scale for the sobol-IPOP-CMA-ES. When the points are above the diagonal, sobol-IPOP-CMA-ES outperforms IPOP-CMA-ES.

D EM-CMA-ES

```
gcesMMCMAES[\[Alpha]cov_, \[Lambda]Init_,
    yInit_, \[Sigma]Init_, \[Sigma]Min_, stopTolFun_, maxfeval_][f_] :=
   Module[{villageInit, initParameters, samplePop, updateEigensystem,
    updateDistribution, run, n, \[Chi]n, sqrMatWarnings, norm,
    countevals, almostResults, fbestever, fInit, results,
    terminatedVillages, maxVillages},
   n = Length[yInit];
   \[Chi]n = Sqrt[n] (1 - 1/(4 n) + 1/(21 n^2));
   norm = NormalDistribution[];
   sqrMatWarnings = 0;
   countevals = 0;
   terminatedVillages = 0;
   fbestever = f[yInit];
   fInit = fbestever;
   maxVillages = 1;
   results = {};
   initParameters[\[Lambda]_] :=
    Module[{\[Mu], wPrime, \[Mu]eff, w, c\[Sigma], d\[Sigma], cc, c1,
      c\[Mu]},
     (*\[Mu]=Floor[\[Lambda]/2];*)
     If[\[Lambda] == 1, Print["Warnin : \[Lambda]==1"]];
     \[Mu] = \[Lambda];
     wPrime = N[Table[Log[(\[Lambda] + 1)/2] - Log[i], {i, \[Mu]}]];
     \[Mu]eff = (Sum[wPrime[[i]], {i, \[Mu]}]^2)/
       Sum[wPrime[[i]]^2, {i, \[Mu]}];
     w = Table[0, {i, \[Lambda]}];
     For[i = 1, i <= \[Mu], i++,
      w[[i]] = wPrime[[i]]/Sum[wPrime[[j]], {j, \[Mu]}]];
     c\[Sigma] = (\[Mu]eff + 2)/(n + \[Mu]eff + 5);
     d\[Sigma] =
      1 + 2 Max[0, Sqrt[Max[0, \[Mu]eff - 1]/(n + 1)] - 1] + c\[Sigma];
     cc = (4 + \[Mu]eff/n)/(n + 4 + 2 \[Mu]eff/n);
     c1 = \[Alpha]cov/((n + 1.3)^2 + \[Mu]eff);
     c\[Mu] =
      Min[1 - c1, \[Alpha]cov (\[Mu]eff - 2 +
          1/\[Mu]eff)/((n + 2)^2 + \[Alpha]cov \[Mu]eff/2)];
     {\[Lambda], \[Mu], w, c\[Sigma], d\[Sigma], cc, c1,
      c\[Mu], \[Mu]eff}
     ];(*initParameters*)
   updateEigensystem[{param_, C_, M_, pC_, p\[Sigma]_, \[Sigma]_,
     terminate_}] := Module[{B, D, sqC, invSqC, newTerminate},
     newTerminate = False;
     If[! PositiveDefiniteMatrixQ[C[[1]]],
```

```
    Print["Positive Definite Matrix warning"]; newTerminate = True;,
    {D, B} = N[Eigensystem[C[[1]]]];
    If[Length[Pick[D, Unitize[Im[D]], 1]] > 0,
     Print["Complex eigenvalues : ", D]; newTerminate = True;,
     If[Length[Pick[D, Positive[D]]] < n,
      Print[ "Negative eigenvalue : ", D]; newTerminate = True;,
      B = Transpose[B];
      invSqC = B.DiagonalMatrix[1/Sqrt[D]].Transpose[B];
      D = DiagonalMatrix[Sqrt[D]];
      sqC = B.D;
      If[N[Norm[sqC.sqC - C[[1]]]] > 10^-10, None,
       sqrMatWarnings++];]];
   {C[[1]], B, D, sqC, invSqC, newTerminate}
   ];(*updateEigensystem*)
samplePop[{param_, C_, M_, pC_, p\[Sigma]_, \[Sigma]_,
    terminate_}] := Module[{z, y, x, fx},
   Table[
    z = RandomVariate[norm, n];
    y = C[[4]].z;
    x = M + \[Sigma] y;
    countevals++;
    fbestever = Min[fbestever, f[x]];
    Sow[1 - fbestever/fInit];
    {f[x], z, y,
     x, {param, C, M, pC, p\[Sigma], \[Sigma], terminate}},
    2 param[[1]]]
   ];(*samplePop*)
updateDistribution[cluster_] :=
 Module[{\[Lambda], \[Mu], w, c\[Sigma], d\[Sigma], cc, c1,
    c\[Mu], \[Mu]eff, param, yw, newC, newM, newPc, newP\[Sigma],
    new\[Sigma], newTerminate},
   (*{param_,C_,M_,pC_,p\[Sigma]_,\[Sigma]_,terminate_}*)
   \[Lambda] = Length[cluster];
   param = initParameters[\[Lambda]];
   {\[Lambda], \[Mu], w, c\[Sigma], d\[Sigma], cc, c1,
     c\[Mu], \[Mu]eff} = param;
   newC =
    Sum[w[[i]] cluster[[i, 5, 2,
        1]], {i, \[Mu]}];(*to be updated with updateEigenSystem*)
   newM = Sum[w[[i]] cluster[[i, 5, 3]], {i, \[Mu]}];
   newPc = Sum[w[[i]] cluster[[i, 5, 4]], {i, \[Mu]}];
   newP\[Sigma] = Sum[w[[i]] cluster[[i, 5, 5]], {i, \[Mu]}];
   new\[Sigma] = GeometricMean[cluster[[All, 5, 6]]];
   newC =
    updateEigensystem[{param, {newC}, newM, newPc, newP\[Sigma],
      new\[Sigma], False}];
```

```
  If[newC[[6]], newTerminate = True;,
   (*begin CMAES-like updates*)
   yw = Sum[w[[i]] cluster[[i, 3]], {i, \[Mu]}];
   newM += new\[Sigma] yw;
   newP\[Sigma] = (1 - c\[Sigma]) newP\[Sigma] +
     Sqrt[c\[Sigma] (2 - c\[Sigma]) \[Mu]eff] newC[[5]].yw;
   new\[Sigma] =
    new\[Sigma] Exp[
      c\[Sigma]/d\[Sigma] (Norm[newP\[Sigma]]/\[Chi]n - 1)];
   newPc = (1 - cc) newPc + Sqrt[cc (2 - cc) \[Mu]eff] yw;
   newC = (1 - c1 - c\[Mu] Total[w]) newC[[1]] +
     c1 Transpose[{newPc}].{newPc} +
     c\[Mu] Sum[
       w[[i]] Transpose[{cluster[[i, 3]]}].{cluster[[i,
           3]]}, {i, \[Mu]}];
   newTerminate = (new\[Sigma] < \[Sigma]Min) || (Max[
         cluster[[All, 1]]] - Min[cluster[[All, 1]]] <
       stopTolFun);];
  If[newTerminate, terminatedVillages++];
  {param, {newC}, newM, newPc, newP\[Sigma], new\[Sigma],
   newTerminate}];(*updateDistribution*)
run[villages_] := Module[{pop, clusters, temp1, temp2, newVillages},
  (*parameters,pop,{C,B,D,sqC,invSqC},M,pC,p\[Sigma],\[Sigma],
  terminate*)
  (*generate population*)
  pop = Sort[Flatten[Map[samplePop, villages], 1]];
  pop = Take[pop, Floor[Length[pop]/2]];
  If[Depth[pop[[All, 4]]] != 3,
   Print["Invalid pop depth : ", Depth[pop[[All, 4]]]]];
  temp1 = FindClusters[pop[[All, 4]], Method -> "GaussianMixture"];
  temp1 = Select[temp1, Length[#] > 1 &];
  maxVillages = Max[Length[temp1], maxVillages];
  (*rematch every point with its village info*)
  clusters = {};
  For[i = 1, i <= Length[temp1], i++,
   temp2 = {};
   For[j = 1, j <= Length[temp1[[i]]], j++,
    temp2 =
     Append[temp2, First[Select[pop, #[[4]] == temp1[[i, j]] &]]];
    ];
   clusters = Append[clusters, Sort[temp2]];
   ];
  newVillages = Map[updateDistribution, clusters];
  (*take out the terminated villages*)
  results =
   Catenate[{results, Select[newVillages, Last[#] &][[All, 3]]}];
```

```
  newVillages = Select[newVillages, ! Last[#] &];
  newVillages =
   Map[{#[[1]],
      updateEigensystem[#], #[[3]], #[[4]], #[[5]], #[[6]], #[[
       7]]} &, newVillages];
  newVillages = Select[newVillages, ! #[[2, 6]] &];
  newVillages
  ];(*run*)
villageInit = {{}, {IdentityMatrix[n]}, yInit, Table[0., n],
  Table[0., n], \[Sigma]Init, False};
villageInit[[1]] = initParameters[Floor[\[Lambda]Init/2]];
villageInit[[2]] = updateEigensystem[villageInit];
almostResults =
 NestWhile[
   run, {villageInit}, (# != {}) && (countevals < maxfeval) &][[
  All, 3]];
Print["Unfinished villages : ", Length[almostResults]];
Print["Terminated villages : ", terminatedVillages];
results = Catenate[{results, almostResults}];
{results, countevals}
];
```

E poison-CMA-ES vs sobol-IPOP-CMA-ES

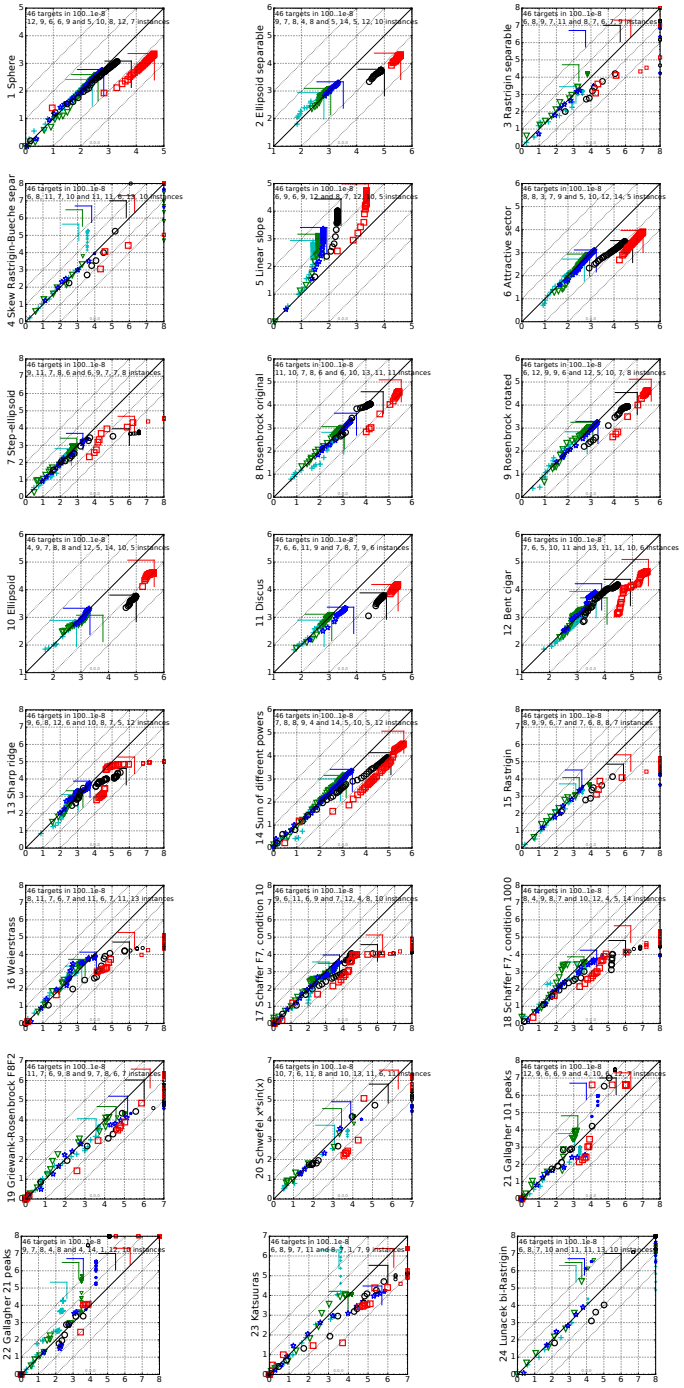


Figure 5: For each graph, corresponding to functions f1 to f24 in the Bbob/Coco framework, the x-axis is the run length for the sobol-IPOP-CMA-ES in log-10 scale, whereas the y-axis is the run length in log-10 scale for the poison-CMA-ES (which uses Sobol's quasi-random sequence too).When the points are above the diagonal, poison-CMA-ES outperforms sobol-IPOP-CMA-ES.