On the Scalability of Particle Swarm Optimisation

Sébastien Piccand Michael O'Neill Jacqueline Walker

***Abstract*— Particle swarm has proven to be competitive to other evolutionary algorithms in the field of optimization, and in many cases enables a faster convergence to the ideal solution. However, like any optimization algorithm it seems to have difficulties handling optimization problems of high dimension. Here we first show that dimensionality is really a problem for the classical particle swarm algorithms. We then show that increasing the swarm size can be necessary to handle problem of high dimensions but is not enough. We also show that the issue of scalability occurs more quickly on some functions.**

I. Introduction

Amongst evolutionary algorithms, Particle Swarm Algorithms [1] have proven their efficiency mainly thanks to their fast convergence. However few experiments have been run on high dimensional problems. By high dimension, we mean problems of size 30 and above. While this is not a large number of dimensions when compared to ones tackled recently by Genetic Algorithms [2] it is a typical problem size tackled in the PSO literature. It seems to be assumed that dimensionality is a problem for particle swarm algorithms, but no studies show it clearly. We are particularly interested in this issue as we are investigating the applicability of Particle Swarm in the domain of Sound Synthesis where we are optimising a model containing over 300 dimensions.

This paper therefore focuses on finding whether problem dimension is a real problem for particle swarm. We try to find out whether increasing the swarm size and/or increasing the optimization process is enough to find a solution.

We focus our studies on two standard algorithms: the LBest and GBest models which are described in section II-A. For each algorithm, the swarm size is increased to observe any change in improvements. The performance measure is the median of the number of iterations required to solve, in a satisfying way, a problem. As far as the problems are concerned, four common benchmark functions are used: Ackley, Griewank, Rastrigin and Rosenbrock.

II. Experimental Settings

This section describes all the parameters used in our experiments.

Sébastien Piccand is with the Department of Computer Science & Information Systems, University of Limerick, Limerick, Ireland and with the Natural Computing Research & Applications group, School of Computer Science & Informatics, University College Dublin, Dublin, Ireland. Michael O'Neill is with the Natural Computing Research & Applications group, School of Computer Science & Informatics, University College Dublin, Dublin, Ireland. Jacqueline Walker is with the Department of Electronic and Computer Engineering, University of Limerick, Limerick, Ireland

A. Particle Swarm Algorithms

LBest and GBest are two variations of the standard PSA (Particle Swarm Algorithm). The standard PSA was developed by Kennedy [1]. The idea is based on the behaviour of flocks or fish swarms looking for food. All the individuals (or Particles in the PSA paradigm) have the same behaviour. The particles represent a solution of a problem and are represented as vectors of real values of fixed length. Each particle, $\vec{x^i}$, is moving with a certain speed $\vec{v^i}$. The particles speed is updated according to the best position the particle has seen so far, $\vec{\overline{x^i}}$, and the best position its neighbours have seen so far, $\vec{\overline{\overline{x^i}}}$. An inertia weight, ω, is also taken into account to prevent the particle from changing its direction or converging too quickly. The particle is also forced not to move to quickly in any direction by constraining its speed in a box. The update rules for particle i are described at equation 1

$$\begin{aligned}
\vec{v^i} &\leftarrow \omega\vec{v^i} + a_c\vec{u_1} \star \left(\vec{\overline{x^i}} - \vec{x^i}\right) + a_g\vec{u_2} \star \left(\vec{\overline{\overline{x^i}}} - \vec{x^i}\right) \\
\vec{v^i} &\leftarrow \max\left(\vec{v^i}, \vec{v_{max}}\right) \\
\vec{v^i} &\leftarrow \min\left(\vec{v^i}, -\vec{v_{max}}\right) \\
\vec{x^i} &\leftarrow \vec{x^i} + \vec{v^i},
\end{aligned} \tag{1}$$

where $\vec{u_1}$ and $\vec{u_2}$ are random real valued vector in $[0,1]^N$, N is the size of $\vec{x^i}$, $\star$ is the term-by-term multiplication, a_c and a_g are the cognitive and global acceleration rates respectively, $\vec{v_{max}}$ is the vector $[v_{max}]^N$, and v_{max} is the speed limit.

1) Topology.: The difference between GBest and LBest is the topology of the swarm. The topology defines the link between the particles, i.e. given a particular particle, which other particles have an influence on it. The topology defines therefore the neighbourhood of the particles. In the GBest version, the neighbourhood is composed of all the particles of the swarm (see Fig. 1(a)), whereas in the LBest version, it is composed only from a few other particles. A typical topology for LBest is the ring topology, where each particle has two neighbours plus itself (see Fig.1(b)). Other topologies have shown some good results, notably the Von Neumann topology [3], where each particle has four neighbours.

(a) Full: GBest (b) Ring: LBest

Fig. 1. The two topology used for a swarm size of 5

2) Constriction factor.: The values for ω, a_c, and a_g have an important role on the algorithms. A classical approach is to used a linearly decreasing inertia ω, from $\omega_{init} = 0.9$ to $\omega_{end} = 0.4$, and to set $a_c = a_g = 2$. An other approach is to use a constriction factor. Clerc [4] proposed to constrain the values of a_c, a_g, and ω in order to ensure the algorithm to converge. A constriction factor, χ is defined according to equation 2.

$$\chi = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|} \tag{2}$$

where $\phi = a_c + a_g > 4$. The velocity update rule, becomes:

$$\vec{v_i} \leftarrow \chi \left[\vec{v^i} + a_c \vec{u_1} \left(\vec{\overline{x^i}} - \vec{x_i} \right) + a_g \vec{u_2} \left(\vec{\overline{\overline{x^i}}} - \vec{x^i} \right) \right]. \tag{3}$$

PSA with constriction factor is therefore equivalent to the standard PSA with $\omega \leftarrow \chi$, $a_c \leftarrow \frac{a_c}{\chi}$, and $a_g \leftarrow \frac{a_g}{\chi}$. Common values are $\phi = 4.1$ and $\chi = 0.7298$, which is equivalent to a standard PSO with $\omega = 0.7298$ and $a_c = a_g = 1.49609$. In the case of the constriction factor, limiting the speed of the particles is not required to ensure convergence, but Eberhart [5] showed that it enables the algorithm to converge more quickly.

After a few experiments, we noticed that the GBest model performed better with a linearly decreasing inertia, whereas the LBest model performed better with the constriction factor. Therefore we only retained those two models. Also in both models, the velocity was limited to the size of the search space, i.e. ,$v_{max} = x_{max}$, where the search space is $[-x_{max}, x_{max}]^N$.

TABLE I

VALUES USED FOR BOTH ALGORITHMS

Algo	ω	a_c	a_g	v_{max}	Topology
GBest	$0.9 \rightarrow 0.4$	2.0	2.0	x_{max}	Ring
LBest	0.7298	1.4609	1.4609	x_{max}	Full connexion

B. Benchmark Functions

To compare the algorithms, the benchmark functions Ackley, Griewank, Rastrigin and Rosenbrock are used. In the following description of the benchmark functions, we define:

- N, the dimension of the problem.
- $\vec{x} = [x_1, x_2, \cdots, x_N]$, the vector representing the position to evaluate.
- x_i, the i^{th} value of vector $\vec{x}$.
- $f_{Function}$, the Function to optimize.

For every function, the minimum, 0, is at $\vec{x} = [0]^N$.

1) Ackley.:

$$f_{Ackley}(\vec{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2} \right) - \exp\left(\frac{1}{N} \sum_{i=1}^{N} \cos(2\pi x_i) \right) + 20 + e$$
$$\vec{x} \in [-32.768, 32.768]^N$$

The solution is considered good enough when $f_{Ackley}(\vec{x}) < 0.01$.

2) Griewank.:

$$f_{Griewank}(\vec{x}) = \frac{1}{4000} \sum_{i=1}^{N} (x_i - 100)^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i - 100}{\sqrt{(i)}} \right) + 1$$
$$\vec{x} \in [-600, 600]^N$$

The solution is considered good enough when $f_{Griewank}(\vec{x}) < 0.05$.

3) Rastrigin.:

$$f_{Rastrigin}(\vec{x}) = \sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right)$$
$$\vec{x} \in [-5.12, 5.12]^N$$

The solution is considered good enough when $f_{Rastrigin}(\vec{x}) < 100$.

4) Rosenbrock.:

$$f_{Rosenbrock}(\vec{x}) = \sum_{i=1}^{N-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$
$$\vec{x} \in [-2.048, 2.048]^N$$

The solution is considered good enough when $f_{Rosenbrock}(\vec{x}) < 100$.

III. Results and Analysis

The experiments have been run for 10,000,000 evaluations of the benchmark function. Each experiment consists of:

- a PSA algorithm: GBest or LBest;
- a benchmark function: Ackley, Griewank, Rastrigin (all three are multimodal) or Rosenbrock (unimodal);
- a problem size: 30, 50, 75, 100, 150, or 200 (and 300, 400, 500 for Ackley and Griewank);
- a swarm size: 25, 50, 75, 100, 150, 200, 250, 300, 400, or 500.

The performance of the algorithm is measured after every 10,000 evaluations of the benchmark function. This means that the precision is +/- 10,000 evaluations. Each experiment has been run 50 times.

We report the median number of evaluations required to have a good result on the benchmark function, according to the problem size. The different lines correspond to the different swarm size used (see Fig. 2, 4, 6, 8, 10, 12, 14, 16). When there are more failure than successful runs, no data is displayed.

In the same way, we also report the successful rate on the benchmark function, according to the problem size (see Fig. 3, 5, 7, 9, 11, 13, 15, 17).

In a third step, to find out if there is a significant difference between the different swarm sizes, a Kruskal-Wallis analysis of variance was made (with a 95% confidence), followed by

a multiple comparison of medians (KW-CM). The Kruskal-Wallis analysis of variance was prefered to an ANOVA because most of the data failed the test of normality. The only variable considered was the swarm size, which means that the Kruskal-Wallis analysis of variance were made for a fixed (PSA algorithm, benchmark function and problem size) triple. However, the analysis can't be directly applied when there are unsuccessful runs. To still be able to compare the effect of the swarm size, we decided to set the results of the unsuccessful runs to twice the maximum number of evaluations allowed (i.e. 20,000,000).

A. Results on Ackley

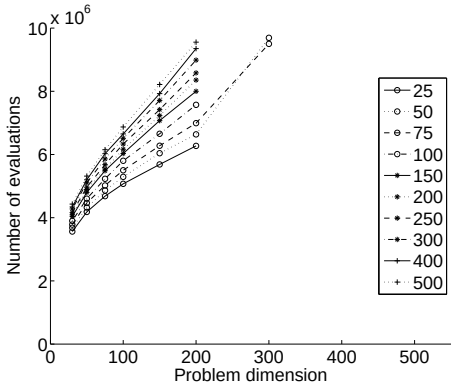

Fig. 2. Median number evaluations to criteria on Ackley with GBest

1) GBest.: Fig. 2 shows two main behaviour: the GBest algorithm manages to solve the problems of size equal or below 200, whatever the swarm size used, but has difficulties handling problems with size of 300 and above.

Let's first focus on the problems of size 200 and below. The curves are quite well separated, and the algorithm seems to perform faster when the swarm size is small. However, the KW-CM, tells us that we can not reject the hypothesis that for a given swarm size (e.g. 75), the algorithms with the swarm size just below or above (i.e. 50 or 100) perform the same. But there is a significant different if the swarm size is increased (or decreased) by two steps (e.g. between swarm size of 50 and 100). It is interesting to notice that for a problem size of 200, GBest starts to have problems to find solutions as seen in Fig. 3. We'll come back on that point later.

Let's now focus on the problems of size 300 and above. Most of the configurations fail more than 50% of the time (see Fig. 3). Only the swarm sizes 50 and 75 are able to find a solution more than 50% of the time within 10,000,000 evaluations.

We can say that there is a problem on Ackley with GBest when the problem size is above 200. However, in one way, one can notice that for a problem size of 200, the number of evaluations required starts to be close to the limit we fixed (10,000,000). Therefore we can not deny that increasing this limit may enable the algorithm to converge. In an other way, if we look at the behaviour of the curves on Fig. 2, there is a change of slope when the problem size reaches 200. More over, for a swarm size of 25, the algorithm can not find a solution for more than 25% of the time. To handle Ackley problem of size 200 and above, it seems therefore necessary to increase both the swarm size (not much) and the maximum number of evaluations (a lot).

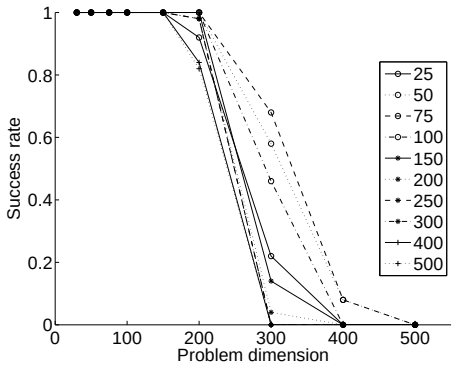

Fig. 3. Success rate on Ackley with GBest

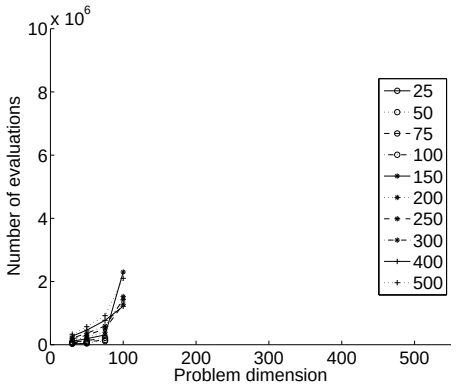

Fig. 4. Median number of evaluations to criteria on Ackley with LBest

2) LBest.: The LBest model is less efficient than the GBest model on the Ackley function, in terms of scaling. It can't solve the problem (at least more than 50% of the time) for problem size over 100 or even 75 in some cases (see Fig. 4). When it can solve the problem (i.e. for low size problems), the LBest model is faster than the GBest.

In terms of swarm size, the KW-CM shows that, for a problem size of 30, the smaller the swarm size, the fastest it finds the solution. But like with the GBest model, we can not reject the hypothesis that for a given swarm size (e.g. 75), the algorithms with the swarm size just below or above (i.e. 50 or 100) perform the same. The same behaviour is also seen from problem size 50 to problem size 100, as soon as the swarm size is bigger than 25. The swarm size of 25 is not enough to find a solution more than 50% of the time. From problem size 150, the swarm size has no real impact on the performance of the LBest model.

On Ackley the LBest model definitely suffers from the increase of the problem size. Fig. 5 shows that its success rate decreases rapidly with the increasing problem size. It

also shows that the smaller the swarm size, the more quickly it fails when the problem size is increased.

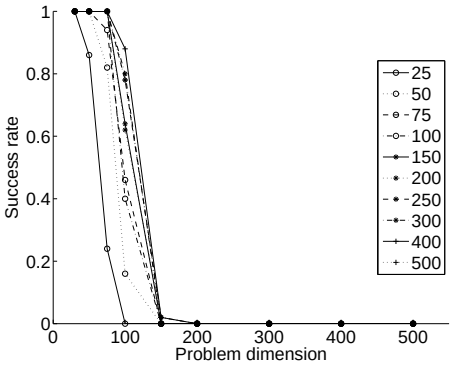

Fig. 5. Success rate on Ackley with LBest

B. Results on Griewank

1) GBest.: According to Fig. 6, the GBest algorithm does not seem to have too many scalability issues. When the problem size is increased, the number of evaluations has to be increased, but quite regularly. It seems that our higher limit of 10,000,000 evaluations is not enough when a large swarm size is used. This is confirmed by Fig. 7. Indeed the success rate is quite stable but for the large swarm sizes for which it decreases on problem of size above 300. The curves on Fig. 6 are well separated, therefore the algorithm seems to perform faster when the swarm size is small. However, the KW-CM tells us that we can not reject the hypothesis that for a given swarm size (e.g. 75), the algorithms with the swarm size one or two steps below or above (i.e. 25, 50, 100 or 150) perform the same. But there is a significant different if the swarm size is increased (or decreased) by three steps (e.g. between swarm size of 50 and 150, there is a significant difference).

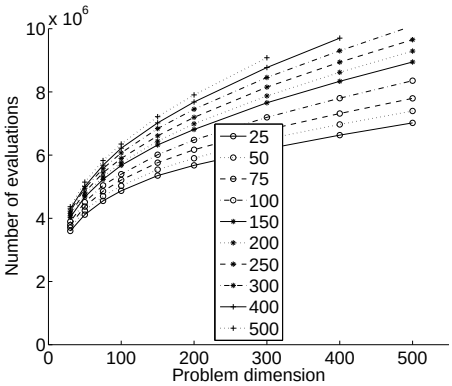

Fig. 6. Median number of evaluations to criteria on Griewank with GBest

Therefore, the GBest model does not have main scalability issues on Griewank until at least a size of 500.

2) LBest.: The performance of LBest on Griewank is quite good. The success rate (see Fig. 9) is all the time at 100% except from a swarm size of 25, for which it decreases

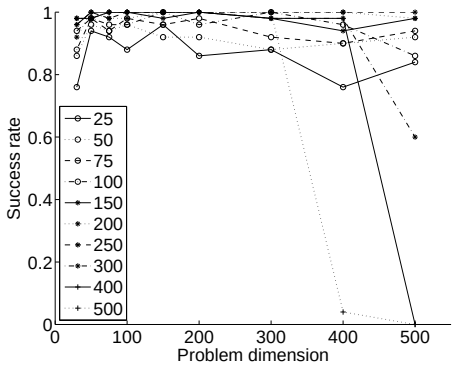

Fig. 7. Success rate on Griewank with GBest

quickly from a problem of size 100. We can also note that with a swarm size of 50 the success rate starts to slightly decrease on the Griewank problem of size 500. This tends to prove that increasing the swarm size may be a good option to solve the problem.

The curves on Fig. 8 seem quite well separated, and the algorithm seems to perform faster when the swarm size is small. However, the KW-CM tells us that we can not reject the hypothesis that for a given swarm size (e.g. 75), the algorithms with the swarm size just below or above (i.e. 50 or 100) perform the same. But there is a significant different if the swarm size is increased (or decreased) by two steps (e.g. between swarm size of 50 and 100).

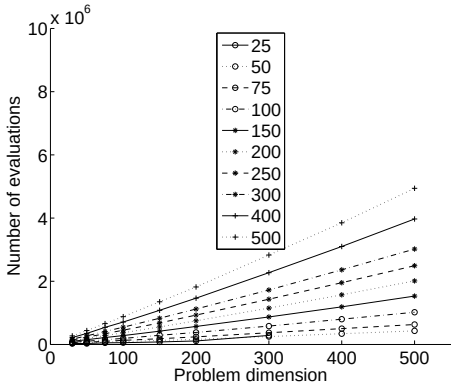

Fig. 8. Median number of evaluations to criteria on Griewank with LBest

On Griewank, the LBest model performs better than the GBest model and does not seem to have many scalability problems as soon as the swarm size is large enough: 50 seems the best compromise for a problem size of 500 or below. We may have to increase the swarm size for problems of larger size though.

C. Results on Rastrigin

1) GBest.: On Rastrigin, the GBest model fails to find a solution more than 50% of the time as soon as the problem size is 100 (except with a swarm size of 300) or above. Fig. 11 shows that for a problem of dimension 100, it is necessary to increase a lot the swarm size to improve the

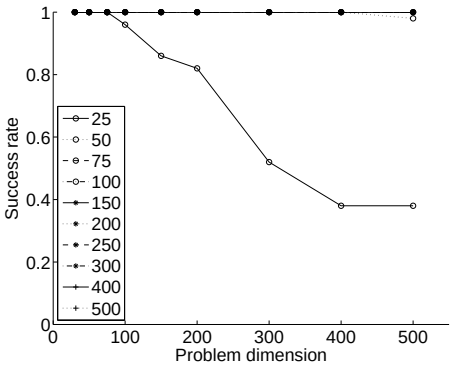

Fig. 9. Success rate on Griewank with LBest

success rate. Swarm sizes of 300, 150 and 400 are the best configurations. From problem size of 150, the GBest model fails all the time.

On problem size below 100, the KW-CM don't show any significant differences, but for problem size 30 where a swarm size of 25 or 50 is significantly better than the others.

The big increase of the sarm size necessary to solve problems of increasing size shows that there is a scalability problem on Rastrigin with the GBest model.

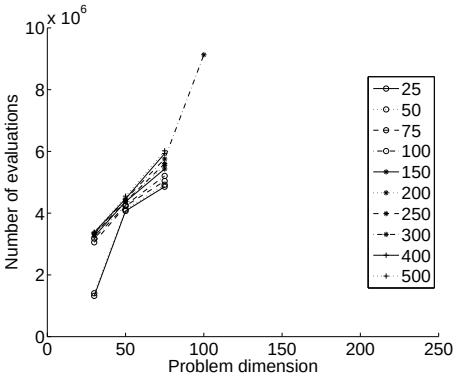

Fig. 10. Median number of evaluations to criteria on Rastrigin with GBest

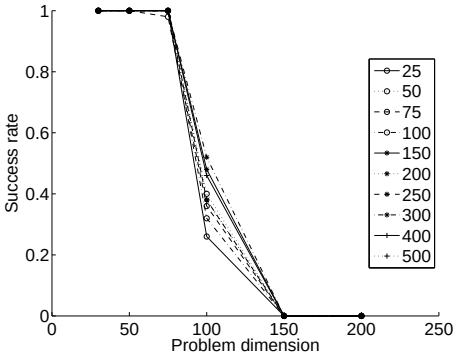

Fig. 11. Success rate on Rastrigin with GBest

2) LBest.: Rastrigin is causing even more trouble to the LBest model. Only the problem of size 30 is solved by all the algorithms. For problem size 50, Fig. 13 shows that the larger the swarm size, the better the success rate. The maximum swarm size of 500 used is however not enough to reach a 100% success rate. For problem size 75 and above, the LBest model fails all the time, whatever the swarm size used.

Like with the GBest model, the Rastrigin function requires a big increase of the swarm size for the LBest algorithm to be able to handle large sizes.

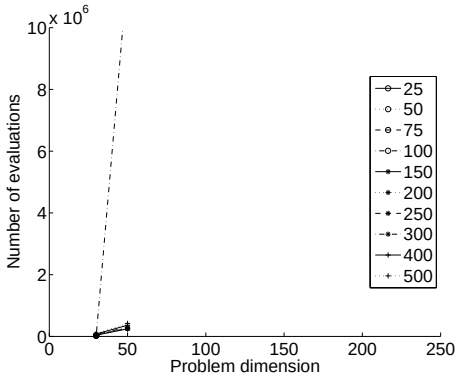

Fig. 12. Median number of evaluations to criteria on Rastrigin with LBest

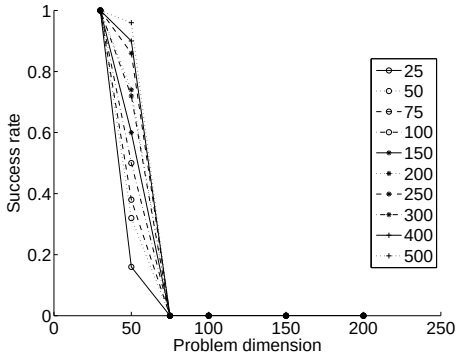

Fig. 13. Success rate on Rastrigin with LBest

D. Results on Rosenbrock

1) GBest.: The GBest algorithm solves the Rosenbrock problem for problem sizes of 25 and 50. For problem size 75, its success rate starts to decrease (Fig. 15) very quickly and from problem size 150, it never solves the problem.

The Fig. 14 tends to show, according to the slope of the curves, that from problem size 100, one may have to use more evaluations that the maximum allowed. The slope of the curve tends to show that an exponentially increasing number of evaluations is necessary to solve the Rosenblock problem when its size increases. But the algorithm may also not be able to find a solution at all.

The KW-CM on problem size below 100 does not show many significant differences between the GBest models of different swarm size: the first significant difference is obtained between GBest models of swarm size 25 and 100. A small swarm size seems therefore to be the best.

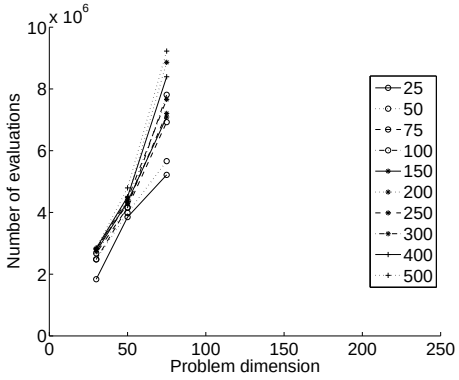

Fig. 14. Median number of evaluations to criteria on Rosenbrock with GBest

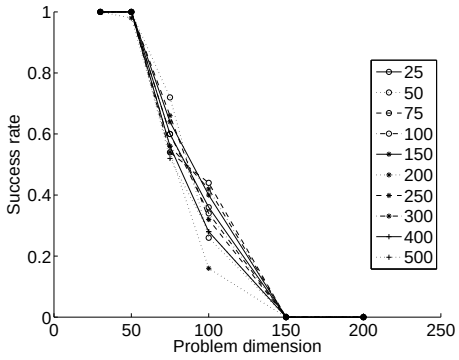

Fig. 15. Success rate on Rosenbrock with GBest

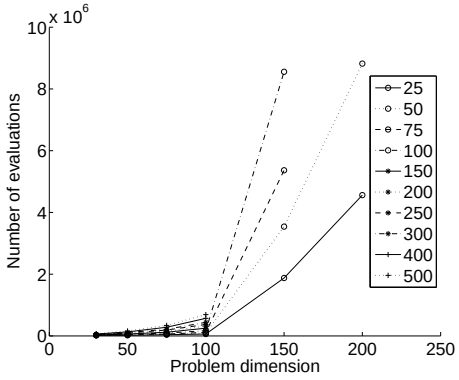

Fig. 16. Median number of evaluations to criteria on Rosenbrock with LBest

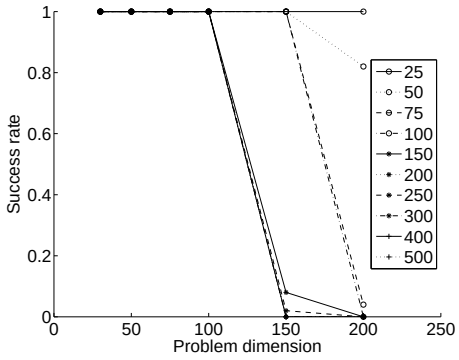

Fig. 17. Success rate on Rosebrock with LBest

2) LBest.: The LBest model performs much better than the GBest on the Rosenbrock problem. Its success rate decreases much later (Fig. 17), from problem size 100. However it is still decreasing very quicly, in particular for large swarm sizes. Surprisingly the smallest swarm size is the only one that enables the LBest to solve the Rosenbrock problem for every problem size.

The Fig. 16 confirms that only the smallest swarm size are efficient on problem sizes 150 and above. The KW-CM gives a significant difference on problem size 200 whereas on problem size 150 and 100, one there is no significant difference between two following swarm sizes. This figure also shows that from problem size 100, the number of evaluations required to solve the problem increasing exponentially, which may be a problem for even bigger problems.

The only issue in terms of scalabilty of the LBest model on Rosenbrock comes from the fact this later observation.

IV. Conclusion

Apart from the Griewank fitness function, we have shown that there is definitely a scalability issue for PSA. To solve problems of increasing size, it is necessary to increase the swarm size and to run for more iterations, but this is not always sufficient to solve the problem in 10,000,000 evaluations. It is, therefore, very difficult to predict the swarm size and the number of evaluations required to solve a problem of known size.

An important problem is when it is necessary to increase the number of evaluations exponentially (e.g. Rosenbrock), which is shown by an abrupt change in the slope of the curves (median number of evaluations necessary to solve a problem when the size of the problem increases) and by the fact that some of these curves stop quickly. Even on the Rastrigin problem it is necessary to increase swarm sizes three fold.

However, one can say that a smaller swarm size has to be prefered as soon as it manages to solve the problem. If the problem size is above 50, a swarm size of 50 is a good choice for any of the functions used.

We are currently replicating this study for a number of the more recent variations on PSA, such as ..... to determine if these more advanced algorithms have better scalability.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948.

[2] D. Goldberg, K. Sastry, and X. Llora, "Towards routine billion variable optimization using genetic algorithms," *Complexity*, vol. 12, no. 3, pp. 27–29, 2007.

[3] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Congress on Evolutionary Computation. CEC'02*, vol. 2, 2002, pp. 1671–1676.

[4] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[5] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particleswarm optimization," in *Congress on Evolutionary Computation. CEC'00*, 2000, pp. 84–88.