2009 Ninth International Conference on Intelligent Systems Design and Applications

Using Smart Sampling to Discover Promising Regions and Increase the Efficiency of Differential Evolution

Vinicius V. Melo and Alexandre C. B. Delbem
Institute of Mathematics and Computer Sciences
University of Sao Paulo
Sao Carlos, SP, Brazil
{vmelo, acbd}@icmc.usp.br

Abstract

This paper presents a novel method to discover promising regions in a continuous search space. Using machine learning techniques, the algorithm named Smart Sampling was tested in hard known benchmark functions, and was able to find promising regions with solutions very close to the global optimum, significantly decreasing the number of evaluations needed by a metaheuristic to finally find this global optimum, when heuristically started inside a promising region. Results show favorable agreement with theories which state the importance of an adequate starting population. The results also present significant improvement in the efficiency of the tested metaheuristic, without adding any parameter, operator or strategy. Being a technique which can be used by any populational metaheuristic, the work presented here has profound implications for future studies of global optimization and may help solve considerably difficult optimization problems.

1. Introduction

The combination of different approaches has been developed to improve the performance of metaheuristics or other global optimization algorithms applied in difficult problems. Some of these approaches have two phases. The first phase (exploration) is used to find promising regions in the search space, where high-quality solutions can be found. The second phase (exploitation) refines the high-quality solutions, from the promising regions found, to find even better solutions. A simple method for the second phase is a local search from the best promising solutions found in the first phase.

To perform the first phase, several approaches have been developed. These approaches use, for example, complex probabilistic models to determine regions of the search space with higher probability of containing optimum solutions. On the other hand, such strategies are, in general, computationally less efficient in in problems with a large number of variables, due to the complexity of the probabilistic models employed.

We propose a novel algorithm for this first phase called Smart Sampling (SS). This technique iteratively explores the search space inside regions with promising solutions. These regions become smaller at each iteration, in a similar way to the strategy proposed in [9]. A machine learning (ML) technique is responsible for determining if new solutions sampled by the SS can be considered promising solutions or not. This process generates high-quality solutions without gradients or hessian matrix. The promising solutions are evaluated and the non-promising are discarded, avoiding computational effort exploiting them. When finished, another ML technique divides the search space into regions involving clusters of promising solutions, which are considered as promising regions.

In this paper, we use the classical Differential Evolution algorithm (DE) [14] as an algorithm for the second phase, running inside each promising region. The DE has been chosen because recent works have been presented showing better performance of this algorithm over other well-known metaheuristics like Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), for instance [4]. The whole approach is called SSDE and its performance is compared to the classical DE in order to evaluate the SS contribution in the optimization process.

It is important to say that the SS is not a global optimization algorithm. It is applied to determine promising regions. Based on this approach, a global optimization problem can be converted into a more localized optimization problem. The SS can be used together with any other population based metaheuristic (GA and PSO, for instance), in contrast to other approaches developed to specific metaheuristics as *niching* and *speciation* [7], and hybrid algorithms,

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.248

which identify these promising regions during the optimization process [5, 8].

This paper is presented as follows. Section 2 describes the SS technique. Section 3, compares the performance of DE and SSDE for hard global optimization benchmark functions. Finally, Section 4 concludes the paper and proposes future works.

2. Smart Sampling

This Section presents a novel approach to explore the search space and determine promising regions. Instead of elaborating complex probabilistic models or let a metaheuristic perform the exploration, an ML technique identifies promising solutions and separate them from non-promising ones.

The proposed SS algorithm is synthesized in Figure 1. In the beginning, the algorithm samples the search space to identify the first large regions for exploration. The higher is the dimensionality of the problem, the larger must be the first sample. The main idea of the SS is to resample only in areas considered promising regions.

When the SS process finishes, the promising regions related to the final solutions must be identified. Some authors propose the use of a clustering algorithm to do this procedure [11, 12, 5]. However, there are some drawbacks in this approach. The clustering result is dependent on the seed, which makes this approach less robust. Moreover, the number of clusters is a parameter that must be specified for the majority of clustering algorithms. However, this parameter is not easily obtained. Fortunately, the unsupervised *k*-windows clustering algorithm [18] does not requires this parameter, and has been successfully used as an operator in a hybrid DE [16]. Nonetheless, the experiments with this hybrid DE were conducted only on 4 benchmark functions and mostly bi-dimensional ones, due to difficulties in clustering higher dimensional points because of orthogonal range search problems [2].

To overcome the drawbacks of the clustering algorithms, another ML technique, called *rule-based learner* (RBL) can be employed. The RBL can adequately represent an undefined number of regions (clusters), being used the proposed approach. The main characteristics of this SS are the following:

1. Given a sub-sample of the problem, called *window*, the algorithm locates smaller windows of the search-space, with a higher concentration of high-quality solutions;

2. To find better solutions, there is no use of gradient or hessian matrix.

3. To avoid over-fitting during the search for promising regions, the algorithm accepts exploration of the search-space, not only exploitation;

4. The algorithm keeps the global optimum inside of one promising region, or close to it;

5. The proposed SS determines promising regions without a pre-specified number of regions (clusters) as a parameter.

In order to achieve these objectives, SS employs a machine learning technique called Classifier, presented in Subsection 2.1.

2.1. Classifiers

Classification is a technique which consists in learning from a previously labeled set of instances (a training set) to generate a model capable of correctly label another set of unknown or unseen instances. A set of known or unknown instances is also called a sample. This technique has been used, for example, in fraud detection, data mining, pattern recognition, and drug discovery, among others [10]. The most common classifiers are based on decision trees, neural networks, or rules [10].

High-quality classifiers, such neural networks and support vector machines, can generate complex models of the data achieving high-precision in the classification task. On the other hand, these techniques may require a high amount of time to be trained.

One of the main objectives of the proposed Smart Sampling is to use simple but efficient techniques in order to provide a good exploration of the search-space. In face of this, we have chosen two of the simplest classifiers: the k-Nearest Neighbor, to separate promising solutions from non-promising ones, and another (more accurate) to identify and separate the final solutions into promising regions (the RBL [10]). Both classifiers are present in the Weka[1] open source machine learning package.

kNN

Instance-based classifiers such as the k-Nearest Neighbor (kNN) algorithm are amongst the simplest of all machine learning algorithms. They classify unknown instances by relating them to the known instances according to some distance/similarity function. Putting it simple, two *close* instances, based on an appropriate distance function, tend to belong to the same class, while two *distant* instances tend to belong to different classes.

An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors. k is a positive integer,

[1] www.cs.waikato.ac.nz/ml/weka/

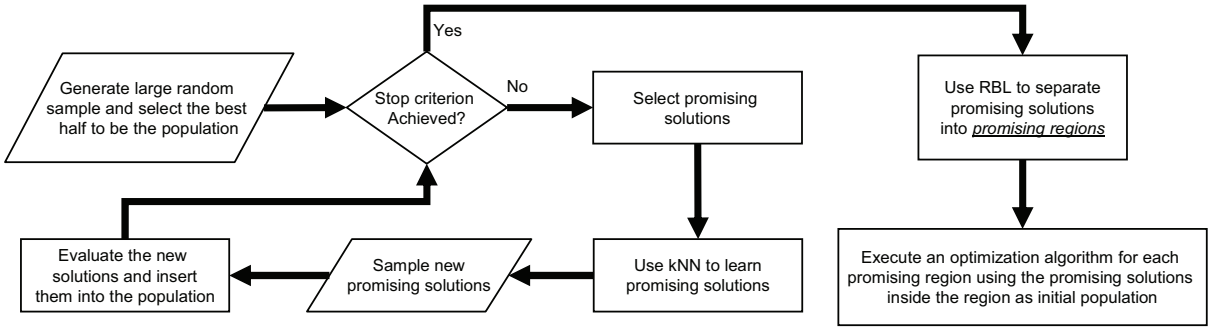

Figure 1. Flowchart synthesizing the proposed SS.

typically small. If k = 1, then the object is simply assigned to the class of its nearest neighbor. The neighbors are from the training set, represented by position vectors in a multi-dimensional feature space. It is usual to use the Euclidean distance and, in an optimization problem where the solutions are points in a response surface, this distance function is the most indicated. The proposed SS uses a instance-based kNN learner called IBk [1].

RBL

An RBL uses a set of IF-THEN rules to determine the classes of training instances based on the information gain criterion, which measures how well a given attribute separates the training instances into targeted classes. The rules can be extracted from the training data using a sequential covering algorithm, where the rules are learned one at a time. Each rule for a given class should *cover* many tuples of that class, and possibly none of other classes. An IF-THEN rule is a logical expression of the form:

- IF *condition* THEN *conclusion*.

The rule's conclusion contains a class prediction. In the presented case, the class is the quality of a solution (*promising* or *non-promising*). By using this kind of structure, it is possible to parse the ruleset and get the set of instances that triggers each rule, thus, splitting them into different regions.

The rule-based learner used in this work was the JRip, the Weka's implementation of the RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [6].

2.2. The proposed SS Algorithm

The basic steps of the proposed SS algorithm, as presented in Figure 1, are explained in more detail as follows.

- Generate initial random population: in this step, a large sample is randomly generated inside the problem's bounds. The best half of this large sample is selected to be the real population of the algorithm;
- Stop if any criterion is met. Three stop criteria have been employed: the number of re-samples (trials), number of trials with no improvement in the best value found (stagnation), and minimum window size;
- Select promising solutions: a fraction of the best solutions is chosen as *promising*. A small fraction can lead to local optima very quickly. A high fraction will probably produce the opposite, since low-quality solutions will be treated as promising ones. We use a conservative fraction of 0.5. The remaining solutions are labeled as *non-promising*.
- Use kNN to learn the difference between solutions labeled as *promising* and *non-promising;*
- Sample new promising solutions: generate new solutions using the re-sampling operator presented in Algorithm 1 and classify them using the kNN. Select the solutions classified as *promising* and discard the others. Repeat this process until a desired number of *promising* solutions is generated. When finished, evaluate them;

Algorithm 1 Resampling Operator.

1. Let $PS_{N,D}$ be a matrix with the promising solutions from the population (where N is the number of promising solutions and D is the number of dimensions of the problem);
2. Generate $NOISE_{N,D}$, a noise matrix generated using a random uniform distribution between -1 and 1;
3. The new solutions will be $PS_{N,D} + (NOISE_{N,D} * PS_{N,D})$;
4. Assert that the new solutions are inside the region given by the problem's bounds. If a coordinate is outside the region, replace its value with the value of the nearest bound.

- Insert the new promising solutions into the population removing the worse ones to keep the population's size constant;

- Finally, use RBL to separate promising solutions into promising regions. For each promising region found, use the promising solutions inside them as initial population for the execution of an optimization algorithm.

In the next Section, experiments employed to evaluate the performance of the SS are presented.

3. Experiments

In this work, we present results on the reduction of the search space and its effects in global optimization problems using the classical Differential Evolution algorithm (DE). The results using DE are compared to the results using SSDE, which corresponds to the approach using DE inside the regions found by the SS.

3.1. Test Benchmark Functions

In this work we use a set of well-known 15 hard benchmark functions (see [17]) in the experimental analysis of the Smart Sampling's performance.

The functions from f_1 to f_{15} are, respectively: *First De Jong*, *Axis parallel hyper-ellipsoid*, *Schwefel's Problem 1.2*, *Rastrigin's function*, *Griewangk's function*, *Sum of different power*, *Ackley's problem*, *Levy function 13*, *Michalewicz function*, *Zakharov function*, *Schwefel's Problem 2.22*, *Step function*, *Alpine function*, *Exponential problem*, and *Salomon problem*.

These functions, being 7 unimodal and 8 multimodal, are tested in two different dimensions (*D* and *2*D*) to increase the problem's difficult, with *D* ranging from 10 to 60, depending on the test function. Therefore, the SSDE is compared to a classical DE on 30 minimization problems, which is randomly started in the problem's bounds.

As 13 of the 15 benchmark functions have a global optimum in the center of the search space, a consecutive series of reduction without loosing the center will never lose the global optimum, making these problems easy for SS. Thus, the problem's domain is shifted to make it asymmetric. The global optimum is moved from the center as follows: if the original parameter bound is $-\alpha \leq x_i \leq \alpha$ and $f_{min} = f(0,...,0) = 0$, then the shifted parameter bound (S.P.B.) is $-\alpha + \frac{\alpha}{2} \leq x_i \leq \alpha + \frac{\alpha}{2}$. However, it is important to notice that the global optimum's position remains the same.

Table 1. Configuration of the SS.

Initial sample size: the best half of D * 100, resulting in D * 50 solutions;
Number of promising solutions: the best 50% of the population;
Maximum iterations: 100;
Maximum iterations without improvement: 10;
Minimum window size: 0.1 for each D;
The code was developed in the R language (*www.r-project.org*).

Table 2. Configuration of the DE.

Population size: N = 100;
Differential amplification factor: F =0.5;
Crossover probability constant: C =0.9**;**
Maximum number of function calls: $MAX_{NFC} = 10^6$**;**
Value to reach: $VTR = o* + 10^{-8}$;
We used the DEoptim package available in R [3].

3.2. Comparison Strategies and Measurements

To compare the results of the algorithms, 50 trials are run with DE and SSDE for each benchmark function and three metrics are used: number of function calls (NFC), success rate (SR), and success performance (SP).

The lower is the NFC, the higher is the convergence speed. The stop criterion of the algorithms is to reach the MAX_{NFC} (the maximum number of function calls allowed) or the VTR (the value-to-reach, the value of the global optimum). The NFC is averaged over the number of successful trials.

The SR value represents the percentage of times the algorithm succeeds to reach the VTR:

$$SR = \frac{number\ of\ times\ reached\ VTR}{total\ number\ of\ trials}.$$

Suganthan et. al. [15] introduced a metric called SP, which combines both NFC and SR:

$$SP = \frac{mean(NFC\ for\ successful\ runs)}{SR}.$$

3.3. Configuration of the Algorithms

Parameter settings of the SS for all conducted experiments are defined as in Table 1. Parameter settings for the DE for all conducted experiments are defined as in Table 2, as presented in [13]. The code was developed in the R language[2]. For the DE algorithm, we used the DEoptim package available in R [3].

The DE is run for each promising region found by the SS, until the VTR is achieved. The VTR is the value of

[2] *www.r-project.org*

the global optimum (o^*) plus a maximum error allowed of 10^{-8}. The SS returns a set of solutions separated in promising regions. The order of the promising regions is determined by sorting the best values found at each promising region. The region which contains the best solution is optimized first by the DE. If the best solution is the same during 300 iterations (stagnation), and it is not the VTR, then the optimization of this region is stopped. This number was determined based on experimental analysis.

The population of the DE is generated as follows: 25% are the best points of a promising region and the other 75% are randomly generated inside the problem's bounds. If the promising region has less than 25 points, we replicate the points adding some noise to them in order to complete 25 solutions.

3.4. Results

Table 3 shows the results obtained applying DE and SSDE to solve 30 test problems (15 test problems with two different dimensions, see Subsection 3.1). D: dimension, NFC: number of function calls (average over 50 trials), SR: success rate, SP: success performance. The last row of the table presents the averages of SR and SP. The best SP for each case is highlighted in boldface. DE is unable to solve the problems f_9 with D = 20, f_{10} with D = 60, and f_{15} with D = 10 and 20. SSDE does not solve the problem f_9 with D = 20. The best SP for each case is highlighted in boldface.

As can be seen in Table 3, SSDE achieved the best SP results on 27 of the 30 functions. On the one hand, the NFC_{SSDE} values are substantially lower on functions $f_3, f_4, f_6, f_{10}, f_{12}, f_{13}$, and f_{14} presenting a reduction of more than 50% in some cases. This results indicate that the DE wastes too much time in non-promising regions, or get trapped in local optima for too many iterations. With SS, this behaviors tends to be minimized, since SS focus on promising regions of the search space, avoiding not only excessive exploration but also competition among high-quality solutions (far from each other, or in local optima).

On the other hand, the NFC of SSDE is considerably higher for functions f_8 and f_9. In these cases, the promising regions determined by the SS did not contain the global optima. Therefore, the SSDE had to search outside the regions to reach the global optimum, which required a considerable larger amount of NFCs.

The SR_{avg} and SP_{avg} measures, shown in the last two rows of Table 3, are respectively the average of the SR and the average of the SP. The SSDE increased the SR_{avg} in practically 9% and reduced the SP_{avg} in approximately 66%, which corresponds to an improvement in the efficacy and a significant reduction in the computational effort to solve the benchmark functions. Thus, it is clear that the proposed SS technique to locate promising regions can improve the performance of the classical DE and, possibly, other populational optimization algorithms.

Table 3. Comparison of DE versus SSDE.

		DE			SSDE		
F	**D**	**NFC**	**SR**	**SP**	**NFC**	**SR**	**SP**
f_1	30	78210	1	78210	41180	1	**41180**
	60	140070	1	140070	83740	1	**83740**
f_2	30	86730	1	86730	51980	1	**51980**
	60	159300	1	159300	100760	1	**100760**
f_3	20	161660	1	161660	86680	1	**86680**
	40	746430	1	746430	363080	1	**363080**
f_4	10	275430	1	275430	16020	1	**16020**
	20	642800	0.1	6428000	37940	1	**37940**
f_5	30	103280	1	103280	66080	1	**66080**
	60	177250	1	177250	119980	1	**119980**
f_6	30	17160	1	17160	7900	1	**7900**
	60	28980	1	28980	18020	1	**18020**
f_7	30	150670	1	150670	114360	1	**114360**
	60	268800	0.7	384000	201000	1	**201000**
f_8	30	87160	1	**87160**	94460	1	94460
	60	155844,4	0.9	**173160.4**	614500	1	614500
f_9	10	183775	0.4	**459437.5**	240400	0.4	601000
	20	-	0	-	-	0	-
f_{10}	30	359640	1	359640	135190	1	**135190**
	60	-	0	-	542070	1	**542070**
f_{11}	30	164890	1	164890	115700	1	**115700**
	60	287910	1	287910	204900	1	**204900**
f_{12}	30	39760	1	39760	10000	1	**10000**
	60	75070	1	75070	23740	1	**23740**
f_{13}	30	383070	1	383070	100800	1	**100800**
	60	403610	1	403610	185420	1	**185420**
f_{14}	10	16820	1	16820	5560	1	**5560**
	20	39940	1	39940	14580	1	**14580**
f_{15}	10	-	0	-	60480	0.2	**112500**
	20	-	0	-	171320	0.2	**264000**
SR_{avg}		0.803			**0.893**		
SP_{avg}		380921.27			**144438.0**		

4. Conclusions

In this work, we presented an approach named Smart Sampling using machine learning techniques to, starting from an initial sample from the search space, find promising regions where the algorithm detects that there is a high chance of finding the global optimum.

As the proposed technique is a pre-optimization process, the objective of this paper is to show how to increase the efficiency of a metaheuristic by discovering promising regions.

Experimental results, conducted on 30 test hard benchmark problems and compared to results of the classical DE, clearly show that our approach is able to outperform the DE based on the number of function calls, success rate, and success performance.

Analyzing the presented results, one can conclude that the proposed approach definitely shows relevant results for the tested benchmark functions. The fact that the SSDE approach presented considerably better results than the classical DE is because the promising regions found by the Smart Sampling contain solutions considerably close to the global optimum, requiring few DE iterations to reach the VTR. This result indicates the significant efficacy of the SS approach.

For future works, we will test the SS with other metaheuristics, as the Genetic Algorithm and Particle Swarm Optimization. Also, one of the key points to the success of the SS algorithm is the resampling operator. The operator presented in this work is a very simple one. Better approaches can lead to even better results.

References

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[2] P. Alevizos and D. Tasoulis. Improving the orthogonal range search k-windows algorithm. In *In Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, pages 239–245, 2002.

[3] D. Ardia. Deoptim package: Differential evolution optimization in r language, 2008.

[4] U. K. Chakraborty. *Advances in Differential Evolution*. Springer Publishing Company, Incorporated, 2008.

[5] R. Chelouah and P. Siarry. Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multi-minima functions. *European Journal of Operational Research.*, 148(2):335–348, 2003.

[6] W. W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[7] K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In *In Schaffer, J., editor, Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, CA, USA, 1989. Morgan Kaufmann.

[8] M. Jelasity, P. Ortigosa, and I. Garcia. Uego, an abstract clustering technique for multimodal global optimization. *Journal of Heuristics.*, 7(3):215–233, 2001.

[9] V. V. Melo, A. C. B. Delbem, D. L. Pinto Junior, and F. M. Federson. Discovering promising regions to help global numerical optimization algorithms. In *MICAI*, pages 72–82, 2007.

[10] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[11] A. C. M. Oliveira and L. A. N. Lorena. Hybrid evolutionary algorithms and clustering search. In *In: Crina Grosan,Ajith Abraham and Hisao Ishibuchi (eds) Hybrid Evolutionary Systems - Studies in Computational Intelligence - Springer SCI Series. 75*, pages 81–102, 2007.

[12] P. M. Ortigosa, I. Garcia, I. García, and M. Jelasity. Reliability and performance of uego, a clustering-based global optimizer. *Journal of Global Optimization*, 19:2001, 2001.

[13] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition versus randomness in soft computing techniques. *Appl. Soft Comput.*, 8(2):906–918, 2008.

[14] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[15] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical Report KanGAL Report 2005005, Nanyang Technological University, Singapore, 2005.

[16] D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis. Clustering in evolutionary algorithms to efficiently compute simultaneously local and global minima. In *Congress on Evolutionary Computation*, pages 1847–1854, 2005.

[17] H. R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence.

[18] M. N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides. The new k-windows algorithm for improving the k-means clustering algorithm. *J. Complex.*, 18(1):375–391, 2002.