Enhancing Incremental Ant Colony Algorithm for Continuous Global Optimization

Udit Kumar, Jayadeva, Sumit Soman
Department of Electrical Engineering, Indian Institute of Technology, Delhi, India
jayadeva@ee.iitd.ac.in

ABSTRACT

We present two enhancements to the local search strategy for Incremental Ant Colony Algorithm ($IACO_{\mathbb{R}}$), that uses Multi-Trajectory Local Search (Mtsls1) as the exploitation engine. First, a new method to handle bound constraints and a modified architecture for Mtsls1 is proposed. The second approach involves a parallel architecture for Mtsls1 along each dimension of the function. We evaluate our approaches on the Soft Computing (SOCO) benchmark functions. The reference approach takes 16% more function evaluations on an average. The proposed parallel approach provides a reduction in the run-time.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Unconstrained Optimization, Global Optimization*

Keywords

Continuous Global Optimization; SOCO; $IACO_{\mathbb{R}}$; Mtsls1.

1. INTRODUCTION

The Incremental ACO ($IACO_{\mathbb{R}}$) framework by Liao et al. [2] uses $IACO_{\mathbb{R}}$ as the exploration engine and Mtsls1 [4] as the exploitation engine. A probability density function constructed around the best solutions is used as the starting point for exploitation. In the case of exploration, all entries in the solution archive are used to generate new solutions.

We present two variations to the original approach. The first involves modifying the method to handle bound constraints, using a new architecture of Mtsls1 to reduce the number of function evaluations. Our second approach gives a parallel flavor to Mtsls1 by optimizing each dimension of the function in parallel, thereby providing an overall reduction in the run-time.

GECCO '15 July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.

DOI: http://dx.doi.org/10.1145/2739482.2764661

2. HANDLING BOUND CONSTRAINTS

In [2], the the weight of bound constraints is a linear function of the number of function evaluations. In our approach, the penalty function weight is updated at the start of each iteration of the local search.

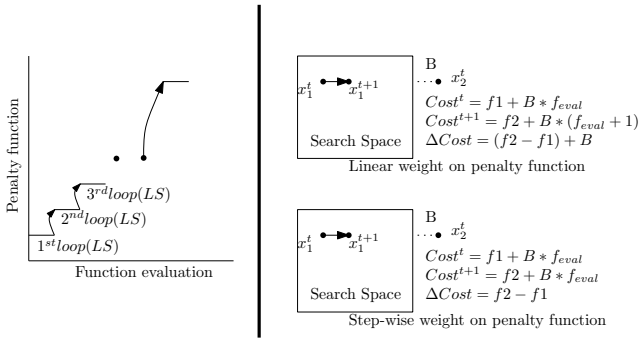

Figure 1: Bound constraint handling and its effect on penalty function weight.

To illustrate the benefits of our approach, consider a 2-D optimization problem with the local search space as shown by the region within the squares in Fig. 1. Let x_1^t and x_2^t be points within and outside the search space, respectively, at iteration t, and let $Cost^t$ represent the total cost. The value of the function at (x_1^t, x_2^t) is $f1$, and the number of function evaluations is f_{eval}. At iteration $t+1$, suppose that only the first dimension x_1^t gets updated to $x_1^{(t+1)}$, and the resultant function value is $f2$.

The change in cost ($\Delta Cost$) would be $(f2 - f1) + B$ and $f2 - f1$ for the original and our approach, respectively. Effectively, the change in cost obtained in our approach is less influenced by the penalty function, which prevents it from dominating over the change in function value.

3. MODIFIED MTSLS1

The Mtsls1 algorithm searches from one dimension to another from an initial point, and moves by a step size of s along one dimension. This is shown in Fig. 2 as the transition from point a to point b. Here, x_k is the k^{th} dimension of a point $x \in \mathbb{R}^n$, while f_n^x is the value of the function f at x. The updated location of x is denoted by $\hat{x}$.

In [2], the penalty function changes weight from one dimension to the next, requiring a function evaluation every time. While moving from a to b, the function is re-evaluated, even though x is the same. In our work, the penalty func-

tion weight remains constant across all dimensions. Hence, the function re-evaluation at a is not required.

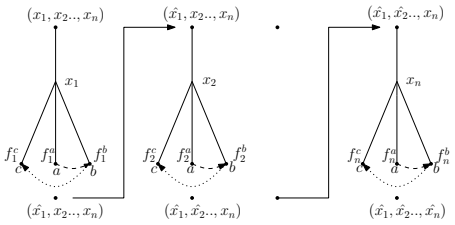

Figure 2: Mtsls1 search procedure.

Specifically, [2] requires 2-3 function evaluations along each dimension, for computing f_k^a, f_k^b and f_k^c at the k^{th} dimension as in Fig. 2. Our approach requires **1-2** function evaluations, f_2^b and f_2^c (when $f_2^b > f_1^a$), or $f_2^a = f_1^{a/b/c}$ from the previous iteration depending on the change in the function value. We term our approach as the $IACO_{\mathbb{R}}$-Modified Mtsls1 ($IACO_{\mathbb{R}}$-MMtsls1).

4. PARALLEL APPROACH FOR MTSLS1

We also propose a parallel approach called the $IACO_{\mathbb{R}}$-Parallel Mtsls1 ($IACO_{\mathbb{R}}$-PMtsls1). In the serial case, we optimize the dimensions of the function without considering their sequence, indicating that these can be optimized individually. Thus, a parallel approach (Fig. 3) would begin the search process along each dimension simultaneously to obtain the optimal point along that dimension. This paper is restricted to the performance analysis of this approach, while speedups may be obtained by implementation on parallel platforms.

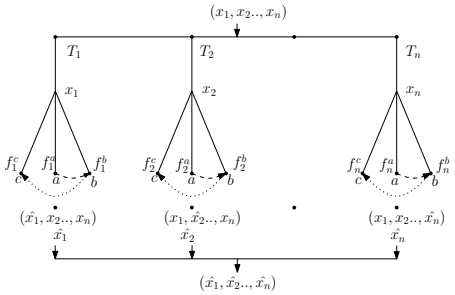

Figure 3: The $IACO_{\mathbb{R}}$-PMtsls1 approach.

5. EXPERIMENTAL STUDY AND RESULTS

We evaluate our approach by comparing its performance on the Soft Computing (SOCO) Benchmarks [1, 3]. Fig. 4 compares $IACO_{\mathbb{R}}$-MMtsls1 and $IACO_{\mathbb{R}}$-PMtsls1 with other approaches. The average and median errors for our approaches are comparable to others.

Table 1 shows the results of applying the Wilcoxon signed-ranks test to estimate the overall difference in performance. $(W+)$ and $(W-)$ denote sum of signed ranks, and (N) indicates the number of instances for which there is a difference in the result between the two algorithms. A p-value less than 0.05 indicates that the result has a significant statistical difference with the algorithm being compared. For $IACO_{\mathbb{R}}$-MMtsls1, there is no statistically significant difference on **10** out of the 16 algorithms, while for

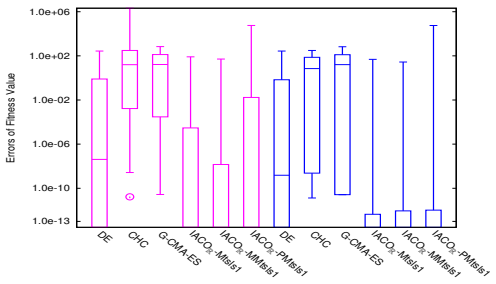

Figure 4: Box plot for average(pink) and median(blue) error.

Table 1: Wilcoxon Signed-Ranks Tests

S. No.	Algorithm	$IACO_{\mathbb{R}}$-MMtsls1				$IACO_{\mathbb{R}}$-PMtsls1			
		W+	W-	N	p-value	W+	W-	N	p-value
1	DE	63	28	13	0.2439	52	39	13	0.6848
2	CHC	190	0	19	**0.0001**	173	17	19	**0.0017**
3	G-CMA-ES	157	14	18	**0.0018**	143	28	18	**0.0123**
4	SOUPDE	47	31	12	0.5693	38	40	12	0.9697
5	DE-D40+Mm	73	32	14	0.2166	66	39	14	0.4263
6	GODE	58	33	13	0.4143	47	44	13	0.946
7	GaDE	34	32	11	0.9658	29	37	11	0.7646
8	jDElscop	17	19	8	0.9453	12	24	8	0.4609
9	MOS-DE	3	25	7	0.0781	0	28	7	**0.0156**
10	MA-SSW-Chains	136	35	18	**0.0279**	126	45	18	0.0778
11	RPSO-vm	136	0	16	**0.0004**	120	16	16	**0.0072**
12	Tuned IPSOLS	82	54	16	0.4691	76	60	16	0.6791
13	EvoPROpt	190	0	19	**0.0001**	154	36	19	**0.0176**
14	EM323	95	10	14	**0.0052**	68	37	14	0.3575
15	VXQR1	75	30	14	0.1726	66	39	14	0.4263
16	$IACO_{\mathbb{R}}$-Mtsls1	25	3	7	0.0781	12	16	7	0.8125

$IACO_{\mathbb{R}}$-PMtsls1, there is no difference on **11**. On an average, the reference approach takes 16% more function evaluations ($IACO_{\mathbb{R}}$-Mtsls1: 135638, $IACO_{\mathbb{R}}$-MMtsls1: 117021 and $IACO_{\mathbb{R}}$-PMtsls1: 117142).

6. CONCLUSION AND FUTURE WORK

We proposed two variants of $IACO_{\mathbb{R}}$-Mtsls1, *viz.* $IACO_{\mathbb{R}}$-MMtsls1 and $IACO_{\mathbb{R}}$-PMtsls1. Our approaches find the optimal solutions, while using a fewer number of function evaluations.

7. REFERENCES

[1] F. Herrera, M. Lozano, and D. Molina. Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. 2010.

[2] T. Liao, M. A. Montes de Oca, D. Aydin, T. Stützle, and M. Dorigo. An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 125–132. ACM, 2011.

[3] M. Lozano, F. Herrera, and D. Molina. Special issue on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. *Soft Comput*, 2011.

[4] L.-Y. Tseng and C. Chen. Multiple trajectory search for large scale global optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3052–3059. IEEE, 2008.