Helper and Equivalent Objective Different Evolution for Constrained Optimisation

Tao Xu
Aberystwyth University
Aberystwyth, UK
tax2@aber.ac.uk

Jun He
Nottingham Trent University
Nottingham, UK
jun.he@ntu.ac.uk

Changjing Shang
Aberystwyth University
Aberystwyth, UK
cns@aber.ac.uk

ABSTRACT

A novel multiobjective evolutionary algorithm is proposed for constrained optimisation in this paper. It transforms a constrained optimisation problem into a two objective optimisation problem. One objective is equivalent to solving the original constrained problem, and the other is the degree of constraint violation which only plays a helper role. This multiobjective problem is decomposed into several single objective optimisation problems using a dynamical weighted sum approach. Each single objective eventually tends to an equivalent objective. A decomposition-based multiobjective optimisation differential evolution algorithm is designed for solving these single objective problems simultaneously.

KEYWORDS

constrained optimisation, multi-objective optimisation, evolutionary algorithms, objective decomposition

ACM Reference format:
Tao Xu, Jun He, and Changjing Shang. 2019. Helper and Equivalent Objective Different Evolution for Constrained Optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019, Prague, Czech Republic, July 13–17, 2019 (GECCO '19),* 2 pages.
https://doi.org/10.1145/3319619.3326752

A constrained optimisation problem (COP) can be formulated in a mathematical form:

$$\begin{array}{ll} \min & f(\vec{x}), \quad \vec{x} = (x_1, \cdots, x_n) \in \Omega, \\ \text{subject to} & \begin{cases} f_i^I(\vec{x}) \le 0, & i = 1, \cdots, q, \\ f_j^E(\vec{x}) = 0, & j = 1, \cdots, r, \end{cases} \end{array} \tag{1}$$

where Ω is a bounded domain in $\mathbb{R}^n$. $f_i^I(\vec{x}) \le 0$ is the ith inequality constraint while $f_j^E(\vec{x}) = 0$ the jth equality constraint. A solution satisfying all constraints is called a feasible solution. Let Ω^* denote the set of optimal feasible solution(s) and Ω_F, Ω_I the sets of feasible and infeasible solutions respectively.

A multi-objective method works by transforming a COP into a multi-objective optimisation problem (MOP) without inequality and equality constraints. The most popular implementation utilises a bi-objective model [4, 11]:

$$\min \vec{f}(\vec{x}) = (f(\vec{x}), v(\vec{x})), \qquad \vec{x} \in \Omega, \tag{2}$$

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07.
https://doi.org/10.1145/3319619.3326752

where $f(\vec{x})$ is the original objective function and $v(\vec{x})$ the sum of constraint violation degrees [9].

Objectives used in the multiobjective optimisation method for COPs can be classified into two types: helper and equivalent objectives. Given a COP and a subset P such that $\Omega^* \cap P \neq \emptyset$, a scalar function $g(\vec{x})$ defined on P is called an *equivalent objective function* if and only if the optimal solution set to $\min g(\vec{x})$ equals to $\Omega^* \cap P$. Otherwise it is called a *helper objective function*. The MOP (2) is composed of two helper objectives [9].

This paper considers a novel multi-objective method which is to transform a COP into a MOP with one equivalent objective and one helper objective [9].

$$\min \vec{f}(\vec{x}) = (e(\vec{x}), v(\vec{x})), \qquad \vec{x} \in \Omega, \tag{3}$$

where $e(\vec{x})$ is an equivalent function and $v(\vec{x})$ the degree of constraint violation, a helper objective function.

A new equivalent function $e(\vec{x})$ is constructed as follows [9]: given a population P, let $x^*(P)$ be the best solution in P:

$$\vec{x}^* = \begin{cases} \arg\min\{v(\vec{x}); \vec{x} \in P\}, & \text{if } P \cap \Omega_F = \emptyset, \\ \arg\min\{f(\vec{x}); \vec{x} \in P \cap \Omega_F\}, & \text{if } P \cap \Omega_F \neq \emptyset, \end{cases} \tag{4}$$

and $\tilde{e}(\vec{x}) := |f(\vec{x}) - f(\vec{x}^*)|$ denote the fitness difference. For any $x \in P$, define

$$e(\vec{x}) = w_1 \tilde{e}(\vec{x}) + w_2 v(\vec{x}), \tag{5}$$

where $w_1 > 0, w_2 > 0$ are weights.

This new equivalent objective aims to reduce the effect of a heavily imposed preference of feasible solutions by the feasible rule [1]. In terms of e, an infeasible solution $\vec{x}$ could be better than a feasible solution $\vec{y}$ if they satisfy the condition

$$w_1 \tilde{e}(\vec{x}) + w_2 v(\vec{x}) < \tilde{e}(\vec{y}).$$

The MOP (3) is decomposed into a group of single objective problems (SOPs) by assigning λ tuples of weights (w_{1i}, w_{2i}, w_{3i}), $i = 1, \cdots, \lambda$.

$$\min f_i(\vec{x}) = w_{1i} \tilde{e}(\vec{x}) + w_{2i} v(\vec{x}) + w_{3i} f(\vec{x}), \tag{6}$$

In the weighted sum, the value of each function is normalised to [0, 1] by min-max normalisation within population P.

A decomposition-based multiobjective optimisation differential evolution, called HECO-DE, is designed for solving the above λ SOPs. HECO-DE adopts mutation and crossover operators similar to those in LSHADE44 [2]. But HECO-DE falls in the framework of multiobjective optimisation while LSHADE44 in single objective optimisation. Two mutation operators and two crossover operators [6, 7] are used in HECO-DE, which are Current-to-pbest/1 mutation [10] and the most popular rand/1 mutation, binomial crossover and exponential crossover [3]. A DE strategy = one mutation + one crossover. The total number of DE strategies in HECO-DE is four and they are labelled by $1, \cdots, 4$. Strategy k is selected with a probability q_k. Strategy k is called success if it generates a better child. Historical memories S_{F_k} and S_{CR_k} preserve the value of mutation factors F_k and crossover rates CR_k for a successful strategy k. F_k and CR_k are adapted in each generation based on their correspondent historical memories. The procedure of HECO-DE is shown in Algorithm 1.

Algorithm 1 The HECO-DE algorithm

Require: objective function $f(\vec{x})$ and constraint violation $v(\vec{x})$;
1: Initialisation: $\lambda \leftarrow 9$, $N_{init} \leftarrow 12n$, $N_{min} \leftarrow \lambda$, circle memories, probabilities $q_k \leftarrow 1/4$ where $k = 1, 2, 3, 4$, archive $A \leftarrow \emptyset$;
2: $t \leftarrow 0$ and $N_t \leftarrow N_{init}$;
3: Randomly generate an initial population P_t of size N_{init};
4: Evaluate $f(\vec{x})$ and $v(\vec{x})$ for $\vec{x} \in P_t$;
5: Number of fitness evaluations $FES \leftarrow NP_{init}$;
6: **while** $FES \le FES_{\max}$ **do**
7: Adjust weights;
8: Sets $S_{F_k} \leftarrow \emptyset, S_{CR_k} \leftarrow \emptyset, k = 1, 2, 3, 4$; $C \leftarrow \emptyset$;
9: Randomly select λ individuals (denoted by Q) from P_t and then update $P_t \leftarrow P_t - Q$;
10: **for** x_i in Q, $i = 1, \ldots, \lambda$ **do**
11: Choose strategy k with probability q_k and generate F_k and CR_k with respective circle memories;
12: Generate a trail vector $\vec{y}_i$;
13: Evaluate $f(\vec{y}_i)$ and $v(\vec{y}_i)$;
14: $Q' \leftarrow Q \cup \{\vec{y}_i\}$;
15: Normalise $\tilde{e}(\vec{x})$, $f(\vec{x})$ and $v(\vec{x})$ in Q';
16: Calculate $f_i(\vec{y}_i)$ and $f_i(\vec{x}_i)$;
17: **if** $f_i(\vec{y}_i) < f_i(\vec{x}_i)$ **then**
18: Store $|f_i(\vec{y}_i) - f_i(\vec{x}_i)|$
19: Save F_k and CR_k into respective S_{F_k} and S_{CR_k};
20: update probability q_k of choosing strategy k;
21: Insert x_i into archive A and insert y_i into set C;
22: **end if**
23: **end for**
24: $P_{t+1} \leftarrow P_t \cup C$;
25: **if** $|A| > N_A$ **then**
26: Randomly delete $|A| - N_A$ individuals from archive A where $N_A = 4.0|P_t|$;
27: **end if**
28: Update circle memories;
29: Recompute population size of P_{t+1};
30: **if** $|P_{t+1}| < |P_t|$ **then**
31: Reduce the population size of P_{t+1} by remove superfluous individuals;
32: **end if**
33: **end while**
34: $t \leftarrow t + 1$;
Ensure: the best individual $\vec{x} \in P_{t+1}$.

Lines 1-5 contribute to initialisation of the initial population size (NP_{init}), minimum population size (NP_{min}), the maximum number of fitness evaluations ($FES_{\max}$), circle memories for DE parameters adaptation, strategy selection probability q_k, external archive A, initial population P_0.

Line 7 is to adjust weights w_{1i}, w_{2i}, w_{3i}. Their initial value is set to $\frac{i}{\lambda}$. Weight $w_{1i,t}$ is adjusted by

$$w_{1i,t} = (\frac{t}{T_{\max}})^L w_{1i,0} \tag{7}$$

where $T_{\max}$ is the maximum count of generations and L is set to 100. Weights $w_{2i,t}$ and $w_{3i,t}$ are adjusted by

$$w_{2i,t} = l(t) w_{2i,0}, \qquad w_{3i,t} = (1 - l(t))\, w_{3i,0}. \tag{8}$$

where $l(t) = (\frac{t}{T_{\max}})$.

In *Line 8*, all sets S_{F_k} and $S_{CR_k}, k = 1, 2, 3, 4$ are set to $\emptyset$. In *Line 9*, λ individuals (of set Q) are randomly chosen from P_t.

In *Lines 10-23*, for each point $\vec{x}_i$ in subpopulation Q, its trail point $\vec{y}_i$ is appended into set Q, then a new subpopulation Q' is generated whose size is $\lambda + 1$. Let $\vec{x}^*(Q')$ be the best solution in Q' which is

$$\vec{x}^* = \begin{cases} \arg\min\{v(\vec{x}); \vec{x} \in Q'\}, & \text{if } Q' \cap \Omega_F = \emptyset, \\ \arg\min\{f(\vec{x}); \vec{x} \in Q' \cap \Omega_F\}, & \text{if } Q' \cap \Omega_F \neq \emptyset. \end{cases} \tag{9}$$

Let $f^*(Q') := f(\vec{x}^*)$.

For an $\vec{x} \in Q'$, calculate the value of $f_i(\vec{x})$ according to formula (6). Thus, the comparison between $\vec{x}_i$ and its trail point $\vec{y}_i$ is based on $f_i(\vec{x}_i)$ and $f_i(\vec{y}_i)$. $\vec{x}_i$ will never be replaced if it is the best in population P.

Afterwards, *Lines 24* is to put back individuals from set C to P_t. In *Lines 25-27*, if the archive size $|A| > N_A$, then randomly delete $|A| - N_A$ individuals from the archive A to ensure the archive size remaining invariant. *Line 28* is to update circle memories [5]. In *Lines 29-32*, the population size of P_{t+1} is linearly decreased by randomly removing superfluous individuals from population P_{t+1} [5].

The theoretical foundation of the helper and objective method for COPs and the detailed explanation of the HECO-DE algorithm can be found in [9]. But the parameters used in this implementation is a little different from [9]. Results and codes are shared on [8]

REFERENCES

[1] K. Deb. 2000. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering* 186, 2-4 (2000), 311–338.

[2] R. Poláková, J. Tvrdík, and P. Bujok. 2016. L-SHADE with competing strategies applied to CEC2015 learning-based test suite. In *Evolutionary Computation (CEC), 2016 IEEE Congress on.* IEEE, 4790–4796.

[3] R. Storn. 1999. System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation* 3, 1 (1999), 22–34.

[4] P. D. Surry and N. J. Radcliffe. 1997. The COMOGA method: constrained optimisation by multi-objective genetic algorithms. *Control and Cybernetics* 26 (1997), 391–412.

[5] R. Tanabe and A. S. Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In *Evolutionary Computation (CEC), 2014 IEEE Congress on.* IEEE, 1658–1665.

[6] J. Tvrdık. 2006. Competitive differential evolution. In *MENDEL*. 7–12.

[7] J. Tvrdík. 2009. Adaptation in differential evolution: A numerical comparison. *Applied Soft Computing* 9, 3 (2009), 1149–1155.

[8] T. Xu. 2019. Results and codes. (2019). https://drive.google.com/open?id=1PS0uvVbQ9EE6Ih78gJotmnlMZHVyMNzz

[9] T. Xu, J. He, and C. Shang. 2019. Helper and Equivalent Objectives: An Efficient Approach to Constrained Optimisation. *arXiv preprint arXiv:1903.04886* (2019).

[10] J. Zhang and A. C. Sanderson. 2009. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation* 13, 5 (2009), 945–958.

[11] Y. Zhou, Y. Li, J. He, and L. Kang. 2003. Multi-objective and MGG Evolutionary Algorithm for Constrained Optimisation. In *Proceedings of 2003 IEEE Congress on Evolutionary Computation.* IEEE Press, Canberra, Australia, 1–5.