DOI: 10.12731/2227-930X-2017-2-26-47
UDC 330.45

COOMA: AN OBJECT-ORIENTED STOCHASTIC OPTIMIZATION ALGORITHM

Tavridovich S.A.

Stochastic optimization methods such as genetic algorithm, particle swarm optimization algorithm, and others are successfully used to solve optimization problems. They are all based on similar ideas and need minimal adaptation when being implemented. But several factors complicate the application of stochastic search methods in practice: multimodality of the objective function, optimization with constraints, finding the best parameter configuration of the algorithm, the increasing of the searching space, etc.

This paper proposes a new Cascade Object Optimization and Modification Algorithm (COOMA) which develops the best ideas of known stochastic optimization methods and can be applied to a wide variety of real-world problems described in the terms of object-oriented models with practically any types of parameters, variables, and associations between objects. The objects of different classes are organized in pools and pools form the hierarchical structure according to the associations between classes. The algorithm is also executed according to the pool structure: the methods of the upper-level pools before changing their objects call the analogous methods of all their subpools. The algorithm starts with initialization step and then passes through a number of iterations during which the objects are modified until the stop criteria are satisfied. The objects are modified using movement, replication and mutation operations. Two-level version of COOMA realizes a built-in self-adaptive mechanism.

The optimization statistics for a number of test problems shows that COOMA is able to solve multi-level problems (with objects of different associated classes), problems with multimodal fitness functions

and systems of constraints. COOMA source code on Java is available on request.

Keywords: *stochastic optimization; object-oriented model; genetic algorithm; particle swarm optimization; differential evolution; multimodal objective function; constraint optimization; self-adaptive parameters.*

СООМА: ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ СТОХАСТИЧЕСКИЙ АЛГОРИТМ ОПТИМИЗАЦИИ

Тавридович С.А.

Стохастические методы оптимизации такие как генетический алгоритм, алгоритм роя частиц и другие успешно применяются для решения оптимизационных задач. Они основаны на схожих идеях и требуют минимальной адаптации при применении. Но существует несколько факторов, затрудняющих использование стохастических методов оптимизации на практике: мультимодальность целевой функции, наличие системы ограничений, необходимость поиска наилучшей конфигурации параметров алгоритма, увеличение объема пространства поиска и др.

В статье предлагается новый Алгоритм каскадной оптимизации и модификации объектов (СООМА), который развивает лучшие идеи известных стохастических методов оптимизации и может применяться к широкому кругу задач, описанных в терминах объектно-ориентированного подхода с использованием практически любых типов параметров, переменных и связей между объектами. Объекты различных классов помещаются в пулы, которые формируют иерархическую структуру в соответствии со структурой связей между классами. Алгоритм также выполняется в соответствии с этой иерархией: методы пулов верхнего уровня перед изменением своих объектов вызывают аналогичные методы вложенных пулов. Алгоритм состоит из шага инициализации и серии итераций, на которых происходит модификация объек-

тов до тех пор, пока не будут выполнены критерии остановки. Объекты модифицируются с использованием операций (инерционного) движения, репликации и мутации. Двухуровневая версия алгоритма реализует механизм автоматического подбора параметров оптимизации.

Результаты проведенных серий тестов демонстрируют возможность использования предлагаемого алгоритма для решения многоуровневых задач (с объектами нескольких взаимосвязанных классов), задач с мультимодальной целевой функцией и системами ограничений. Алгоритм реализован на языке Java, исходный код может быть предоставлен по запросу.

Ключевые слова: *стохастическая оптимизация; объектно-ориентированный подход; генетический алгоритм; метод роя частиц; дифференциальная эволюция; мультимодальная целевая функция; условная оптимизация; автоматический подбор параметров.*

Introduction

Real-world optimization problems in such fields as economy, technology, sociology, and computer science, etc., in general, have nonlinear objective functions depending on a large number of floating-point, integer, boolean parameters and variables combined in linear and nonlinear constraints. Although many special optimization methods have been developed for certain types of problems, it is hard to find an universal method applicable in every case. For example, well-known exhaustive search or gradient methods are not always efficient. Nevertheless there are different implementations of stochastic search methods such as genetic algorithm (GA) [6], particle swarm optimization algorithm (PSO) [3], and others that can be successfully applied to a wide variety of problems with minimal adaptation.

A GA uses mechanisms inspired by biological evolution: reproduction, mutation, recombination and selection. A PSO algorithm imitates the collective behavior of decentralized, self-organized systems. The methods have similar ideas: the initial population of candidate solutions passes through a number of iterations during which the candidate solu-

tions are changed individually (mutation in GA, the inertia and cognitive terms in the velocity-update rule of PSO) and under the influence of other solutions in the population (recombination in GA, the social component in the velocity-update rule of PSO), the best solutions are saved (the new population after reproduction in GA, the particle's best known position and the swarm's best known position in PSO).

Proposed algorithm

A new Cascade Object Optimization and Modification Algorithm (COOMA) is proposed in this paper. It is the universal method that can be applied for the optimization of problems described in the terms of object-oriented models. The basic ideas of COOMA are:

1. The solutions to a given problem and their elements are represented by the objects of different classes extending the base parent class (*Object*). Each class has a number of attributes (extending *Attribute* base parent class) which encode parameters (kept fixed during optimization) and variables (their values are modified). The attributes of simple types (integer, boolean, floating-point, datetime, enumerated, etc.), structured types (e. g. arrays) and object type may be used. The attributes of object type encode associations between objects (many-to-many and any specific combinations): the solutions and their elements, the elements of the solutions and their elements and so on.

2. The objects of different classes are collected in pools (extending *Pool* base parent class) which form the hierarchical structure according to the associations between classes (the topmost pool (main pool) collects the solutions and contains the pools with the elements of the solutions, these pools in their turn contain the pools with the elements of the elements of the solutions and so on). The pool may contain the elements of different classes: e. g. the elements of several child classes extended from one parent class.

3. The pools are logically divided into subsets containing feasible and infeasible elements. The feasible elements satisfy all constraints of the class and meet all requirements of the pool. The feasible and infeasible subsets are conducted in different ways. The infeasible elements are

exposed to strong negative selection (the pressure of changes is heavy) and the feasible elements undergo positive selection (the changes are much more light). The strength of the changes may be different within one subset. For example, the strength of the changes in the feasible subset may be regulated by the value of the fitness function: the elements with better values of the fitness function may be modified with less probability (the analog of the elite population in [12]).

4. The pools may realize different requirements: a) produce the list of objects ordered by the value of the fitness function; b) generate the list of the unique feasible objects (the fitness function is not important in this case); c) combine a) and b). When one optimal solution has to be found, main pool is switched to the first mode. When a list of different solutions with the best value of fitness function has to be found, main pool is switched to the last (combined) mode. Subpools may work in any mode depending on the problem requirements.

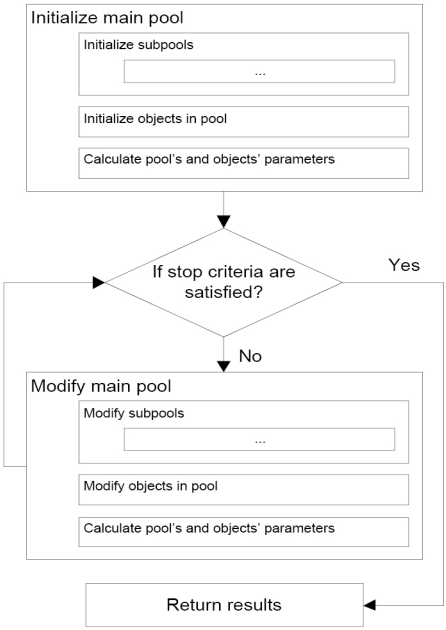

Fig. 1. The flow chart of COOMA

The flow chart of COOMA is shown in Fig. 1. The algorithm is executed according to the pool structure. The *initialize* and *modify* methods of the upper-level pool before changing their objects call the analogous methods of all their subpools in order to get changed objects to use in object attributes. The *initialize* and *modify* methods of the main pool and subpools have the same structure.

The *initialize* method of the pool after initializing its subpools sets the initial values of all attributes of the objects in the pool with the initial variety $v_a \in [0;1]$, $a \in A_b$, $b \in B_c$, $c \in C$ (where C is the set of all pools, B_c is the set of the objects in the pool c and A_b is the set of the attributes of the object b) and calculates pool's and objects' parameters afterwards (see later). $v_a > 0$ means that the attribute is initialized with its default value (nothing is done in fact, this setting is used for attributes-parameters) and $v_a > 0$ means that the attribute is initialized with random value within the limitations (maximal and minimal values, etc.) with probability v_a. The default value for attributes-variables is $v_a = 1$.

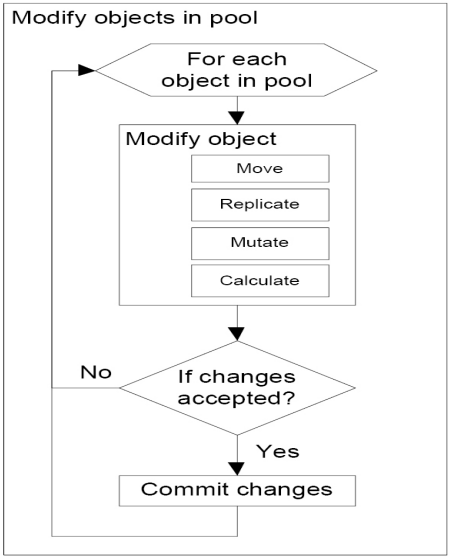

Fig. 2. The flow chart of the "Modify objects in pool" block

The *modify* method of the pool after modifying its subpools modifies all the objects in the pool and calculates pool's and objects' parameters afterwards (see later). The flow chart of the "Modify objects in pool" block is shown in Fig. 2.

The whole idea of the object modification block in COOMA is close to differential evolution algorithm (DE) [16]: if the changed object is better than its previous version, the changes are accepted, otherwise the previous values of attributes remain.

The *modify* method of the object consists of four operations: *movement*, *replication*, *mutation* and final *calculation*.

The *movement* operation repeats the last accepted changes with probability p_c^{mov} for all objects of the pool c (by default $p_c^{mov} = 0.33$). The idea of the operation is close to using the inertia term in the velocity-update rule of PSO. This operation adds last committed changes to floating-point and ordinal attributes. Other types of attributes are not changed.

The *replication* operation: the object being modified b_1, $b_1 \in B_c$ gets the values of the attributes from another object b_2, $b_2 \in B_c$ and uses them for self changes. The probability of operation is $K_c^{rep} \cdot p_{b_1}^{rep}$, where K_c^{rep} – replication coefficient of the pool c and $p_{b_1}^{rep}$ – replication probability of the object b_1 (by default $K_c^{rep} = 1$ and $p_{b_1}^{rep} = 0.25$).

The uniform, one-point, two-point and superposition modes of replication are available. First three modes are the same as in crossover operation in GA (the only difference is that the result of operation is assigned to object b_1, not to a new child object). Superposition is closer to PSO and DE (a random coefficient $k \in [0;1]$ is generated; the values of all corresponding floating-point and ordinal attributes of the objects b_1 and b_2 are combined using formula $x^*_{a_{b1}} = (1-k) \cdot x_{a_{b1}} + k \cdot x_{a_{b2}}$, where $x_{a_{b1}}$ and $x_{a_{b2}}$ are current values of attributes of objects b_1 and b_1, $x^*_{a_{b1}}$ is a new value of attribute of the object b_1; the values of all corresponding object attributes are mixed with probability k as the lists of objects: the objects from attribute a_{b_2} are included into new value with probability k and the objects from attribute a_{b_1} with probability 1 – k.

The object b_2 is selected for replication using roulette-wheel or tournament techniques (like in GA) from the list of objects the values of all floating-point and ordinal attributes of which satisfy the constraint $x_{a_{b2}} \in [x_{a_{b1}} - s^{rep}; x_{a_{b1}} + s^{rep}]$, $s^{rep} = r_{a_{b1}} \cdot K_c^{rsc} \cdot k_{b_1}^{rsc} \cdot k_{a_{b1}}^{rsc}$, where $r_{a_{b1}}$ – the range of values of attribute a_{b_1}, K_c^{rsc} – replication scope coefficient of the pool c, $k_{b_1}^{rsc}$ – replication scope coefficient of the object b_1 and $k_{a_{b1}}^{rsc}$ – replication scope coefficient of the attribute a_{b_1} (by default $K_c^{rsc} = 1$, $k_{b_1}^{rsc} = 1$ and $k_{a_{b1}}^{rsc} = 0.1$); for object attributes the variety of objects in lists is compared with $K_c^{rsc} \cdot k_{b_1}^{rsc} \cdot k_{a_{b1}}^{rsc}$ value (the number of objects with the same genotypes divided by total number of objects in lists should be less than this value).

The *mutation* operation is similar to mutation in GA. It changes the values of attributes with probability $K_c^{mut} \cdot k_b^{mut} \cdot p_a^{mut}$, where K_c^{mut} – mutation coefficient of the pool c, k_b^{mut} – mutation coefficient of the object b, $b \in B_c$ and p_a^{mut} – mutation probability of the attribute a, $a \in A_b$ (by default $K_c^{mut} = 1$, $k_b^{mut} = 1$ and $p_a^{mut} = \frac{1}{|A_b|}$, where $|A_b|$ is a number of attributes of the object b).

The values of floating-point and ordinal attributes are changed using uniform distribution within the mutation scope $[x_{a_b} - s^{mut}; x_{a_b} + s^{mut}]$, $s^{mut} = r_{a_b} \cdot K_c^{msc} \cdot k_b^{msc} \cdot k_{a_b}^{msc}$, where r_{a_b} – the range of values of attribute a_b, K_c^{msc} – mutation scope coefficient of the pool c, k_b^{msc} – mutation scope coefficient of the object b and $k_{a_b}^{msc}$ – mutation scope coefficient of the attribute a_b (by default $K_c^{msc} = 1$, $k_b^{msc} = 1$ and $k_{a_b}^{msc} = 0.1$); the object attributes' lists are changed within the variety regulated with $K_c^{msc} \cdot k_b^{msc} \cdot k_{a_b}^{msc}$ value (the number of changes in the list divided by total number of objects in the list should be less than this value).

The *calculate* method of the object b computes its fitness function's value F(b), penalty function's value $P(b) \geq 0$ and genotype (the string describing unique attributes' values). The penalty function's value is calculated using approach described in [1, Section IV].

The changes of the object are accepted when: a) current penalty function's value is positive (the object is infeasible) and the new penalty function's value is less than its current value; b) the object is feasible and the new fitness function's value is better than its current value;

c) the object has unique value of the new genotype (only for the pools with the unique objects' requirement).

The "Calculate pool's and objects' parameters" block in Fig. 1 computes minimal and maximal values of fitness and penalty functions of objects in the pool and calculates new values of objects' parameters p_b^{rep}, k_b^{rsc}, k_b^{mut} and k_b^{msc} (see above). All objects in the pool are divided into three subsets: a) duplicates (objects with non-unique genotypes) (only for the pools with the unique objects' requirement); b) feasible objects (with zero penalty function's value and unique if required); c) infeasible objects (with positive penalty function's value and unique if required). Each subset is treated individually. All duplicates get the same values of parameters defined by pool settings $p_c^{rep(dup)}$, $k_c^{rsc(dup)}$, $k_c^{mut(dup)}$ and $k_c^{msc(dup)}$ (by default this values are higher than for feasible and infeasible unique objects because duplicates have to be modified to become unique). The objects in feasible and infeasible subsets get values proportionally the values of their fitness and penalty functions. The idea is shown in Fig. 3 (the mutation coefficients are set in range $[k_c^{mut(F^+)}; k_c^{mut(F^-)}]$ for feasible unique objects and in range $[k_c^{mut(P^+)}; k_c^{mut(P^-)}]$ for infeasible unique ones). By default the values for infeasible objects are higher than for feasible. Other parameters of feasible and infeasible unique objects are set similarly in ranges $[p_c^{rep(F^+)}; p_c^{rep(F^-)}]$ and $[p_c^{rep(P^+)}; p_c^{rep(P^-)}]$ (replication probabilities), $[k_c^{rsc(F^+)}; k_c^{rsc(F^-)}]$ and $[k_c^{rsc(P^+)}; k_c^{rsc(P^-)}]$ (replication scope coefficients), $[k_c^{msc(F^+)}; k_c^{msc(F^-)}]$ and $[k_c^{msc(P^+)}; k_c^{msc(P^-)}]$ (mutation scope coefficients).

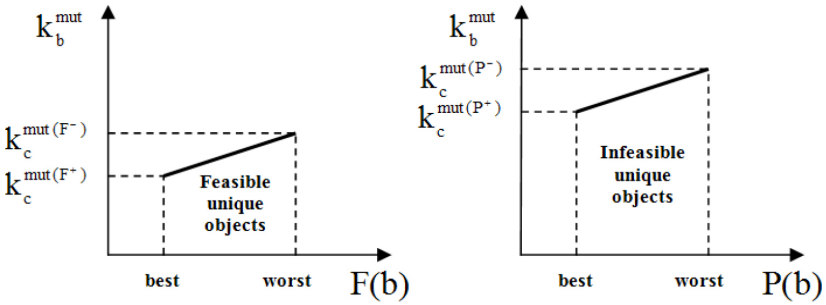

Fig. 3. Setting mutation coefficients of feasible and infeasible unique objects

Standard stop criteria are used: a) stopping after a number of iterations; b) stopping when the specified value of fitness function is reached (can be used mostly for tests); c) stopping after a number of iterations when best fitness and penalty functions' values in the pool don't change; d) stopping by command of operator.

When a subpool generates the list of unique feasible objects, the situation when the total number of possible unique feasible objects is greater than the capacity of the subpool is very likely. In this case the object attributes using this subpool are limited by the list of the objects already generated and many possible objects which provide better solutions can be missing. The *bang* technique is proposed: when the changes in the upper-level pool stop for some time (the fitness and/or penalty functions' values don't change for a period), the *bang* technique is applied to subpools – the attributes of all objects in subpools are mutated with specified probability and scope coefficient and the changes are committed even when the objects get worse fitness and penalty functions' values. A new population with some features of the previous one is produced and after a number of iterations other possible unique objects can be generated.

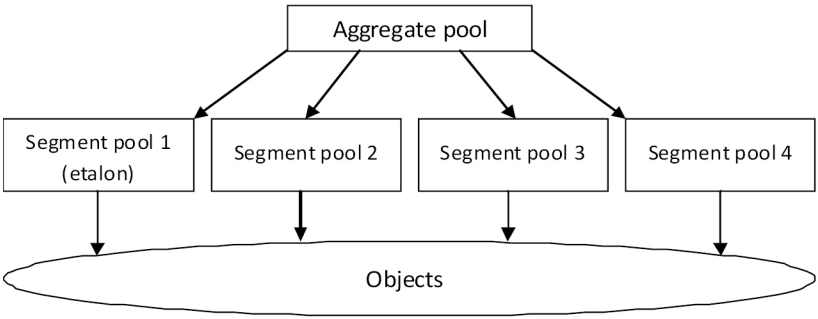

Fig. 4. Two-level approach in COOMA

This version of COOMA is very difficult to use because there are a lot of settings that have to be adopted for different types of problems. And these settings may not work well during the whole optimization process. So there is a further two-level version of COOMA which re-

quires minimal settings because it is self-adaptive. The basic idea of this approach is shown in Fig. 4.

All pools are divided into one aggregate pool c^A and a number of segment pools $c^S \in B_{c^A}$. Segment pools are treated as the objects of the aggregate pool and at the same time act as the pools with their own objects – the solutions and their elements. The fitness function of segment pools is calculated after each N_{c^A} iterations (by default $N_{c^A} = 100$) via the formula: $F(c_i^S) = F^p(c_i^S) + F^f(c_i^S)$, $F^p(c_i^S) = \frac{\frac{n_{c_i^S}^{pc}}{|B_{c_i^S}|}}{\max\limits_{c_j^S \in B_{c^A}} \frac{n_{c_j^S}^{pc}}{|B_{c_j^S}|}} \times \frac{\frac{n_{c_i^S}^{pt}}{|B_{c_i^S}|}}{\max\limits_{c_j^S \in B_{c^A}} \frac{n_{c_j^S}^{pt}}{|B_{c_j^S}|}}$, $F^f(c_i^S) = \frac{\frac{n_{c_i^S}^{fc}}{|B_{c_i^S}|}}{\max\limits_{c_j^S \in B_{c^A}} \frac{n_{c_j^S}^{fc}}{|B_{c_j^S}|}} \times \frac{\frac{n_{c_i^S}^{ft}}{|B_{c_i^S}|}}{\max\limits_{c_j^S \in B_{c^A}} \frac{n_{c_j^S}^{ft}}{|B_{c_j^S}|}}$, where $n_{c^S}^{pc}$ is the count of the accepted penalty changes of the objects in the pool c^S during N_{c^A} iterations, $|B_{c^S}|$ is the total number of the objects in the segment pool c^S, $n_{c^S}^{pt}$ is the total improvement of the best penalty function's value in the segment pool c^S during N_{c^A} iterations, $n_{c^S}^{fc}$ is the count of the accepted fitness changes of the objects in the pool c^S during N_{c^A} iterations, $n_{c^S}^{f}$ is the total improvement of the best fitness function's value in the pool c^S during N_{c^A} iterations. If there are no infeasible objects in the segment pool, $F^p(c_i^S)$ is set to 1.

Initially all segment pools have the equal number of objects. After each N_{c^A} iterations the objects are redistributed among segment pools proportionally the values of their fitness functions, but the number of objects in the segment pool c^S can't be less than its minimal quota q_{c^S} (by default $q_{c^S} = 1$).

The parameters of segment pools (such as p_c^{mov}, K_c^{rep}, $p_c^{rep(dup)}$, $p_c^{rep(F^+)}$, $p_c^{rep(F^-)}$, $p_c^{rep(P^+)}$, $p_c^{rep(P^-)}$, etc.) become attributes which are modified each N_{c^A} iterations after redistributing of objects among segment pools. The first segment pool is called *etalon* pool and its attributes are not changed (it allows to preserve initial parameters).

As the optimization process may pass through a number of stages which require different settings, the *predefined strategies* technique can be realized. There are three basic predefined strategies (the number can be increased): a) standard search, b) local search and c) global search. The default strategy is *standard search*. It has homogenized parameter configuration applicable for different problems. After a number of iterations since the best fitness and penalty functions' values of the objects in all segment pools stopped to change the random strategy is selected. *Local search* strategy helps in situations when COOMA is close to optimal solution and more accurate search in the possible optimum area is required (it has more precise replication and mutation scope coefficients). *Global search* strategy is successful when the local optimum is already reached and we need to explore full search space to find better solutions (it has higher replication and mutation scope coefficients). The new selected strategy is assigned to *etalon* pool.

Results

To demonstrate the operation of COOMA, two multi-level problems were optimized using two-level version of algorithm with four segment pools.

First is one-dimensional wood board cutting problem. Suppose we have three types of wood boards with lengths 200, 300 and 600 cm costing each \$1, \$1.25 and \$1.75 respectively. The boards are to be cut into small board parts: 36 parts with length 120 cm, 25 parts – 300 cm and 14 parts – 500 cm. The total cost of materials should be minimized.

In the terms of COOMA there are three subpools with cutting methods $c_i, i = 1, 2, 3$ (for each possible length of boards) and one main pool with solutions c_4. Objects representing cutting methods $b_{i,j} \in B_{c_i}, j = 1, 2, ..., 40$ have three attributes-variables $x_{b_{i,j},k}$ encoding a number of parts of type k cut from one board of type i and two attributes-parameters $L_{b_{i,j}} = L_i$ and $P_{b_{i,j}} = P_i$ – length and cost of the board of type i. The solutions $b_l \in B_{c_4}, l = 1, 2, ..., 12$ have three object at-

tributes-variables $s_{b_1,i}, i = 1, 2, 3$ containing from 0 to 40 objects from subpools of each type. So solution shows how many boards of each type should be cut using different methods.

The fitness function of the objects representing cutting methods is $F(b_{i,j}) = 120x_{b_{i,j},1} + 300x_{b_{i,j},2} + 500x_{b_{i,j},3} \to \max$, the single constraint is $120x_{b_{i,j},1} + 300x_{b_{i,j},2} + 500x_{b_{i,j},3} \le L_i$ (when the constraint is not satisfied, the penalty function's value equals the difference between the left and the right part of the expression, otherwise the penalty function's value is zero).

The fitness function of the solutions is $F(b_1) = \sum_{b_{i,j} \in S_1} P_{b_{i,j}} \to \min$ and constraints are $\sum_{b_{i,j} \in S_1} x_{b_{i,j},1} \ge 36$, $\sum_{b_{i,j} \in S_1} x_{b_{i,j},2} \ge 25$ and $\sum_{b_{i,j} \in S_1} x_{b_{i,j},3} \ge 14$, where $S_1 = s_{b_1,1} \cup s_{b_1,2} \cup s_{b_1,3}$.

The minimal fitness function's value is \$59.5, the optimal solution is shown in Table 1. The minimal fitness function value is achieved when 34 boards with length 600 cm are cut using the combination of 4 cutting methods.

Table 1.

The optimal solution of the wood board cutting problem (one of)

Board type	Cost per board, \$	Cuts	Part 1, 120 cm	Part 2, 300 cm	Part 3, 500 cm	Cost per cutting method, \$
Board 1, 200 cm	1	–	–	–	–	–
Board 2, 300 cm	1.25	–	–	–	–	–
Board 3, 600 cm	1.75	6	5	–	–	10.5
		3	2	1	–	5.25
		11	–	2	–	19.25
		14	–	–	1	24.5
Total	**–**	**34**	**36**	**25**	**14**	**59.5**

The average optimization statistics of 1000 runs of COOMA for wood board cutting problem is given in Table 2. *Bang* technique practically shows no effect in this case.

Table 2.

COOMA statistics for wood board cutting problem

Test number	Stop when no change, steps	Bang technique applied	Average fitness	Average step number	Solution not found, runs
1	1000	–	60.2	1 867.7	4
2	1000	+	60.14	1 863.6	4
3	2000	–	59.9	3 293.9	6
4	2000	+	59.75	3 321.4	7
5	3000	–	59.72	4 595.8	1
6	3000	+	59.63	4 607.5	6
7	5000	–	59.65	7 034.9	5
8	5000	+	59.54	6 831.9	6

Second test problem is an abstract engineering problem – the machine containing components is designed. The solutions in the main pool c_1 (pool of the machines) have one object attribute s_{b_i}, $b_i \in B_{c_i}$, $i = 1, 2, ..., 12$ which contains 3 objects from the subpool c_2 (components' pool). The objects in the subpool $b_j \in B_{c_2}$, $j = 1, 2, ..., 20$ have three attributes-variables $x_{b_j,k} \in [-3; 3], k = 1, 2, 3$ and are optimized via formula $F(b_j) = 5 \cdot \sum_{k=1}^{3} [1 - \cos(5x_{b_j,k})] \to \min$. The solutions in the main pool are optimized using formula $F(b_i) = \sum_{b_j \in s_{b_i}} \sum_{k=1}^{3} x_{b_j,k}^2 \to \min$.

It is clear that the minimal value of the fitness function is 0 when all three selected components have all variables with zero values. The average optimization statistics of 1000 runs of COOMA for machine design problem is given in Table 3. The capacity of the subpool (20) is less than the possible number of optimal components (125 components with $F(b_j) = 0$). So the *bang* technique helps to find optimal solutions. Without it the average fitness function's value doesn't go lower 2.618 even after 5800 iterations.

Though COOMA is not mainly intended to work with classical mathematical functions and its major mission is to optimize multi-level object structures, to demonstrate the operation of COOMA several

well-known test functions were optimized: 8 functions without constraints and 11 functions with constraints.

Table 3.

COOMA statistics for machine design problem

Test number	Stop when no change, steps	Bang technique applied	Average fitness	Average step number	Solution not found, runs
1	1000	–	2.905	1 380.7	–
2	1000	+	0.066	1 896.0	–
3	2000	–	2.927	2 531.4	–
4	2000	+	0.002	3 131.2	–
5	3000	–	2.923	3 667.8	–
6	3000	+	0.0	4 229.4	–
7	5000	–	2.618	5 800.0	–
8	5000	+	0.0	6 229.0	–

Table 4.

COOMA results for test functions without constraints

Test function	Number of variables	Average step number	Average operations' usage, % (movement : replication : mutation)	Solution not found, runs
De Jong 1	50	6 830.6	1.91 : 12.62 : 85.47	–
Rosenbrock's saddle	50	1 009 831.9	11.18 : 1.29 : 87.53	–
De Jong 3	50	11 868.8	0.63 : 14.76 : 84.61	–
Rastrigin	50	30 002.3	9.58 : 4.7 : 85.72	–
Schwefel	50	77 805.1	21.93 : 1.53 : 76.55	–
Griewank	50	13 451.1	7.72 : 6.92 : 85.36	1
Ackley	50	17 055.3	7.65 : 7.56 : 84.79	–
Schaffer N. 4	2	190.6	11.53 : 7.56 : 80.91	–

The average optimization statistics of 100 runs of COOMA for test functions without constraints is given in Table 4. The parameters of optimization: 12 solutions in the main pool divided into 4 segment pools, fitness function's precision ±0.001, apply new strategy after

100n steps during which fitness function doesn't change (where n is the number of variables), stop when exact or better value of fitness function is achieved (the search also stops and reports "Solution not found" if after 50 changes of strategy, fitness function's value is still not found).

All functions were successfully optimized. The worst results (maximal average step number 1 009 831.9) were shown for Rosenbrock's saddle test.

The functions with constraints marked G1-G11 were taken from [8, Appendix]. The average optimization statistics of 100 runs of COOMA for test functions with constraints is given in Table 5. The parameters of optimization: 12 solutions in the main pool divided into 4 segment pools, fitness and penalty functions' precision ±0.001 (±0.002 for G7), equalities' precision ±0.0001, apply new strategy after 100*n* steps during which fitness function doesn't change (where *n* is the number of variables), stop when exact or better value of fitness function is achieved (the search also stops and reports "Solution not found" if after 1000 changes of strategy, fitness function's value is still not found).

Table 5.

COOMA results for test functions with constraints

Test function	Number of variables	Average step number	Average operations' usage, % (movement : replication : mutation)	Solution not found, runs
G1	13	10 615.1	20.81 : 2.53 : 76.66	–
G2	20	1 250 319.8	20.29 : 2.58 : 77.14	6
G3	20	16 785.2	18.85 : 0.93 : 80.22	3
G4	5	24 748.6	27.4 : 1.25 : 71.35	–
G5	4	2 263.5	29.6 : 0.67 : 69.73	1
G6	2	8 021.6	28.6 : 0.77 : 70.62	–
G7	10	1 216 486.5	18.88 : 9.33 : 71.79	9
G8	2	83.0	27.85 : 0.77 : 71.37	–
G9	7	866 667.7	20.82 : 5.0 : 74.18	54
G10	8	618 702.6	28.06 : 0.44 : 71.49	72
G11	2	1 298.19	22.72 : 1.11 : 76.17	–

The problems occurred with G7 function. COOMA was able to find optimal solution only with fitness and penalty functions' precision ±0.002 and still the average step number (1 216 486.5) is one of the biggest. And the maximal average step number (1 250 319.8) was shown for G2 function.

G10 and G9 functions lead in "Solution not found" situation (72 and 54 failures respectively).

The fitness function's value received for G5 function (average 4605.68, minimal 4220.53, maximal 5120.36) is better than reported in [8, Appendix] (5126.4981).

To demonstrate the ability of COOMA to solve multimodal functions, test function $f(x) = 5 \cdot \sum_{i=1}^{n} (1 - \cos(5 \cdot x_i))$, $x_i \in [-3; 3]$ was optimized. Function's minimal value is 0 and the number of optimums is 5^n, where n is the number of variables.

The average optimization statistics of 100 runs of COOMA for multimodal test function with different number of variables is given in Table 6. The parameters of optimization: main pool divided into 4 segment pools, unique objects' requirement of the main pool is set, fitness function's precision ±0.001, stop after 2000n steps.

Table 6.

COOMA results for multimodal test function

Number of variables	Step count	Total number of solutions in main pool	Average found optimums count	Total number of optimums	Average operations' usage, % (movement : replication : mutation)
1	2 000	12	4.8	5	22.27 : 1.36 : 76.37
2	4 000	32	18.1	25	20.7 : 1.53 : 77.77
3	6 000	160	88.6	125	21.94 : 1.21 : 76.86
4	8 000	200	169.1	625	19.77:1.68:78.55
5	10 000	200	187.9	3 125	16.12 : 1.85 : 82.03
10	20 000	200	192.4	9 765 625	14.11 : 2.21 : 83.68

Statistics shows that the maximal number of optimums from known number is found for test cases with smaller dimensions (4.8 from 5

for one variable's case and 18.1 from 25 for two variables' case). In other cases the average number of found optimums is much more less than known total number because (possible reasons) a) the number of known optimums grows faster than the number of steps, b) the ability to find solutions is limited by the total capacity of the main pool (for tests with 4 variables and more).

Discussion

As an efficient optimization algorithm COOMA should deal with several factors that complicate the application of stochastic search methods in practice (the list is not complete).

The first one is the multimodality of the objective function. Classical stochastic search algorithms are likely to find one best solution and that's a problem because a) the "best" solution is usually only a local optimum and the global optimum may be not found; b) in some cases we need to find out all the solutions to satisfy the requirements first and then select the best one among them manually according to other criterions [4]. The problem can be partially solved with special settings of the classical algorithms (such as increased level of mutations and cross-breeding in GA) and using the new approaches [5, 9, 10].

COOMA works with multimodal objective functions realizing mechanism which limits the area of replication and mutation operations. The replication and mutation scopes are regulated on several levels – by replication and mutation scope coefficients of the pools, objects and attributes. These coefficients depend on the fitness and penalty functions' values of the object, on the strategy (standard, local or global search) implemented by the pool and on the initial settings of the algorithm.

The second difficulty is constraint optimization. The goal of constraint optimization is to optimize the fitness function while satisfying a group of constraints. There are different modifications of optimization algorithms [2, 11, 13] that can deal with constraint optimization. The basic approaches of handling constraints (on the example of genetic algorithms) are described in [8].

In COOMA for each object in pool the new fitness and penalty functions' values are calculated and the changes are accepted when a) current penalty function's value is positive (the object is infeasible) and the new penalty function's value is less than its current value or b) the object is feasible and the new fitness function's value is better than its current value. Realizing this model COOMA at first finds the feasible area and then searches for the optimal solutions within this area.

The next factor is that there is no optimal parameter configuration of the certain algorithm for all types of problems. The situation is described by so called "No free lunch" theorem [17]. There are three basic approaches: a) to use some homogenized parameter configuration (which will work with some average performance for different problems); b) to change parameter configuration manually before solving each problem; c) to adopt parameters of the algorithm automatically before or while solving the problem (the idea which seems to be more perspective). The realization of the last approach is described in [7], [15, Section 10].

COOMA implements a self-adaptive two-level approach when each pool is divided into one aggregate pool and a number of segment pools which are treated as the objects of the aggregate pool. This approach helps to adopt parameters of the algorithm automatically while solving the problem.

And the last but not the least difficulty is a number of variables and their behavior in real-world problems. When the number of variables increases and their behavior becomes rather complex, even the very best algorithms are not able to find the optimal solution in a reasonable time because the searching space is too large. One of the possible ways is to separate a single big problem into several smaller ones according to the problem's structure, and realize a multi-level hierarchical algorithm. This approach is realized e. g. in [14].

COOMA realizes an Object-Relational Mapping (ORM) model supporting practically any types of parameters, variables, and associations between objects. The objects of different classes are organized in pools and pools form the hierarchical structure according to the associations between classes.

Conclusion

The main advantage of the proposed algorithm (COOMA) is that it can find optimal solutions of the problems described in the terms of object-oriented models with practically any types of parameters, variables, and associations between objects. It can easily be implemented in a form of computer program in which problem models can be built using visual interface or exported from existing database and ORM structures.

As the optimization algorithm COOMA is able to solve problems with multimodal fitness functions and a system of constraints and has a built-in self-adaptive settings' mechanism.

COOMA source code on Java is available on request.

Acknowledgements

The author thanks Prof. Victor Muraviev for his help in developing mathematical models and Elena Okhnyanskaia and Martin Kenny for their help in preparing this manuscript.

References

1. Brest J., Zumer V., and Maucec M.S. "Self-Adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization". *2006 IEEE International Conference on Evolutionary Computation* (2006): 215-22. doi:10.1109/cec.2006.1688311.
2. Chehouri, Adam, Rafic Younes, Jean Perron, and Adrian Ilinca. "A Constraint-Handling Technique for Genetic Algorithms using a Violation Factor". *Journal of Computer Science* 12, no. 7 (2016): 350-62. doi:10.3844/jcssp.2016.350.362.
3. Eberhart R., and Kennedy J. "A new optimizer using particle swarm theory." *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* (1995): 39–43. doi:10.1109/mhs.1995.494215.
4. Gao, Qin, Yi Zhong, and Xinjuan Zheng. "Hierarchical particle swarm optimization algorithm for multimodal function optimization". *Metallurgical and Mining Industry,* no. 9 (2015): 908-916.

5. Hall Matthew. "A Cumulative Multi-Niching Genetic Algorithm for Multimodal Function Optimization". *International Journal of Advanced Research in Artificial Intelligence* 1, no. 9 (2012): 6-13. doi:10.14569/ijarai.2012.010902.
6. Holland, John H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, Mass.: MIT Press, 2010.
7. Jiang, Zhong-Yang, Zi-Xing Cai, and Yong Wang. "Hybrid Self-Adaptive Orthogonal Genetic Algorithm for Solving Global Optimization Problems". *Journal of Software* 21, no. 6 (2010): 1296-307. doi:10.3724/sp.j.1001.2010.03592.
8. Koziel, Slawomir, and Zbigniew Michalewicz. "Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization". *Evolutionary Computation* 7, no. 1 (1999): 19-44. doi:10.1162/evco.1999.7.1.19.
9. Li, Xiaodong. "Niching Without Niching Parameters: Particle Swarm Optimization Using a Ring Topology". *IEEE Transactions on Evolutionary Computation* 14, no. 1 (2010): 150-69. doi:10.1109/tevc.2009.2026270.
10. Liang, Yong, and Kwong-Sak Leung. "Genetic Algorithm with adaptive elitist-population strategies for multimodal function optimization". *Applied Soft Computing* 11, no. 2 (2011): 2017-034. doi:10.1016/j.asoc.2010.06.017.
11. Long, Qiang. "A constraint handling technique for constrained multi-objective genetic algorithm". *Swarm and Evolutionary Computation* 15 (2014): 66-79. doi:10.1016/j.swevo.2013.12.002.
12. Martikainen, Jarno, and Seppo J. Ovaska. "Hierarchical Two-Population Genetic Algorithm". *International Journal of Computational Intelligence Research* 2, no. 4 (2006): 367-80. doi:10.5019/j.ijcir.2006.74.
13. Mazhoud, Issam, Khaled Hadj-Hamou, Jean Bigeon, and Patrice Joyeux. "Particle swarm optimization for solving engineering problems: A new constraint-handling mechanism". *Engineering Applications of Artificial Intelligence* 26, no. 4 (2013): 1263-273. doi:10.1016/j.engappai.2013.02.002.

14. Neoh, Siew Chin, Norhashimah Morad, Chee Peng Lim, and Zalina Abdul Aziz. "A Layered Matrix Cascade Genetic Algorithm and Particle Swarm Optimization Approach to Thermal Power Generation Scheduling". *Advances in Soft Computing Soft Computing in Industrial Applications* 39 (2007): 241-50. doi:10.1007/978-3-540-70706-6_23.
15. Parsopoulos, K. E., and M. N. Vrahatis. "Recent approaches to global optimization problems through particle swarm optimization". *Natural Computing* 1, no. 2/3 (2002): 235-306. doi:10.1023/a:1016568309421.
16. Storn, Rainer, and Kenneth Price. *Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces*. Berkeley, CA: ICSI, 1995.
17. Wolpert, D. H., and W. G. Macready. "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation* 1, no. 1 (1997): 67-82. doi:10.1109/4235.585893.
18. Zou, Dexuan. "A Self-adaptive Global Particle Swarm Optimization Algorithm for Unconstrained Optimization Problems". *International Journal of Signal Processing, Image Processing and Pattern Recognition* 7, no. 6 (2014): 183-200. doi:10.14257/ijsip.2014.7.6.15.

DATA ABOUT THE AUTHOR

Tavridovich Stanislav Alexandrovich, System Analyst, Candidate of Economics, Associate Professor
Systematica Consulting LLC
68N, B. Sampsonievsky Prospect, Saint-Petersburg, 194100, Russian Federation
stanislav@tavridovich.ru
ORCID: 0000-0002-1530-9217

ДАННЫЕ ОБ АВТОРЕ

Тавридович Станислав Александрович, системный аналитик, кандидат экономических наук, доцент
ООО «Систематика Консалтинг»
пр. Б. Сампсониевский, 68Н, г. Санкт-Петербург, 194100, Российская Федерация
stanislav@tavridovich.ru