IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), Compiégne, France, 2005: 38-44

A Compact Multiagent System based on Autonomy Oriented Computing

Xiao-Feng Xie, Jiming Liu
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
{xfxie, jiming}@comp.hkbu.edu.hk

Abstract

A compact multiagent optimization system ($MAOS_C$) based on autonomy oriented computing (AOC) is presented. Performed by a society of autonomous entities in iterative cycles, an optimization algorithm can simply be described by a macro generate-and-test behavior, which deploys a few elemental generating behaviors under conditioned reflex behaviors supported by a testing operation library. $MAOS_C$ provides a simple framework for not only realizing and comparing algorithms, but also deploying evolvable algorithms. The experimental results of $MAOS_C$ cases on benchmark functions are compared with those of other algorithms, which show its efficiency.

1. Introduction

Problems which involve numerical optimization are ubiquitous throughout scientific communities. A general numerical optimization problem F can be defined as:

$$\text{Miniminze}: f(\vec{x}) \tag{1}$$

where f is an *objective function*, $\vec{x}=\{x_1,...,x_d,...,x_D\}^T$ and *search space* (S) is a D-dimensional space bounded by all parametric constraints $x_d \in [\underline{x}_d, \overline{x}_d]$. Suppose for a certain solution $\vec{x}^*$, there exists $f(\vec{x}^*) \le f(\vec{x})$ for $\forall \vec{x} \in S \subseteq \mathbb{R}^D$, then $\vec{x}^*$ and $f(\vec{x}^*)$ are the global optimum solution and the optimum value, respectively. The *solution space* is defined as $S_O = \{\vec{x} \mid f(\vec{x}) - f(\vec{x}^*) \le \varepsilon_O\}$, where $\varepsilon_O \ge 0$ is a small value. Normally, S_O/S is quite small. As a goal for finding $\vec{x} \in S_O$ with high probability, typical challenges include: a) little *a priori* knowledge is available for each problem; and b) total computational resource is bounded.

Agent-based methods have been developed in the last few years [4][18][23], where each agent is an entity featured by knowledge as the medium and the principle of rationality as the law of behavior [20].

Autonomy oriented computing (AOC) methods [18] address the modeling of autonomy in the entities of a complex system and the self-organization of them in achieving a specific goal, which is suitable for solving hard computational problems. The complex system behaviors can emerge from the interactions of individual entities following simple behaviors [12][16][17][28].

Under the framework of autonomy oriented computing (AOC), an optimization algorithm can be described by a specific generate-and-test behavior, which is performed by every autonomous entity in iterative cycles. The law of behavior is *fast-and-frugal heuristics* [9], which makes a tradeoff on generality [29] versus specificity and avoids the trap from specificity by their very simplicity so as to generalize well to new situations [26]. Comparing with existing algorithmic frameworks [19][25][30], AOC-based optimization system addresses the capability for deploying evolvable algorithms as well as the capability for realizing and comparing algorithms.

The paper is organized as follows. In Section 2, the compact multiagent optimization system ($MAOS_C$) based on AOC is presented. In Section 3, the macro generate-and-test behavior, which is the essential behavior of each agent, is described. With the conditioned reflex behaviors provided by a testing operation library, users may focus on realizing elemental generating behaviors. In Section 4, a specific macro generate-and-test rule is implemented, where three generating rules are extracted from existing algorithms [3][14][24]. In Section 5, the $MAOS_C$ cases are applied on benchmark functions, and the experimental results are compared with those of existing algorithms [10][22][31]. In the last section, this paper is concluded.

2. Compact multiagent optimization system

$MAOS_C$ is a compact problem solver with the specific structure and operating mechanism based on AOC.

$MAOS_C$ runs in iterative cycles. For a run, the number of cycles is T. Hence the system behavior in the tth ($t \in [1,T]_{\mathbb{Z}}$) cycle only depends on the system status in the (t-1)th cycle. By running as a Markov chain process, the system can be analyzed in each cycle.

General problem solving capability arises from the interaction of declarative knowledge and procedural knowledge [1] [21]. In $MAOS_C$, declarative knowledge is represented in units called INFO elements, and procedural knowledge is represented in units called behavioral rules.

Each INFO element is described by a tuple <I_TYPE, I_CON>, where I_TYPE indicates a certain data structure that satisfies some specific conditions, and I_CON is its content that can be changed during the search process.

The memory [7][25] is used for storing retrievable INFO elements (T_INFO). Here a T_INFO element is described by a tuple <I_RC, INFO>, where I_RC is a retrieval cue. Each T_INFO element can be retrieved from a memory according to its I_RC.

If the memory is to be useful, it must be updated during a run. But updating is not encoding a new T_INFO element. Instead, only the I_CON is subjected to change. Then a T_INFO element can be referred as a trajectory that comprise of many T_INFO instances. At the *t*th cycle, a T_INFO instance is expressed as $\text{I\_TYPE}_{\text{I\_RC}}^{(t)}$.

All behavioral rules are stored in a *behavioral library*. Each rule is retrieved by a tuple <T_NAME, T_KEY>, where T_KEY indicates a virtual interface for handling specific T_INFO elements, and T_NAME indicates the realized operations, which can be controlled by specifying setting parameters within a bounded *rule-parameter space*. Each rule has default values for setting parameters.

An executable *behavioral instance* is a behavioral rule with specified parameter values. Hence an instance itself has no parameters. Here both a behavioral rule and its default instance are expressed as $R_{\text{T\_KEY}}^{\boxed{\text{T\_NAME}}}$.

At *t*=0, T_INFO elements in memory are initialized at random. At each cycle, as in a learning process, selected behaviors are executed for acquiring the memory [21].

Although AOC methods are implemented with different approaches, they have similar structures and operating mechanisms. The key elements of AOC include an environment, agents and their interactions [18].

2.1. Environment

The environment is the task-dependent computational problem on which a society of agents works [28]. As one of the main components in an AOC system, an environment usually plays two roles [18].

Firstly, it serves as the domain in which agents roam. From the static view, it contains a functional form *F* of the optimization task and provides feedback in the form of a scalar value for each potential solution within *S*.

Secondly, the environment acts as the blackboard where agents can read and/or post their local information. From this dynamic view, the environment holds a social sharing memory, called M_S, which is shared by all agents. In this sense, the environment can be regarded as an indirect communication medium among agents.

2.2. Agents

There exists a compact society of *N* homogenous agents, where each agent is an autonomous entity. Autonomy is an attribute of a self-governed, self-directed entity with respect to its own status, free from the explicit control of another entity [18]. However, only direct perturbation is prohibited; indirect influence is encouraged. The essence is that each entity is able to make decisions for itself, subject to the limitations of the available information.

At each moment, an agent is in a certain state. It, according to its behavioral rules, selects and performs its behaviors so as to achieve its goal with respect to its state. A compact agent is formalized as follows:

a) Goal: The goal of each agent is to explore the environment and find a good solution of problem *F*.

b) Evaluation function: It is used for evaluating whether or not a potential solution of *F* is a good one, which is realized by a $R_M^{\boxed{O}}(\vec{x}_a, \vec{x}_b)$ rule: for $\forall\ \vec{x}_a, \vec{x}_b \in S$, if $f(\vec{x}_a) \le f(\vec{x}_b)$, then the goodness of $\vec{x}_a$ is better than that of $\vec{x}_b$, and $R_M^{\boxed{O}}$ returns TRUE, else it returns FLASE.

The goodness evaluation implies motivation [13]: saying that one potential solution is better than another is only a way of saying that agents try to attain that solution.

c) State: It is characterized by a T_INFO set situated in a private memory, called M_A, which can only be accessed by agent itself. The T_INFO set in M_A is expressed as \$E_MA, where the symbol "\$" indicates a T_INFO set.

d) Behaviors and behavioral rules: Central to an agent is its local behaviors and behavioral rules that govern how it should act or react to the available information while determine the next state to which the entity will transit.

The law of behavior is *socially biased individual learning* (SBIL) heuristics [8][30], i.e. a mix of reinforced practice of own experience in M_A and shared information in M_S, which [5] a) gains most of the advantages of both individual learning and social learning; and b) allows cumulative improvement to the next learning cycle.

The essential behavioral rule is *generate-and-test* rule (R_{GT}), which is used to: a) generate new potential solutions by estimating the distribution of promising space according to available information situated in M_A and M_S; b) modify its own M_A and exert changes to M_S for facilitating the learning behaviors in the next cycle.

Moreover, behaviors benefit from a common feature - the presence of stochastic nature [18]. For optimization problems, two kinds of random operators are involved: a) For $\forall c \in \mathbb{N}$, the $U_{\mathbb{N}}(c)$ returns a natural number selected from $[1, c]_{\mathbb{Z}}$ at random; b) For $\forall c_l, c_u \in \mathbb{R}$, the $U_{\mathbb{R}}(\underline{c}, \overline{c})$ returns a real value selected from $[\underline{c}, \overline{c}]_{\mathbb{R}}$ at random. Moreover, $U_{\mathbb{R}}(0,1)$ is abbreviated as $U_{\mathbb{R}}$.

2.3. Interactions

The emergent behavior of an AOC system is not inherent in individual entities, which can only result from the internal interactions. Generally speaking, there are two kinds of interactions [18], namely, *interactions between entities and environment* and *interactions among entities*.

The interactions between agents and their environment are realized by background operations: at the beginning of the *t*th cycle, the INFO instances in all agents with a specified I_RC and I_TYPE are collected into a T_INFO element, called $\$\text{I\_TYPE}^{(t)}_{\text{I\_RC}}$, which is situated in M_S.

The collected elements in M_S can either be cloned or be referred from M_A of agents. Here we use former situation, i.e. the elements in M_S will be unchanged during a cycle.

Hence M_S contains observable information from the M_A of agents. It is significant since *observational learning* can lead to cumulative evolution of knowledge that no single individual could invent on its own [2].

An agent is communicated with the others by indirect interactions, which are implicit implemented through the communication medium role of M_S. While an agent behaves, it will consider the information in M_S, which has been "transferred" to the M_S by the others.

3. The macro generate-and-test behavior

Here we describe the macro generate-and-test behavior. First, the generate-and-test behavior is described. Then the normal macro R_{GT} rule is introduced. At last, a macro R_{GT} rule with a testing operation library (TOL) is presented.

3.1. The generate-and-test behavior

Since the agents can exert the changes on M_S by only modifying its own M_A during a cycle, the *generate-and-test* behavior can be represented as:

$$M_A^{(t)}, M_S^{(t)} \xrightarrow{R_{GT}} \begin{cases} \{\vec{x}^{\,(t)}\} \\ M_A^{(t+1)} \end{cases} \tag{2}$$

The essential components of an R_{GT} rule include a generating part (R_G), a solution-extracting part (R_S) and a testing part (R_T). Without loss of generality, it can be supposed that only the generating part can create new information that contains new potential solution (s), and the testing part only produces conditioned reflex behavior.

The R_G part estimates the distribution of promising solutions and samples a T_INFO set, called $\$\text{E\_G}^{(t)}$:

$$M_A^{(t)}, M_S^{(t)} \xrightarrow{R_G} \$\text{E\_G}^{(t)} \tag{3}$$

The $\$\text{E\_G}^{(t)}$ contains potential solution(s), which is exported by the solution-extracting part (R_S):

$$\$\text{E\_G}^{(t)} \xrightarrow{R_S} \{\vec{x}^{\,(t)}\} \tag{4}$$

The R_S part has no influence on the problem-solving process. It just exports potential solution(s) during a run.

During a run, the T_INFO elements in LTM are subjected to be updated by the testing rule (R_T):

$$\$\text{E\_MA}^{(t)}, \$\text{E\_G}^{(t)} \xrightarrow{R_T} M_A^{(t+1)} \tag{5}$$

Under specified M_A and M_S, various R_{GT} rules can be realized by using full or part of the available information.

3.2. The macro R_{GT} rule (R_{GTM})

The macro R_{GT} rule (R_{GTM}) manages an array of R_{GT} rules by a deploying rule (R_{DEP}) [30]. At this time, such R_{GT} rules are defined as elemental R_{GT} rule (R_{GTE}).

Here a simple proportional-deploying rule ($R_{DEP}^{\boxed{P}}$) is used. It can be viewed as a single-layer network, where an *association strength* value w_S ($w_S \geq 0$, $w_S \in \mathbb{R}$) is assigned to each output node associated with an R_{GTE} rule. At each time, only an output node is activated, where the selection probability for each node is equal to $w_S / (\sum w_S)$.

For an R_{GTM} rule, various R_{GT} rules can be realized by specifying different w_S value set. In particular, if we keep the w_S value for a specified R_{GTE} rule larger than 0 and keep the w_S values for other R_{GTE} rules equal to 0, then the R_{GTM} rule is equivalent to the specified R_{GTE} rule.

If all R_{GTE} rules in the R_{GTM} rule do not share T_INFO elements in M_A and M_S, then the performance of an R_{GTM} rule is simply the weighted performance of its R_{GTE} rules.

Some R_{GTE} rules may share the T_INFO elements in M_A and/or M_S. During a run, those cooperative R_{GTE} rules may facilitate each others by searching in a reduced space of possibilities [11] and escaping from the local traps. Of course, the successful cooperation often benefits from the available knowledge on sharing T_INFO elements.

3.3. R_{GTM} with a testing operation library ($R_{\text{GTM}}^{\boxed{\text{TOL}}}$)

In an $R_{\text{GTM}}^{\boxed{\text{TOL}}}$ rule, each R_{GTE} rule is represented by an R_G rule under the *testing operation library* (TOL), i.e.,

$$\{\ R_{GTE}\ \} \rightarrow \text{GOL+TOL} \tag{6}$$

where the *generating operation library* (GOL) contains a set of R_G instances, and the TOL is employed for representing some predefined R_T instances. Each R_T instance is accessed by certain $\$\text{E\_MA}_{\text{G}*}$ and $\$\text{E\_G}_{\text{G}*}$, which are the input and the output of an R_G instance, respectively. The R_{DEP} and the GOL supports generating new information, and the TOL provides conditioned reflex behaviors for updating the memory automatically.

The TOL contains a set of *testing operation tables* (TOTs). Each TOT table is defined for specified $\$\text{E\_G}_* \subseteq \$\text{E\_G}$ and $\$\text{E\_MA}_* \subseteq \$\text{E\_MA}$, which contains a set of *elemental testing operations* (TO_E). Each TO_E responds to one element E_MA_* in $\$\text{E\_MA}_*$:

$$\text{TO}_\text{E}: \$\text{E\_O}_\text{E}{}^{(t)} \xrightarrow{R_{TOE}} \text{E\_MA}_* \tag{7}$$

where R_{TOE} is an *elemental testing operation* rule. $\$\text{E\_O}_\text{E}{}^{(t)} \subseteq \$\text{E\_O}^{(t)}$, $\$\text{E\_O}=\$\text{E\_MA}_{*\text{C}}{}^{(t)} \cup \$\text{E\_G}_*{}^{(t)}$, where $\$\text{E\_MA}_{*\text{C}}{}^{(t)}$ is a T_INFO set cloned from $\$\text{E\_MA}_*{}^{(t)}$. Using of $\$\text{E\_MA}_{*\text{C}}{}^{(t)}$ in $E_O ensures that we need not mind the executing sequence of the TO_E operations.

For the $\$\text{E\_G}_{\text{G}*}$ and $\$\text{E\_MA}_{\text{G}*}$ of an R_G rule, with the TOT table retrieved from TOL by the $\$\text{E\_G}_{\text{G}*}$, the R_T part is implicitly realized by executing all TO_E operations corresponding to all the elements in the $\$\text{E\_MA}_{\text{G}*}$.

A TOT table is capable of supporting a number of R_T operations if we consider all the possible combinations of elements in its $\$\text{E\_MA}_*$. Hence, the TOL can support various R_G rules by realizing a few TOT tables.

Using the TOL also brings two advantages: a) it may facilitate realizing new R_G part since some nontrivial properties of T_INFO elements may be known in advance; b) it may be possible to know more details about the difference for some R_{GTE} rules, especially for those with same $\$\text{E\_G}_{\text{G}*}$, that are sharing the same TO_E instances.

The R_{GTM} with TOL is apt to be evolvable during different runs: a) to add a new TO_E instance into a certain TOT table or to add a new TOT table into the TOL will have no impacts on existing R_G rules; b) as a TO_E instance of a certain TOT table is removed, those R_G rules that use the TO_E instance are unsupported.

The evolving of GOL is achieved by adding/removing R_G instances instead of modifying the R_G rules so as to reserve the R_G instances with established knowledge.

The adaptability is normally achieved by adjusting R_{DEP}, since both the GOL and the TOL are relatively stable during different runs and they contain no parameters.

4. The implemented $R_{\text{GTM}}^{\boxed{\text{TOL}}}$ rule

To implement the $R_{\text{GTM}}^{\boxed{\text{TOL}}}$ rule, the following steps are required: a) register supported {I_TYPE} for memory elements; b) specify $E_MA and $E_G; c) realize some R_{TOE} rules, R_G rules and R_{DEP} rule; d) define the TOT tables in TOL based on R_{TOE} instances; and e) specify R_G instances and adjust w_s value set for R_{DEP} rule.

Here we only support the simplest I_TYPE, i.e., $\vec{x} \in S$.

Two $R_{TOE}(\vec{x}_a, \vec{x}_b)$ rules, i.e. $R_{TOE}^{\boxed{D}}$ and $R_{TOE}^{\boxed{G}}$, are realized, where each replaces $\vec{x}_a$ by $\vec{x}_b$ when $isUpdate() \equiv$ TRUE.

We use three T_INFO elements, i.e., $E_MA={ $\vec{x}_P$, $\vec{x}_R$, $\vec{x}_O$ }. Hence the T_INFO elements in M_S include $\underline{\$}\vec{x}_P$, $\underline{\$}\vec{x}_R$, and $\underline{\$}\vec{x}_O$. Here we only care for two of them, i.e., $\underline{\$}\vec{x}_P^{(t)}$={ $\vec{x}_{P(i)}^{(t)} \mid i \in [1, N]_\mathbb{Z}$ } and $\underline{\$}\vec{x}_R^{(t)}$={ $\vec{x}_{R(i)}^{(t)} \mid i \in [1, N]_\mathbb{Z}$ }.

As shown in Table 1, here only one TOT table, which responds to $E_G={ $\vec{x}_C$ } and $E_MA, is implemented.

Table 1. TOT table for $E_G={ $\vec{x}_C$ }

E_MA	E_O_E	R_{TOE}: *isUpdate*()
$\vec{x}_P$	$\vec{x}_C$	$R_{TOE}^{\boxed{G}} : R_M^{\boxed{O}}(\vec{x}_C, \vec{x}_P)$
$\vec{x}_R$	$\vec{x}_C$	$R_{TOE}^{\boxed{D}}$: TRUE
$\vec{x}_O$	$\vec{x}_R$	$R_{TOE}^{\boxed{D}}$: TRUE

Some nontrivial properties about $E_MA elements can be revealed from the TOT table. For each agent at each cycle, $\vec{x}_P$ stores the best solution ever obtained, $\vec{x}_R$ stores the most recently solution, and $\vec{x}_O$ stores the last $\vec{x}_R$.

Besides, the best solution in $\underline{\$}\vec{x}_P^{(t)}$ is defined as $\vec{x}_G^{(t)}$, which represents the best solution among agents.

Based on the TOL, three R_G rules are implemented, where the main bodies are extracted from existing algorithms. Once $\vec{x}_C \notin S$, the testing part will be aborted. To ensure $\vec{x}_C \in S$, different boundary handling methods may be integrated with the R_G part according to knowledge about T_INFO elements, if necessarily.

4.1. Social impact R_G rule ($R_G^{\boxed{SI}}$)

The $R_G^{\boxed{SI}}$ rule is originated from an Electromagnetism-like (EM) heuristic algorithm [3], which intends to make an analogy with the attraction-repulsion mechanism for sampling solutions. However, such mechanism is actually realized by comparing the goodness of two points, which is hardly explained by its physical metaphor.

Here we explain its behavior using a metaphor inspired by the social impact theory [15], which declares that the likelihood of a person responding to a social influence will increase with: a) *strength*: how important the influencing person is to you; b) *immediacy*: how close the person is to you at the time of the influence attempt.

The $R_G^{\boxed{SI}}$ rule uses a T_INFO element $\vec{x}_R^{(t)}$ in M_A and a T_INFO element $\underline{\$}\vec{x}_R^{(t)}$ in M_S. It is realized as follows:

a) Evaluate the *social influence strength* information into $\vec{V}_{ISS}$ according to $\underline{\$}\vec{x}_R^{(t)}$:

$$V_{ISS,i}^{(t)} = \exp\left(\frac{-D \cdot (f(\vec{x}_{R(i)}^{(t)}) - f_G^{(t)})}{\sum_{j=1}^{N} (f(\vec{x}_{R(j)}^{(t)}) - f_G^{(t)})} \right), \ \forall i \tag{8}$$

where f_G is the goodness value of the best solution in $\underline{\$}\vec{x}_R^{(t)}$.

Its *personal influence* (V_{ISP}) is the $V_{ISI,i}^{(t)}$ as $\vec{x}_{R(i)}^{(t)} = \vec{x}_R^{(t)}$.

b) Integrate the *influential force* vector $\vec{V}_{IF}$ according to the *influence strength* and *immediacy* of the states of others, i.e., for $\forall\ i \in [1, N]_\mathbb{Z}$ as $\| \vec{x}_{R(i)}^{(t)} - \vec{x}_R^{(t)} \| \neq 0$,

$$\vec{V}_{IF}^{(t)} = \sum \frac{c_{SIGN}\cdot V_{ISP}^{(t)}\cdot V_{ISS,i}^{(t)}\cdot(\vec{x}_{R(i)}^{(t)}-\vec{x}_R^{(t)})}{\| \vec{x}_{R(i)}^{(t)}-\vec{x}_R^{(t)} \|},\ \forall i \tag{9}$$

where c_{SIGN} =1 if $V_{ISS,i}^{(t)} \geq V_{ISP}^{(t)}$, else c_{SIGN} =-1.

c) Normalize $\vec{V}_{IF}$ to $\vec{V}_{IFN}$: $\vec{V}_{IFN}^{(t)} = \vec{V}_{IF}^{(t)} \Big/ \left\| \vec{V}_{IF}^{(t)} \right\|$;

d) Generate $\vec{x}_C^{(t)}$: if $f(\vec{x}_R^{(t)}) \equiv f_G^{(t)}$, then $\vec{x}_C^{(t)} = \vec{x}_R^{(t)}$, else for the *d*th dimension [3], if $V_{IFN,d}^{(t)} > 0$, then

$$x_{C,d}^{(t)} = \begin{cases} x_{R,d}^{(t)} + U_{\mathbb{R}}\cdot(\overline{x}_d - x_{R,d}^{(t)})\cdot V_{IFN,d}^{(t)} & \text{IF } V_{IFN,d}^{(t)} > 0 \\ x_{R,d}^{(t)} + U_{\mathbb{R}}\cdot(x_{R,d}^{(t)} - \underline{x}_d)\cdot V_{IFN,d}^{(t)} & \text{ELSE} \end{cases} \tag{10}$$

The social impact rule has no parameter.

4.2. Differential evolution R_G rule ($R_G^{\boxed{DE}}$)

The $R_G^{\boxed{DE}}$ rule is extracted from the behavior of an individual in differential evolution (DE) [24], where its rationality is to *imitate the successful* [5] in some aspects.

By using one element in M_A, i.e., $\vec{x}_P^{(t)}$, and one element in M_S, i.e. $\underline{\$}\vec{x}_P^{(t)}$, it generates $\vec{x}_C^{(t)}$ as in the following steps:

a) Create a sample T_INFO set $\$\vec{x}_S^{(t)}$:

$$\$\vec{x}_S^{(t)} = \{\vec{x}_{S(i)}^{(t)} \mid i \in [1, 2\cdot C_{NV}]_{\mathbb{Z}}\} \tag{11}$$

where for $\forall i$, $\vec{x}_{S(i)}^{(t)} = \vec{x}_{P(U_{\mathbb{N}}(N))}^{(t)}$ is selected from $\underline{\$}\vec{x}_P^{(t)}$ at random. $C_{NV} \in \mathbb{N}$ is the number of *difference vectors* [24];

b) Assigns $\vec{x}_C^{(t)} = \vec{x}_P^{(t)}$ and $c_{DR} = U_{\mathbb{N}}(D)$;

c) For the *d*th dimension, if $U_{\mathbb{R}} < C_{CR}$ or $d \equiv c_{DR}$, then

$$x_{C,d}^{(t)} = x_{G,d}^{(t)} + \sum_{n=1}^{C_{NV}} (x_{S(2\cdot n),d}^{(t)} - x_{S(2\cdot n-1),d}^{(t)}) \Big/ C_{NV} \tag{12}$$

where $C_{CR} \in [0,1]_{\mathbb{R}}$ is *crossover factor* [24], and c_{DR} ensures the variation is at least in one dimension.

Since $\vec{x}_P^{(t)}$ is a steady-state INFO element, the boundary handling method for ensuring $\vec{x}_C^{(t)} \in S$ can be realized easily. For the *d*th dimension, if $x_{C,d}^{(t)} \notin [\underline{x}_d, \overline{x}_d]$, then

$$x_{C,d}^{(t)} = U_{\mathbb{R}}(\underline{x}_d, \overline{x}_d) \tag{13}$$

The default parameter values include: C_{NV} =2, C_{CR} =0.1.

4.3. Particle swarm R_G rule ($R_G^{\boxed{PS}}$)

The $R_G^{\boxed{PS}}$ rule is extracted from the behavior of a particle in particle swarm optimization (PSO) [14]. The psychological assumption for the behavior of a particle is [13]: in search for consistent cognition in a society, an entity will tend to retain its own best beliefs, and will also consider the beliefs of its colleagues.

The particle swarm rule uses all three elements in M_A, and one element in M_S, i.e. $\underline{\$}\vec{x}_P^{(t)}$.

It generates $\vec{x}_C^{(t)}$ by using the experiences of its own and its colleagues, for the *d*th dimension:

$$\begin{aligned} x_{C,d}^{(t)} := x_{R,d}^{(t)} + c_K\cdot((x_{R,d}^{(t)} \tfrac{\Delta}{(d)} x_{O,d}^{(t)}) \\ + C_a\cdot U_{\mathbb{R}}\cdot(x_{P,d}^{(t)} \tfrac{\Delta}{(d)} x_{R,d}^{(t)}) \\ + C_b\cdot U_{\mathbb{R}}\cdot(x_{G,d}^{(t)} \tfrac{\Delta}{(d)} x_{R,d}^{(t)})) \end{aligned} \tag{14}$$

where $c_K = 2/(\sqrt{\varphi\cdot(\varphi-4)}+\varphi-2)$ is called as *constriction factor* [6] valid as $\varphi = C_a + C_b > 4$, and $x_a \frac{\Delta}{(d)} x_b$ calculates the difference for $\forall x_a, x_b \in \mathbb{R}$ at the *d*th dimension of *S*.

Normally, a plain $\frac{\Delta}{(d)}$ (called $\frac{\Delta P}{(d)}$) , which is equivalent to the "-" operator, is used [6]. However, studies showed that such a generating rule may suffer from a severe degenerated performance caused by too many unnecessary mutations for $\vec{x}_R$ occurred at the boundary of *S* [32].

The success by the *Periodic* boundary handling mode [32] implies that infinite space is much preferred by each particle so as to eliminating unnecessary mutations. Here a circled version is applied for each dimension, i.e. which provides an infinite vision while keeping $\vec{x}_C^{(t)} \in S$.

As a dimension is rolled into a cycle, there are two directions for calculating the difference of two points on the cycle. Here the direction with a short distance is used. For $\forall x_a, x_b \in [\underline{x}_d, \overline{x}_d]$, a *cycled* $\frac{\Delta}{(d)}$ operator (called $\frac{\Delta C}{(d)}$) is:

$$x_a \frac{\Delta C}{(d)} x_b = \begin{cases} x_{\Delta ab} + \underline{\overline{x}}_d & \text{IF } x_{\Delta ab} < -\underline{\overline{x}}_d/2 \\ x_{\Delta ab} - \underline{\overline{x}}_d & \text{IF } x_{\Delta ab} > \underline{\overline{x}}_d/2 \\ x_{\Delta ab} & \text{ELSE} \end{cases} \tag{15}$$

where $\underline{\overline{x}}_d = \overline{x}_d - \underline{x}_d$ and $x_{\Delta ab} = x_a - x_b$.

Finally, the *d*th dimension of $\vec{x}_C^{(t)}$ is repaired as:

$$x_{C,d}^{(t)} = \begin{cases} \overline{x}_d - (\underline{x}_d - x_{C,d}^{(t)})\% \underline{\overline{x}}_d & \text{IF } x_{C,d}^{(t)} < \underline{x}_d \\ \underline{x}_d + (x_{C,d}^{(t)} - \overline{x}_d)\% \underline{\overline{x}}_d & \text{IF } x_{C,d}^{(t)} > \overline{x}_d \end{cases} \tag{16}$$

The default parameter values include: C_a=C_b=2.05 [6].

5. Results and discussions

All INFO elements in M_A of all agents are initialized in *S* at random. By taking the initialization stage into account, total evaluation times (T_E) can be calculated as:

$$T_E = N\cdot(T+N_{MA}) \approx N\cdot T \tag{17}$$

where N_{MA} is the number of T_INFO elements in M_A.

By using three default R_G instances, four R_{GTM} cases with different w_S value sets are listed in Table 2.

Table 2. R_{GTM} cases with different w_S value sets

Case\\R_G	$R_G^{\boxed{SI}}$	$R_G^{\boxed{DE}}$	$R_G^{\boxed{PS}}$
SI#	1	0	0
DE#	0	1	0
PS#	0	0	1
DP#	0	0.5	0.5

Here two function sets are used for evaluating the performance of the MAOS$_C$ cases.

The low-dimensional function set contains those Dixon and Szegö functions [3] tested by Yao and Liu [31], which are generally considered as easy problems [27]. Table 3 summaries their basic information (dimension D and optimum value f^*) and published mean results by fast evolutionary programming (FEP) [31] averaged over 50 runs. For each run of FEP, T_E=10000.

Table 3. Summary of Dixon and Szegö functions and mean results by FEP (T_E=10000) [31]

Function [3] [31]	D	f^*	FEP[31]
Branin (*BR*)	2	0.39789	0.398
Goldstein Price (*GP*)	2	3	3.02
Six Hump Camel (*C6*)	2	-1.03163	-1.03
Hartman3D (*H3*)	3	-3.86278	-3.86
Hartman 6D (*H6*)	6	-3.32237	-3.27
Shekel5 (*S5*)	4	-10.15318	-5.52
Shekel7 (*S7*)	4	-10.40292	-5.52
Shekel10 (*S10*)	4	-10.53639	-6.57

Table 4 summaries the mean results by MAOS$_C$ cases as N=10 and T=100 ($T_E \approx 1000$) averaged over 500 runs, where a result in boldface indicates that it is no worse than the result by FEP as in same precision. Here the SI# is not performed well. But it can be improved by incorporating a local search strategy [3] or by specifying larger N and T. Both PS# and DE# perform well for several cases. With their combinations of good features of PS# and DE#, DP# performs well in all cases by comparing to FEP in only tenth evaluation times. Besides, it also shows that sometimes DE# performs better than both PS# and DE#, e.g., *H3*, which may due to the interactions on shared $\vec{x}_P$.

Table 4. Mean results by R_{GTM} cases ($T_E \approx 1000$)

F	SI#	DE#	PS#	DP#
BR	0.40787	**0.39795**	**0.39789**	**0.39793**
GP	3.24542	4.62017	**3.00000**	**3.00589**
C6	-1.02493	**-1.03163**	**-1.03163**	**-1.03163**
H3	**-3.85608**	-3.84933	**-3.86070**	**-3.86278**
H6	-3.23163	**-3.29149**	-3.26733	**-3.27604**
S5	-5.05961	**-6.41440**	-5.23902	**-5.91482**
S7	**-6.00746**	**-7.10326**	**-5.75942**	**-6.96877**
S10	-5.93186	**-7.49203**	-5.87551	**-7.12365**

Ho et al. [10] have tested seven algorithms, including IEA, OEGA, UEGA, TEGA, BLXGA, BOA [22] and OGA, for solving 10-D functions as T_E=10000. Among these algorithms, BOA has the best performance [10].

Table 5 summaries the f^*, the mean results by BOA in [10], and the mean results by MAOS$_C$ cases (except for SI#) as N=10 and T=1000 ($T_E \approx 10000$) averaged over 500 runs. Here PS# performs better than BOA for 6 problems, while both DE# and DP# perform better than BOA for 10 problems. For f_1 and f_9, DP# performs as No.3 and No.2 among the algorithms used by Ho et al. [10], respectively.

Table 5. Mean results for 10D functions ($T_E \approx 10000$)

F	f^*	BOA [10]	DE#	PS#	DP#
f_1	-12.1598	-12.151	-12.0943	-11.5503	-12.1201
f_2	-18	-12.29	**-16.1427**	**-15.8501**	**-15.9473**
f_3	0	0.77	**0.034**	3.276	**0.002**
f_4	0	5.32	**1.09689**	13.48593	**1.12454**
f_5	0	0.0077	**4.84E-21**	**2.27E-12**	**2.38E-11**
f_6	-18.5027	-18.01	**-18.0569**	-13.5354	**-18.1481**
f_7	0	0.019	**0.00116**	**0.00130**	**0.00702**
f_8	0	0.93	**0.11513**	1.17806	**0.01390**
f_9	0	8.4	152.7844	682.8690	90.6056
f_{10}	0	8.93	**5.26981**	**6.96009**	**5.1077**
f_{11}	0	10.067	**1.348**	**7.192**	**0.416**
f_{12}	0	1.008	**0.02544**	**0.14677**	**0.06032**

6. Conclusions

In this paper, we have proposed a compact multiagent system (MAOS$_C$) based on autonomy oriented computing (AOC). Performed by a society of autonomous entities in iterative cycles, an optimization algorithm can simply be described by a macro generate-and-test behavior.

The MAOS$_C$ system has the following characteristics:

Firstly, it is a framework for realizing various numerical optimization algorithms in an easy way. Several existing algorithms, including particle swarm optimization (PSO), differential evolution (DE), and electromagnetism-like heuristic algorithm (EM), have been implemented.

Secondly, it is a natural framework for comparing different generate-and-test behaviors. Based on a simple TOL, we may focus on their generating parts by utilizing some known properties of existing declarative knowledge.

Thirdly, it is capable of producing macro algorithms by deploying some simple R_G instances, where each covers a part of problems, for cooperative search. As shown in the experiments, the macro R_{GT} rule performs better than each of its elemental R_G instances on specified problem sets.

Finally, it may provide a base towards realizing user-oriented algorithms in future. The MAOS$_C$ is not intended to be a universal solver at all time. Instead, it is focused on the problems, which a certain user feels interested in, varied at different periods. The components of the R_{GTM} rule, i.e., the deploying rule, the GOL and the TOL, are subjected to be evolvable for such a purpose. As time goes by, we may only focused on tackling those problems poorly solved by all existing R_G instances while those obsolete R_G instances may be discarded by some methods, e.g., minimum set cover, based on those solved examples.

Acknowledgment

The reported work has been supported by an RGC CERG research grant (HKBU 2121/03E).

References

[1] J. Anderson, "ACT: A simple theory of complex cognition," American Psychologist, vol. 51, pp. 355-365, 1996.

[2] A. Bandura, Social Foundations of Thought and Action: A Social Cognitive Theory. NJ: Prentice-Hall, 1986.

[3] S. I. Birbil and S.-C. Fang, "An electromagnetism-like mechanism for global optimization," J. Global Optimization, vol. 25, pp. 263-282, 2003.

[4] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," Proc. Natl. Acad. Sci. USA, vol. 99, pp. 7280-7287, 2002.

[5] R. Boyd and P. J. Richerson, The Origin and Evolution of Cultures. New York: Oxford University Press, 2005.

[6] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," IEEE Trans. Evol. Comput., vol. 6, pp. 58-73, 2002.

[7] K. A. Ericsson and W. Kintsch, "Long-term working memory," Psychol. Review, vol. 102, pp. 211-245, 1995.

[8] B. G. Galef, "Why behaviour patterns that animals learn socially are locally adaptive," Animal Behaviour, vol. 49, pp. 1325-1334, 1995.

[9] G. Gigerenzer and D. G. Goldstein, "Reasoning the fast and frugal way: Models of bounded rationality," Psychol. Review, vol. 103, pp. 650-669, 1996.

[10] S. Y. Ho, L. S. Shu, J. H. Chen, "Intelligent evolutionary algorithms for large parameter optimization problems," IEEE Trans. Evol. Comput., vol. 8, pp. 522-541, 2004.

[11] T. Hogg and C. P. Williams, "Solving the really hard problems with cooperative search," Proc. of AAAI, 1993.

[12] B. Hu, J. Liu, and X. Jin, "From local behaviors to global performance in a multi-agent system," IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology, pp. 309-315, 2004.

[13] J. Kennedy, "The particle swarm: Social adaptation of knowledge," Int. Conf. on Evolutionary Computation, Indianapolis, Indiana, 1997.

[14] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," IEEE Int. Conf. on Neural Networks, 1995.

[15] B. Latané, "The psychology of social impact," American Psychologist, vol. 36, pp. 343-356, 1981.

[16] J. Liu and Y. Y. Tang, "Adaptive image segmentation with distributed behavior-based agents," IEEE Trans. on PAMI, vol. 21, pp. 544-551, 1999.

[17] J. Liu, J. Han, and Y. Y. Tang, "Multi-agent oriented constraint satisfaction," Artificial Intelligence, vol. 136, pp. 101-144, 2002.

[18] J. Liu, X. Jin, and K. C. Tsui, Autonomy Oriented Computing (AOC): From Problem Solving to Complex Systems Modeling: Kluwer Academic Publishers, 2005.

[19] M. Milano and A. Poli, "MAGMA: A multiagent architecture for metaheuristics," IEEE Trans. Systems, Man and Cybernetics - Part B, vol. 34, pp. 925-941, 2004.

[20] A. Newell, "The knowledge level," Artificial Intelligence, vol. 18, pp. 87-127, 1982.

[21] H. Okano, T. Hirano, and E. Balaban, "Learning and memory," Proc. Natl. Acad. Sci. USA, vol. 97, pp. 12403-12404, 2000.

[22] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," Genetic and Evolutionary Computation Conference, 1999.

[23] A. S. Rao and M. Georgeff, "BDI Agents: From theory to practice," Int. Conf. on Multi-Agent Systems, 1995.

[24] R. Storn and K. V. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," J. Global Optimization, vol. 11, pp. 341-359, 1997.

[25] E. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin, "Adaptive memory programming: A unified view of metaheuristics," European J. Operational Research, vol. 135, pp. 1-16, 2001.

[26] P. M. Todd and G. Gigerenzer, "Précis of simple heuristics that make us smart," Behavioral and Brain Sciences, vol. 23, pp. 727-741, 2000.

[27] A. Törn, M. M. Ali, and S. Viitanen, "Stochastic global optimization: Problem classes and solution techniques," J. Global Optimization, vol. 14, pp. 437-447, 1999.

[28] K. C. Tsui and J. Liu, "Multiagent diffusion and distributed optimization," Int. Joint Conf. on Autonomous Agents and Multiagent Systems, Melbourne, Australia, pp. 169-176, 2003.

[29] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Trans. Evol. Comput., vol. 1, pp. 67-82, 1997.

[30] X. F. Xie and W. J. Zhang, "SWAF: Swarm algorithm framework for numerical optimization," Genetic and Evolutionary Computation Conference, 2004.

[31] X. Yao, Y. Liu, "Evolutionary programming made faster," IEEE Trans. Evol. Comput., vol. 3, pp. 82-102, 1999.

[32] W. J. Zhang, X. F. Xie, and D. C. Bi, "Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space," Congress on Evolutionary Computation, Oregon, USA, 2004.