[Cooperative Group Optimization] http://www.wiomax.com/optimization

A cooperative group optimization system

Xiao-Feng Xie · Jiming Liu · Zun-Jing Wang

Abstract A cooperative group optimization (CGO) system is presented to implement CGO cases by integrating the advantages of the cooperative group and low-level algorithm portfolio design. Following the nature-inspired paradigm of a cooperative group, the agents not only explore in a parallel way with their individual memory, but also cooperate with their peers through the group memory. Each agent holds a portfolio of (heterogeneous) embedded search heuristics (ESHs), in which each ESH can drive the group into a stand-alone CGO case, and hybrid CGO cases in an algorithmic space can be defined by low-level cooperative search among a portfolio of ESHs through customized memory sharing. The optimization process might also be facilitated by a passive group leader through encoding knowledge in the search landscape. Based on a concrete framework, CGO cases are defined by a script assembling over instances of algorithmic components in a toolbox. A multilayer design of the script, with the support of the inherent updatable graph in the memory protocol, enables a simple way to address the challenge of accumulating heterogeneous ESHs and defining customized portfolios without any additional code. The CGO system is implemented for solving the constrained optimization problem with some generic components and only a few domain-specific components. Guided by the insights from algorithm portfolio design, customized CGO cases based on basic search operators can achieve competitive performance over existing algorithms as compared on a set of commonly-used benchmark instances. This work might provide a basic step toward a user-oriented development framework, since the algorithmic space might be easily evolved by accumulating competent ESHs.

X.-F. Xie (✉)
The Robotics Institute, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
e-mail: xfxie@cs.cmu.edu

J. Liu
Department of Computer Science, Hong Kong Baptist
University, Kowloon Tong, Hong Kong
e-mail: jiming@comp.hkbu.edu.hk

Z.-J. Wang
Department of Physics, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
e-mail: zwang@cmu.edu

1 Introduction

Under a suitable formulation, an optimization problem can be cast to a search in a *landscape* (Stadler and Happel 1999) over a space of states, which is conceptually simple, but often computationally difficult. The paradigm is general from computational, evolutionary and cultural perspectives.

Over the past few decades, many general-purpose optimization algorithms have been proposed. Single-start examples include hill climbing, simulated annealing, tabu search, and plenty of other stochastic local search heuristics (Hoos and Stutzle 2004). Population-based examples include genetic algorithm (GA) (Deb 2000; Farmani and Wright 2003), evolution strategy (ES) (Mezura-Montes and Coello 2005; Runarsson and Yao 2005; Wang et al. 2008), memetic algorithm (MA) (Chen et al. 2012; Ong et al. 2006), cultural algorithm (CA) (Becerra and Coello 2006; Reynolds et al. 2008), ant colony optimization (ACO) (Socha and Dorigo 2008), particle swarm optimization (PSO) (Kennedy et al. 2001; Lu and Chen 2008), differential evolution (DE) (Becerra and Coello 2006; Omran

and Engelbrecht 2009; Price et al. 2005), social cognitive optimization (SCO) (Xie et al. 2002), group search optimizer (GSO) (He et al. 2009), and some other algorithms (Barkat Ullah et al. 2009; Liu et al. 2007; Takahama and Sakai 2005; Xie and Liu 2009; Xie and Zhang 2004).

On the one hand, existing algorithms have explored various metaphors, in which evolution and learning (Hinton and Nowlan 1987) are central issues for the adaptability in different landscapes. Algorithms inspired by biological evolution, such as GA and ES, indicate the power of emergent collective intelligence at the population level. Both CA and MA try to emulate cultural evolution upon a canonical population: CA (Reynolds et al. 2008) is a dual inheritance system, which uses a belief space to provide positive clues for the population; and MA (Chen et al. 2012; Ong et al. 2006) stresses that individual learning, which is normally realized by local search heuristics, can guide the evolution (Hinton and Nowlan 1987). From the viewpoint of learning, evolution can be seen as evolutionary learning (Curran and O'Riordan 2006), in which public information can be regarded as a collective memory used by cooperative search entities. In ACO, heuristics are owned by reflex agents called *ants* (Socha and Dorigo 2008) without individual memory, which are cooperated on inadvertent public memory.

Groups are very common in animals (Galef 1995; Leonard et al. 2012) and human communities (Dennis and Valacich 1993; Goncalo and Staw 2006; Nemeth 1986; Paulus 2000; Tomasello et al. 1993). Well-studied group phenomena include collective cognition (Leonard et al. 2012), cultural learning (Boyd et al. 2011; Curran and O'Riordan 2006; Galef 1995; Tomasello et al. 1993), and group intelligence (Goncalo and Staw 2006; Paulus 2000; Satzinger et al. 1999; Woolley et al. 2010), of which can promote adaptability and productivity. From an algorithmic viewpoint, a group can be represented by multiple agents that search the solutions in a common environment (Platon et al. 2007), in which the problem landscape can be seen as a common metric space associated with a computational or cognitive representation. In a *cooperative group*, each agent possesses a limited search capability through a mix of both individual and social learning (Boyd et al. 2011; Curran and O'Riordan 2006; Galef 1995; Tomasello et al. 1993). Compared to a *stigmergic group*, e.g., ACO (Socha and Dorigo 2008), the agents in a cooperative group can preserve some promising minority patterns (Nemeth 1986) with their personal memories (Ericsson and Kintsch 1995; Glenberg 1997) while they search in a parallel way. Compared to a *nominal group* (Dennis and Valacich 1993) that individuals work separately, the cooperative agents also interact with their peers through the shared group memory (Danchin et al. 2004; Dennis and Valacich 1993). On the other hand, existing algorithms have provided plenty of search

components, and hybrid metaheuristics has been widely used for optimization. In Ong et al. (2006), local search strategies were adaptively employed. In Runarsson and Yao (2005), ES was improved by using a differential variation. DE has been hybridized with different algorithms (Omran and Engelbrecht 2009; Zhang and Xie 2003). In OEA (Liu et al. 2007), several evolutionary operators searched together. There are some significant practices in multimethod (Vrugt et al. 2009), multi-operator (Elsayed et al. 2011; Elsayed et al. 2012, 2013), and ensemble algorithms (Mallipeddi et al. 2010a, b; Mallipeddi and Suganthan 2010). These algorithms, either in pure or hybrid forms, have been shown to be *competent* over different sets of problem instances, as measured using some quality metrics (Hoos and Stutzle 2004).

The motivation behind metaheuristic frameworks (Lau et al. 2007; Parejo et al. 2012) might be explained using the No Free Lunch theorems (Wolpert and Macready 1997) that any algorithm can only be competent on some problem instances. Conceptual frameworks (Milano and Poli 2004; Raidl 2006; Taillard et al. 2001; Talbi 2002) have been proposed for providing common terminologies and classification mechanisms. Typical software realizations include HeuristicLab (Wagner 2009), ParadisEO (Cahon et al. 2004), and JCLEC (Ventura et al. 2008), etc. Within these frameworks, different algorithm paradigms are coded with some specific interfaces, and are then configured using configuration files to pick instances in a toolbox (Anderson 2005; Gigerenzer and Selten 2001; Raidl 2006) of reusable components. Some frameworks provide basic relay and teamwork hybrids (Parejo et al. 2012; Talbi 2002), and some frameworks include advanced mechanisms, e.g., "request, sense and response" (Lau et al. 2007) and "operator graph" (Wagner 2009), that facilitates rapid prototyping of hybrid metaheuristics. Each framework can provide an algorithmic space, but walking within the space might be inefficiently, since algorithmic components are effective only as they are embedded in certain environments, and no easy hint is available to understand their behavioral changes. For end users, advanced knowledge is needed to adapt the framework to user-specific problem sets (Parejo et al. 2012).

Theoretic work in algorithm portfolio design has provided two nontrivial insights (Huberman et al. 1997; Streeter and Smith 2008). First, combining some competent strategies into a portfolio may improve the overall performance by exploiting the negative correlation among their individual performance. Second, the performance can be further strengthened through low-level cooperative search among individual algorithms. Thus, any competent algorithm cases become precious knowledge to be accumulated to adapt to changes over time. For end users, it is much easier to understand the offline performance of individual algorithms rather than to understand the complex behavior of algorithmic components.

It is challenging to support cooperative algorithm portfolios in a development framework, though. Traditionally, heterogeneous algorithms might only loosely cooperate through a communication medium (Talbi 2002). Implementing low-level hybridization of heterogeneous algorithms would often require expert-level modification of framework code in forming meaningful algorithms.

In this paper, a cooperative group optimization (CGO) system is proposed to utilize the synergy between the cooperative group and algorithm portfolio design. This nature-inspired metaphor allows us not only inheriting the adaptability and productivity of a cooperative group, but also possessing the generality to accumulate various search heuristics for existing metaphors. Furthermore, each agent hold a portfolio of heterogeneous *embedded search heuristics* (ESHs), in which each ESH can drive the whole group into a stand-alone CGO case, and hybrid CGO cases can be defined by cooperative search among a set of competent ESHs that share customized memory elements. In addition, the optimization process might also be accelerated by a passive group leader through adaptively shaping the search landscape, if any global features are available.

Based on a concrete framework, CGO cases are defined by a script assembling over different instances of algorithmic components in a toolbox. A multilayer design of the script, with the support of the inherent updatable graph in the protocol among memory elements, enables a possible way to address the challenge of accumulating heterogeneous ESHs with a few algorithmic components, and building customized portfolios without writing any additional code. For end users, it is possible to easily define competent hybrid metaheuristics for specific problem sets using offline performance information of individual ESHs, based the insights from portfolio algorithm design.

The rest of this paper is organized as follows. In Sect. 2, a generic CGO system is presented in details. In Sect. 3, the CGO system is implemented for solving the constrained optimization problem (Deb 2000) with a few domain-specific components. In Sect. 4, based on a set of well-known benchmark instances (Liang et al. 2006; Runarsson and Yao 2005), the process of algorithm portfolio design is demonstrated, and the experimental results of customized CGO cases are compared to that of existing algorithms in literature. In Sect. 5, we discuss related work and possible extensions. This paper is concluded in the last section.

2 CGO system

The whole CGO system can be represented by a triple, i.e., $\langle Framework, Toolbox, Script \rangle$, as shown in Fig. 1. The toolbox contains some reusable algorithmic *components*. The multiagent framework realizes cooperative group optimization (CGO) algorithms, which is driven by the script with some *interfaces* for embedding valid *instances* of components in the toolbox. Figure 1 is used in the whole section while the details of the CGO system are gradually introduced.

The system is designed to accommodate three levels of usages. First, the CGO framework supports the basic concept of low-level portfolio algorithm design in a cooperative group. Second, algorithm designers might realize different algorithmic components in the toolbox with some basic interfaces. Finally, basic users can realize (hybrid) CGO algorithms using a multi-layer script, whereas the framework and any components in the toolbox are simply reusable black-box objects. The last two usages also enable the CGO system to be evolvable.

Basically, the CGO framework follows a modular (and autonomous) design. For a module contains multiple components, an connection to that module means the information can be accessed by all of its components (although might only be used by some of them). For example, $F_R^{(t)}$ is accessed by the components in the
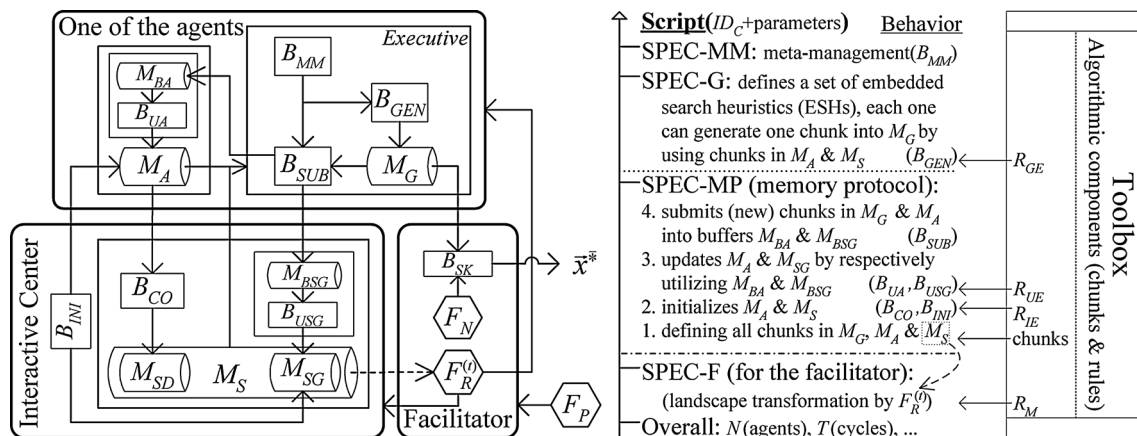

Fig. 1 The CGO system: the multiagent framework, the multilayer script and primary interfaces, and the toolbox

interactive center and all agents, and M_A and M_S are used by components in the executive module of each agent. There is a direct connection between two components across two modules if the usage is specific. For example, M_A of each agent is accessed by B_{CO} in the interactive center. For simplicity, some modules are anonymous.

This section is structured as follows. We first introduce basic type-based concepts and notations. Sections 2.2–2.4 then describe the framework, toolbox and script. Further details of memory and behavior in the CGO framework will be described in Sect. 2.5. In Sect. 2.6, the execution process of the CGO framework is described based on all these building blocks.

2.1 Preliminary concepts

The CGO system is full of knowledge elements that can be organized in a *type*-based representation. Each knowledge element can be accessed by using its *identifier*, referring to a *name* and a *type*, in which the type defines some *properties* for facilitating knowledge sharing, and the name ensures the uniqueness. For each type, a *compatible type* is a *subtype* or the same type.

The general problem-solving capability arises from the interaction of declarative and procedural knowledge (Anderson 2005). A basic declarative component, called a *chunk*, aggregates a small amount of problem information in a specific data structure. A procedural component contains *actions*, in which each works on some *input/output parameters*. It is called a *rule* if it has one action. Each component might have some *setting parameters*. For a *macro* component, one or more setting parameters are component types rather than primitive data types.

In the CGO toolbox, each algorithmic component is a binary object that can be instantiated using its *actual type* and valid setting parameters. Each parameter or script interface has a *formal type*, which is either a primitive data type or a component type for accepting an instance of any knowledge component if its actual type is a compatible type.

The CGO script is used for calling instances of algorithmic components of specific *interfaces* in the toolbox. *Primary interfaces* are directly supported types, whereas association interfaces might be introduced from components that are embedded as setting parameters of macro components.

2.1.1 Notation

Normally, a type is notated in the form of $TG_{TK}^{(t)}$ or $TG_{(TT)}^{(t)}$, in which TG indicates the *general type*; TK represents a *key variant*, e.g., a subtype or with a nontrivial property; TT in the subscript parentheses means a simple variant, which is often used as the names of similar instances; and t in the

superscript parentheses stresses it possesses the dynamic property in a time-varied style, where t means at the tth learning cycle.

Here are general types[1] to be used in this paper. Some types are related to the problem, where "F" means a problem representation and "S" means a space of states. As major building blocks for solving capability in the CGO framework, "M" is a *memory* containing some chunks, and "B" is a *behavior* with actions that directly or indirectly interact with some chunks in memory. "C" means a setting parameter of a component.

Some general types are used for chunks. "CH" is used to mention a chunk in general, but each specific chunk has a general type. A set of chunks of the same type "CH" can be organized into a *chunk set*, called "$\$CH$", in which "$\$$" means a set. "E" means a list of ordered chunks of arbitrary types.

Only the notation for rules is more complicated, since lots of rules, in which actual types can be subtypes of (subtypes of) some formal types, might be realized to make the system flexible and evolvable.

A rule is notated in the form of $R_{TK(TT)}^{TA(t)}$, in which "R" is the general type, TA stresses an actual type. If TA is not used, it is a formal type (an abstract rule) that is used for a parameter. If necessarily, a subtype of TK is notated as $TK{:}TKC$, where TKC after "$:$" indicates the unique properties associated with the subtype, and a further subtype can be notated in the same way.

2.2 CGO framework: overall description

The CGO framework supports the cooperative search of a group of totally N agents. All the agents are of the same structure. Figure 1 shows one of the agents and the shared environment that contains a *facilitator* and an *interactive center*. Each agent possesses a limited search capability and can only indirectly interact with its peers through the shared environment, in order to achieve the common goal of finding a near optimal solution $\mathbf{x}^*$ for the problem F_P.

The CGO framework runs in iterative learning cycles. The execution is terminated if the number of learning cycles (t) achieves the maximum cycle number (T).

2.2.1 Facilitator

For a global optimization problem F_P to be solved, an essential landscape (Paulus 2000; Satzinger et al. 1999) can be represented as a tuple $\langle S_P, R_M \rangle$. The *problem space*

---

[1] Note that the same symbol no longer means a general type if it appears at other places. Taking "M_S" as an example ("M" is its general type), "S" means a key variant rather than the general type of a space of states.

(S_P) contains all the *states* to be searched, in which each state $\mathbf{x}$ is a potential solution. For a real-world group, the states of F_P might be viewed as creative ideas (Paulus 2000). The *quality-measuring* rule (R_M) measures the difference of *quality* between $\forall \mathbf{x}_{(a)}, \mathbf{x}_{(b)} \in S_P$: if the quality of $\mathbf{x}_{(a)}$ is better than that of $\mathbf{x}_{(b)}$, then $R_M(\mathbf{x}_{(a)}, \mathbf{x}_{(b)})$ returns TRUE, otherwise it returns FALSE.

The facilitator maintains a *natural representation* (F_N) and an *internal representation* (F_R), which are both formulated from the problem F_P. For $F_N = \langle S_P, R_{M:N} \rangle$, the $R_{M:N}$ rule is an R_M rule possessing the *natural* property that faithfully measuring the quality among candidate states as the same as in the original problem F_P. It is used by the *solution-keeping* behavior (B_{SK}) to update the *best-so-far* state $\mathbf{x}^*$ of F_P among all states that are generated by agents. Specifically, B_{SK} replaces $\mathbf{x}^*$ by any state $\mathbf{x} \in S_P$ if $\mathbf{x}$ has a better quality, based on the $R_{M:N}$ rule. The optimal solution of F_P is ensured to be kept if it is visited.

For $F_R = \langle S_P, R_{M(R)}, AUX \rangle$, the $R_{M(R)}$ rule can be an arbitrary R_M rule, and AUX contains any auxiliary components associated with structural information of F_P that might be useful for search. The basic usage of F_R is to encapsulate any knowledge in F_P that will be further processed to find solutions. It is the internal problem used in the interactive center and all agents.

Both F_N and F_R are representations on S_P. However, F_N is not used for providing search clues, whereas F_R might deviates from original problem landscape during the run-time for facilitate the search process.

For a specific problem type, S_P, $R_{M:N}$, and AUX can be predefined, since S_P is defined in F_P, only one $R_{M:N}$ rule is required (since different $R_{M:N}$ rules are equivalent for the usages in F_N), and AUX is normally rather concise in practical usages (although it is suitable to put any available and useful knowledge into AUX). Thus the main effort is to implement an R_M rule as the input for $R_{M(R)}$. A simple way is to set $R_{M(R)} = R_{M:N}$, but more domain-specific $R_{M(R)}$ might be designed if landscape features are available. An example of the facilitator will be demonstrated in Sect. 3.3.

The facilitator might be viewed as a passive group leader, who can influence the solving process by providing and adaptively updating F_R, but not directly managing the operations of any agents. The problem landscape might be transformed by using an unnatural R_M rule that incorporates suitable knowledge. For example, approximate models (Jin et al. 2002) may smooth a rugged landscape, and constraint-handling techniques (Hamida and Schoenauer 2002; Runarsson and Yao 2005; Xie and Zhang 2004) have been widely used. An adaptive $R_{M(R)}^{(t)}$ rule might be realized by feeding a run-time chunk in M_S.

2.2.2 Agents and interactive center

The search process for solving F_R is performed by the agents with the support of the interactive center. The general solving capability arises from the interplay between *memory* and *behavior* (Anderson 2005).

In cognitive theories, memory (Anderson 2005; Ericsson and Kintsch 1995; Glenberg 1997) is a basic component for supporting the learning process. The importance of using (public) memory has been also addressed in some computational frameworks (Lau et al. 2012; Milano and Poli 2004; Taillard et al. 2001).

Specifically, memory is used for storing and retrieving chunks, in which each chunk contains certain particularities of state(s) in F_R. In this paper, a conceptualization model (Glenberg 1997) is used, which only requires a bounded space complexity, as compared to the unbounded memory used in some cognitive architectures (Anderson 2005). Each memory holds a list of permanent *cells*, in which each cell possesses a unique *cell type* and only stores the chunk of a compatible type as its content.

Two memory types, i.e., *long-term memory* (LTM) (Ericsson and Kintsch 1995) and *buffer*, are classified according to if all stored chunks are cleared at the end of each learning cycle or not. For a LTM, the chunk in each cell must be filled at the initialization stage and is subjected to be updated during the run-time. Note that each LTM cell only keeps the most recently updated chunk.

From the viewpoint of each agent, there are three basic memories, including a generative buffer (M_G) and an individual memory (M_A) of its own and a social memory (M_S) in the interactive center. For each agent, the two long-term memories, i.e., M_S and its M_A, store all currently available past experience for it, while the buffer M_G temporarily stores a chunk that is newly generated by it, in each learning cycle.

In a LTM, a chunk possesses either the *genuine* or *dependent* property according to if it only contains independent data or it is a specific data structure only designating the references to other chunks. In this paper, we only considered three kinds of LTM: M_A only contains genuine chunks, whereas two sub-memories M_{SG} and M_{SD} in M_S respectively possess genuine and dependent chunks. Here M_{SD} is used for sharing *non-private* chunks (Liu and Tsui 2006) in M_A of all agents.

All chunks in LTMs must be initialized and might be updated during learning cycles. The memories with genuine chunks, including M_A in all agents and M_{SG} in the interactive center, are initialized by using the *initializing behavior* (B_{INI}). These genuine chunks are updated in a similar way: chunks in M_A and M_{SG} are respectively updated by the M_A-*updating* (B_{UA}) and M_{SG}-*updating* (B_{USG}) behaviors through respectively using the chunks in

the buffers M_{BA} and M_{BSG}. The dependent chunks in M_{SD} are initialized by the *collecting behavior* (B_{CO}) for collecting non-private chunks in M_A of all agents, and are automatically updated if the referring chunks in M_A of any agents are changed.

For each agent, its executive module performs the *meta-managing* (B_{MM}), *generating* (B_{GEN}), and *submitting* (B_{SUB}) behaviors at each learning cycle. For a given CGO algorithm case, the B_{MM} behavior probabilistically selects one of its executive rows, in which the generative part is executed by the B_{GEN} behavior for outputting a new chunk into M_G by using the inputs in M_A and M_S, whereas the updating list is used by the B_{SUB} behavior for submitting chunks from M_A (cloned) and M_G into specific buffer cells of M_{BA} and M_{BSG}. Each chunk in M_G possesses the *solution* property that can export a potential solution $\mathbf{x}$. The potential solution is exported to the facilitator as a candidate for the best-so-far solution $\mathbf{x}^{\ast}$.

2.3 CGO toolbox

The toolbox contains some addable/removable algorithmic components of specific interfaces. Each component can be called symbolically using its identifier and setting parameters (thus the actual realization might be in a black box for end users). Primary interfaces are directly called by the CGO script (in Sect. 2.4), whereas background components of association interfaces might be used in components of primary interfaces, if necessarily. One nontrivial usage of the toolbox is to embed knowledge units that are commonly used in existing optimization algorithms, whereas novel algorithmic components might also be supported, if available.

The chunks in M_A, M_S, and M_G are of some primary chunk interfaces. In general, a chunk interface is notated as "*CH*". Since there are a group of agents, if any type *CH* is used in M_A and M_G, then $\$CH$ is automatically considered as a primary interface, in which "$\$$" means a set of chunks.

Most straightforward primary chunk interfaces include a state $\mathbf{x}$ and a state set $\$\mathbf{x}$. For example, $\$\mathbf{x}$ can be used as a population of individuals in many evolutionary algorithms. In stochastic local search strategies (Hoos and Stutzle 2004), $\mathbf{x}$ is used as an incumbent solution to be improved. There are some other types in existing algorithms. In ES (Runarsson and Yao 2005), the chunk to be generated is of a combined type ($\mathbf{x}, \boldsymbol{\sigma}$), in which $\boldsymbol{\sigma}$ is used for a log-normal distribution. In model-based algorithms, probabilistic models (e.g., a pheromone matrix; Socha and Dorigo 2008) are used, which can be seen as chunks in the public memory.

For the facilitator, only R_M is considered as a primary rule interface for realizing $R_{M(R)}$ in F_R. For constraint optimization, R_M can be used for embedding constraint-handling methods (Hamida and Schoenauer 2002;

Runarsson and Yao 2005; Xie and Zhang 2004). R_M might use one chunk in M_S as its input for run-time guidance.

For the agents and the interactive center, there are three primary rule interfaces, i.e., elemental *initializing* (R_{IE}), *updating* (R_{UE}), and *generating* (R_{GE}) rules.

Specifically, R_{IE} instances are used by B_{INI} for initializing M_A and M_{SG}, R_{UE} instances are used by B_{UA} and B_{USG} for updating M_A and M_{SG} by respectively using the buffers M_{BA} and M_{BSG}, and R_{GE} instances are used by B_{GEN} for generating new chunks into M_G.

All the components the facilitator can access F_N, whereas all the components in the agents and the interactive center can access F_R by default. Primary rule interfaces might have various subtypes. Some subtypes are problem-specific, whereas some subtypes are generic across different problem types (e.g., the constrained optimization problem, graph coloring, and traveling salesman problem), by using only generic knowledge in F_R. Only some generic subtypes are introduced here as examples, whereas some problem-specific subtypes will be introduced in Sect. 3, when we demonstrate the actual implementation on a specific problem type.

2.3.1 Background rules

In this paper, two *selecting* rules are used in other rules ($R_{UE:X}$ and R_{GE} rules respectively in Sects. 2.3.3 and 3.4). A *selecting* rule (R_{SEL}) chooses one state $\mathbf{x}_{(O)}$ from a state set $\$\mathbf{x}_{(I)}$.

The *greedy* R_{SEL} rule (R_{SEL}^G) has no setting parameter. It simple returns the best state among the states in $\$\mathbf{x}_{(I)}$ by comparing to each $\mathbf{x} \in \$\mathbf{x}_{(I)}$ one by one, using the $R_{M(R)}$ rule in F_R.

The *tournament* R_{SEL} rule (R_{SEL}^{TS}) has two setting parameters, i.e., a tournament size C_{NTS}, a Boolean quality flag C_{BQ}. It is executed as the follows. First, totally C_{NTS} states are selected from $\$\mathbf{x}_{(I)}$ at random. Second, these states are compared by using the $R_{M(R)}$ rule, and the state with a better quality or a worse quality is kept, if C_{BQ} is TRUE or FALSE, respectively. Finally, the last surviving state is outputted as the selected state $\mathbf{x}_{(O)}$.

2.3.2 Elemental initializing rule

The R_{IE} rule has an output chunk $CH_{(I)}$. Each actual R_{IE} rule is used for initializing each chunk in M_{SG} and each chunk set $\$CH_{(M)}$, in which $CH_{(M)}$ is a chunk in M_A of each agent.

The $R_{IE:X}$ subtype is an R_{IE} rule that outputs a state set $\$\mathbf{x}$ as $CH_{(I)}$. For the *random* $R_{IE:X}$ rule ($R_{IE:X}^{RND}$), each element in $\$\mathbf{x}$ is randomly generated by R_{GE}^{RND} within the problem space S_P.

2.3.3 Elemental updating rule

The R_{UE} rule has two input chunks, i.e., $(CH_{(M)}, CH_{(U)})$, and updates the chunk $CH_{(M)}$. Each R_{UE} rule is used for updating a genuine chunk $CH_{(M)}$ in M_A or M_{SG}. Here we only consider two basic subtypes.

The $R_{UE:S}(\mathbf{x}_{(a)}, \mathbf{x}_{(d)})$ rule replaces $\mathbf{x}_{(a)}$ by $\mathbf{x}_{(d)}$ in a specific condition. There are two simple $R_{UE:S}$ rules: (a) the *direct* $R_{UE:S}$ rule ($R_{UE:S}^D$), which replaces unconditionally; and (b) the *greedy* $R_{UE:S}$ rule ($R_{UE:S}^G$), which carries out the replacement if $R_{M(R)}(\mathbf{x}_{(a)}, \mathbf{x}_{(d)}) \equiv$ TRUE.

The $R_{UE:X}(\$\mathbf{x}_{(a)}, \$\mathbf{x}_{(d)})$ rule forms a new $\$\mathbf{x}_{(a)}$ by picking some of the states in $\$\mathbf{x}_{(a)} \cup \$\mathbf{x}_{(d)}$. Various subtypes of $R_{UE:X}$ are used in evolutionary algorithms for updating the population with new individuals. The *tournament-selection* $R_{UE:X}$ rule ($R_{UE:X}^{TS}$), which has one setting parameter called C_{NTW}, is realized as follows. For each state in $\$\mathbf{x}_{(b)}$, it replaces one state in $\$\mathbf{x}_{(a)}$ that is selected by an R_{SEL}^{TS} instance (defined in Sect. 2.3.1) with $C_{NTS} = C_{NTW}$ and $C_{BQ} =$ FALSE.

There are some other subtypes used in existing algorithms. For example, in ACO (Socha and Dorigo 2008), $CH_{(M)}$ is a pheromone matrix and $CH_{(U)}$ is a state set $\$\mathbf{x}$.

2.3.4 Elemental generating rule

The R_{GE} rule has an ordered list of input chunks $E_{(IG)}$, and an output chunk $CH_{(OG)}$ that has the solution property. Each actual R_{GE} rule performs the search role for generating a new chunk.

Inputs/outputs of R_{GE} rules might be arbitrarily defined, although most of them are defined in simple forms. Using a list of input chunks in $E_{(IG)}$ allows flexible cooperative search between R_{GE} rules by sharing some chunks. Using a single chunk in $CH_{(OG)}$ enables a simple realization, but without loss of generality.

In many existing algorithms, their search operators can be seen as R_{GE} rules that output $\mathbf{x}$, although they might use different element(s) in $E_{(IG)}$. An extreme case is for a search rule start from scratch, in which $E_{(IG)} = \varnothing$. For example, a random R_{GE} rule (R_{GE}^{RND}) is a generic rule that generates a random state within S_P. Some of them only use one element in $E_{(IG)}$. For example, a local search heuristic R_{LS} uses an incumbent state, whereas each ant in ACO (Socha and Dorigo 2008) uses a pheromone matrix. Some of them, e.g., PSO, DE, and SCO (as will be introduced in Sect. 3.4), use multiple elements in $E_{(IG)}$. Some search operators do use other elements rather than $\mathbf{x}$ as $CH_{(OG)}$. For example, the chunk to be generated in ES (Runarsson and Yao 2005) is of a macro type $(\mathbf{x}, \boldsymbol{\sigma})$ for encoding a log-normal distribution around $\mathbf{x}$.

Moreover, R_{GE} can be realized in macro forms that support some association interfaces. For example, for an R_{GE} rule that has $E_{(IG)} = \{\$\mathbf{x}\}$ and $CH_{(OG)} = \mathbf{x}$, a possible relay form (Talbi 2002) is a tuple $\langle R_{SEL}, R_{XS} \rangle$ (Xie and Liu 2010), in which R_{XS} is a *recombination* rule that outputs $\mathbf{x}$ by using two parents $\mathbf{x}_{(1)}$ and $\mathbf{x}_{(2)}$ independently selected from $\$\mathbf{x}$ by the R_{SEL} rule. Furthermore, a mutation rule or a local search rule can be linked for perturbing or improving the output state $\mathbf{x}$ (Ong et al. 2006; Talbi 2002).

2.4 CGO script

As shown in Fig. 1, the CGO framework is driven by a script realized in multiple layers. The script body includes overall setting parameters, problem specification (SPEC-F), memory protocol specification (SPEC-MP), generative specification (SPEC-G), and meta-management specification (SPEC-MM). For all the interfaces in the script, instances of components in the toolbox are used. Only a few setting parameters on component instances might be defined as script parameters if they are explicitly assigned. In the practical usage, a CGO algorithm case can be defined by a case identifier (ID_C) and a few script parameters, based on a given script body that reusing a few existing specifications.

The lowest two layers are quite simple. The overall script parameters include the number of agents (N) and the maximum number of learning cycles (T). In the facilitator, the problem specification (SPEC-F) is

$$\text{SPEC-F} = \langle R_{M(R)} \rangle,$$

since the other elements, i.e., S_P, $R_{M:N}$, and AUX, can be easily predefined for a specific problem type.

The upper three layers are used for driving all modules in the agents and the interactive center. Each layer contains addable/removable elemental rows for supporting an evolvable property in an algorithmic space.

2.4.1 Memory protocol specification

The memory protocol specification (SPEC-MP) defines how will the chunks be initialized and updated in M_A of all agents and M_S of the interactive center, given any chunk is newly generated in M_G of the agents. SPEC-MP might be seen as a domain ontology for encoding low-level knowledge units (Edgington et al. 2004). In Fig. 1, it is used for driving the modules in all agents and the interactive center, except for the *meta-managing* (B_{MM}) and *generating* (B_{GEN}) behaviors in the agents.

Formally, SPEC-MP contains a table of *memory protocol rows*, in which each contains five elements, i.e.,

$$\text{SPEC-MP Row} = \langle ID_M, CH_M, R_{IE}, R_{UE}, CH_U \rangle,$$

where $ID_M \in \{M_A, M_{SG}, M_{SD}\}, CH_M$ at each row is a unique chunk in the memory referred by ID_M. In SPEC-MP, the second column defines the three lists of chunks in M_A, M_{SG}, and M_{SD}. The list of chunks in the last column contains all chunks in M_A and M_G. The chunks in the columns of CH_M and CH_U belong to primary chunk interfaces. Each chunk in M_G possesses the *solution* property that can export a state **x**.

The last three elements are defined differently for genuine and dependent chunks. If $CH_M \notin M_{SD}, R_{IE}$ and R_{UE} are elemental *initializing* and *updating* rules for CH_M, and $CH_U \in M_A \cup M_G$ is a candidate chunk for updating CH_M. If $CH_M \in M_{SD}, R_{IE}$ and R_{UE} are *null*, and $CH_U \in M_A$, since each chunk in M_{SD} is automatically updated using chunks in M_A of all agents.

The validity of SPEC-MP can be locally checked in two steps. The first step is to ensure that the types used in each row are locally compatible. Notice the fact that there are multiple agents but only one interactive center. If $CH_M \in M_{SD}$, then its type is $\$CH_U$, in which each element is CH_U in M_A of all agents. Table 1 gives the types of input/output parameters of R_{IE} and R_{UE} for the chunks in M_A and M_{SG}. There are two special cases of using a chunk set. If $CH_M \in M_A, R_{IE}$ is used for initializing CH_M in M_A of all agents. If $CH_M \in M_{SG}, M_{BSG}$ will collect all chunks that are submitted from agents in each learning cycle as the inputs of R_{UE} for updating CH_M.

The second step is to ensure the validity across all rows. First, each CH_M must be unique, and each $CH_U \in M_G$ must possess the solution property. Second, each chunk CH_M should be *updatable*, i.e., has the probability to be updated by chunks that are generated in M_G, across multiple cycles. Notice that if a row is used in a cycle, CH_M is updated by CH_U.

The validity of updatable relations can be easily checked by using an *updatable graph* that is formed from all rows in SPEC-MP: For each row, CH_U is the parent node of CH_M. A valid updatable graph contains separate trees, where for each tree, the root is a chunk in M_G, the children are chunks in M_A and M_S, and each chunk in M_S is always a leaf node. An example of a valid updatable graph, which contains a single updatable tree, is provided later in Table 2.

Furthermore, SPEC-MP can be easily maintained by using the updatable graph. Except for the root nodes, each other node in the updatable graph is CH_M in a unique memory protocol row. Each leaf node (i.e., the

Table 1 Generic types of input/output parameters of R_{IE} and R_{UE} rules

ID_M	$CH_{(I)}$ of R_{IE}	$CH_{(M)}$ of R_{UE}	$CH_{(U)}$ of R_{UE}
M_A	$\$CH_M$	CH_M	CH_U
M_{SG}	CH_M	CH_M	$\$CH_U$

corresponding row), if it is not used by the upper layer, can be removed without changing the validity of the remaining graph. A whole tree is removed from the graph if only its root node is left. Each new node can be added to either a leaf node or a root node.

SPEC-MP provides an essential support for the stability in cooperative search. If $CH_M^{(t)}$ is used by different search heuristics, it is always updated by R_{UE} and $CH_U^{(t)}$ in the same row.

2.4.2 Generative specification

The generative specification (SPEC-G) contains a set of *generative rows*, in which each generative row is

SPEC-G Row $= \langle ID_G, R_{GE}, E_{IG}, CH_{OG} \rangle$,

where ID_G is a unique name, R_{GE} is an elemental generating rule, E_{IG} is an ordered list of chunks, CH_{OG} is a chunk. Each ID_G designates an ESH that is corresponding to a stand-alone algorithm case.

The validity of SPEC-G only needs to be locally ensured in each ESH, based on the memory protocol specification (SPEC-MP). First, each chunk in E_{IG} belongs to $M_A \cup M_S$, and $CH_{OG} \in M_G$. Second, E_{IG} and CH_{OG} are respectively linked to the input/output parameters of the R_{GE} rule. Third, all chunks in $E_{IG} \cup CH_{OG}$ must be updatable, i.e., these chunks form a *subtree* that contains one root node (i.e., a chunk in M_G) in the updatable graph defined in Sect. 2.4.1.

Each generative row is independent to the other. New ESHs can be freely added into SPEC-G, or be removed from SPEC-G if it is not used in any portfolio.

2.4.3 Meta-management specification

The meta-management specification (SPEC-MM) defines a customized portfolio within an algorithm space that is formed by using some ESHs as the bases.

Each algorithm case has a name (ID_C) and contains a set of *executive rows*, in which each executive row contains three elements, i.e.,

SPEC-MM Row $= \langle ID_G, E_{UPD}, C_W \rangle$,

in which ID_G is the name of an ESH in SPEC-G, the updating list E_{UPD} contains a list of genuine chunks, and $C_W \geq 0$ is a weight value for the row.

The selection probability for each executive row is $C_W / \sum C_W$. Thus any executive row is ignored if it has $C_W = 0$. Any two executive rows are called *cooperative* rows if their E_{UPD} lists share at least one element, otherwise they are *independent* to each others.

For ensuring the validity of each CGO algorithm case, E_{UPD} in each row must satisfy $E_{IGG} \subseteq E_{UPD} \subseteq E_{IGM}$, in

which E_{IGG} contains the list of all genuine chunks in E_{IG} of the corresponding ESH, and $E_{IGM} = \bigcup E_{IGG}$ for E_{IGG} of all ESHs. The intuition is that each chunk that is used by ESHs must be actively updated. By default, there are $C_W = 1$ and $E_{UPD} = E_{IGG}$ for each (independent) ESH that is newly added as an executive row in SPEC-MM.

If ESHs use input chunks in the same tree of the updatable graph in SPEC-MP, their E_{UPD} lists might be customized in E_{IGM}. An algorithm case is called as in the *customized* or *default* mode according to any E_{UPD} is customized or not. The customization of E_{UPD} lists can turn independent ESHs into cooperative ESHs, and can make cooperative ESHs cooperating more.

2.5 CGO framework: memory and behavior

As shown in Fig. 1, the memory and behavior in the agents and the interactive center are driven by the script using some components in the toolbox.

2.5.1 Long-term memory and buffer modules

As described in Sect. 2.4.1, the genuine chunks in M_A and M_{SG}, and dependent chunks in M_{SD} are defined in the first two columns of SPEC-MP, and the root nodes in the updatable graph of SPEC-MM form the list of chunks that are supported in M_G. Each chunk in M_A and M_S can be retrieved by each agent. Any new solution contained in M_G is submitted to the facilitator.

The buffers M_{BA} and M_{BSG} are respectively used for updating M_A and M_{SG}, where their cells are of one-to-one mapping based on each row of SPEC-MP, i.e., each cell in a buffer accepts CH_U if it is used for updating a chunk CH_M in M_A or M_{SG}. As shown in the last column of Table 1, the corresponding *cell types* of M_{BA} and M_{BSG} are respectively CH_U and $\$CH_U$, since each agent only submit once to its M_{BA}, whereas all agents might submit chunks to cells in M_{BSG}.

2.5.2 Initializing and updating behavior

The initializing behavior (B_{INI}) is used for initializing the genuine chunks in LTMs during the initialization stage ($t = 0$). For each row in SPEC-MP, B_{INI} executes the R_{IE} instance to obtain $\$CH_M$ for $CH_M \in M_A$ of all agents and $CH_M \in M_{SG}$ of the interactive center (as the output types shown in Table 1).

Duraing the runtime ($t > 0$), the chunks in M_A and M_{SG} are respectively updated by the M_A-*updating* behavior (B_{UA}) and M_{SG}-*updating* behavior (B_{USG}).

For each row of SPEC-MP, the input/output parameters ($CH_{(M)}$, $CH_{(U)}$) of R_{UE} are linked to the corresponding cells in M_A and M_{BA} if $CH_M \in M_A$, or in M_{SG} and M_{BSG} if

$CH_M \in M_{SG}$. Then in each cycle, each B_{UA} or B_{USG} executes the corresponding R_{UE} instance if the corresponding buffer cell is not empty.

2.5.3 Collecting behavior

The collecting behavior (B_{CO}) is used for managing the dependent chunks in M_{SD} of the interactive center. At $t = 0$, B_{CO} forms each dependent chunk $CH_M = \{CH_{U(i)} | i \in [1, N]\}$ into M_{SD}, where $CH_{U(i)}$ is from M_A of the ith agent. During the runtime, dependent chunks in M_{SD} are automatically updated if the reference chunks in M_A of any agents are updated.

2.5.4 Submitting behavior

The *submitting* behavior (B_{SUB}) submits chunks into M_{BA} and M_{BSG}, given M_A, M_G, and a updating list E_{UPD} of genuine chunk identifiers. For each chunk identifier $CH_M \in E_{UPD}$, the corresponding row in SPEC-MP is found, and then a cloned chunk of $CH_U \in M_A \cup M_G$ is submitted into the corresponding buffer cell in M_{BA} or M_{BSG} if $CH_M \in M_A$ or $CH_M \in M_{SG}$.

2.5.5 Generating behavior

The *generating* behavior (B_{GEN}) generates a chunk with the solution property into M_G. Based on a given ID_G, the corresponding ESH in SPEC-G is chosen. Then B_{GEN} executes the R_{GE} instance and generates a chunk CH_{OG} into M_G by using the input chunk list $E_{IG} \in M_A \cup M_S$.

2.5.6 Meta-managing behavior

The *meta-managing* behavior (B_{MM}) picks the algorithm case with a given ID_C from SPEC-MM. Afterward, one of the executive rows in the CGO case is probabilistically selected, according to the associated C_W values. Afterward, ID_G and E_{UPD} in the selected executive row are used as the inputs for consecutively executing the B_{GEN} and B_{SUB} behaviors.

2.6 CGO framework: execution process

Algorithm 1 gives the execution process of the CGO framework. In each line, the working module (entity), the required inputs, and the outputs or updated modules are provided.

In Line 1, F_P is formulated into F_N and F_R by forming the elements $\langle S_P, R_{M:N}, R_{M(R)}, AUX \rangle$. In Lines 2 and 3, all long-term memories used by the agents and the interactive center are initialized by using B_{INI} and B_{CO}. After the

initialization, the CGO framework runs in iterative learning cycles, in which each learning cycle $t \in [1, T]$ is executed between Lines 5–16.

In Line 5, an option is provided for the facilitator for updating $F_R^{(t)}$ by using a chunk in M_S. In Lines 7–10, each agent $i \in [1, N]$ is executed. In Line 7, $B_{MM(i)}$ is executed to select an executive row, which contains ID_G and E_{UPD}, in SPEC-MM. In Line 8, the embedded search heuristic (ESH), which is named ID_G in SPEC-G, is triggered to generate its output chunk $CH_{OG} \in M_{G(i)}$ by using the list of input chunks $E_{IG} \in M_{A(i)} \cup M_S$. In Line 9, the buffer cells in $M_{BA(i)}$ and M_{BSG}, which are corresponding to the updating list E_{UPD} in LTMs, are filled by $B_{SUB(i)}$ by using SPEC-MP. In Line 10, the solution contained in $CH_{OG} \in M_{G(i)}$ is processed by B_{SK} to obtained the best-so-far solution $\mathbf{x}^{\circledast}$, based on the quality evaluation by using F_N. During Lines 6–11, all LTMs remain unchanged. In Line 12–15, M_{SG} and each $M_{A(i)}$ are independently updated by B_{USG} and each $B_{UA(i)}$. Line 16 mentions the fact that M_{SG} is automatically updated if the corresponding chunks in M_A of agents are updated. Finally, $\mathbf{x}^{\circledast}$ is returned while the framework is terminated.

performs lifetime learning by only using its individual memory (M_A); (2) a *stigmergic/evolutionary group*, in which each agent does not possess its M_A, but agents can indirect cooperate with their peers through the social memory (M_S); and (3) a *cooperative group*, in which each agent performs a mix of the individual and social learning by using both M_A and M_S. Both nominal and cooperative groups have been used for studying group creativity (Goncalo and Staw 2006; Paulus 2000).

The cooperative group is an advanced algorithm designed by natural evolution over millions of years. The paradigm helps striking a natural balance between exploitation and exploration in the problem landscape. In a cooperative group, the agents explore in a parallel way with their individual memory, as well as cooperate with their peers through the group memory.

For each agent, its individual memory (Ericsson and Kintsch 1995; Glenberg 1997), i.e., M_A, supports its lifetime learning (Curran and O'Riordan 2006), e.g., "trial-and-error", for discovering novel knowledge based on experience. A sequence of the chunks (or thoughts; Ericsson and Kintsch 1995) updated in the same cell can be

---

Algorithm 1 The execution process of the CGO framework, given a script and a toolbox

Require: The optimization problem F_P; A CGO case (ID_C + script parameters: N, T, $\cdots$)
1: Facilitator: $\langle \text{SPEC-F}, F_P \rangle \to \langle F_N, F_R \rangle$ $\{F_R$ is used by all agents and the interactive center$\}$
2: B_{INI}: SPEC-MP $\to \langle \{M_{A(i)} | i \in [1, N]\}, M_{SG} \rangle$ $\{R_{IE}$ instances, Section 2.5.2$\}$
3: B_{CO}: SPEC-MP $\to M_{SD}$ $\{$Section 2.5.2$\}$
4: **for** $t = 1$ to T **do**
5:      Facilitator: $\langle F_R^{(t)}, M_S \rangle \to F_R^{(t+1)}$ $\{$[Optional] landscape transformation$\}$
6:      **for** $i = 1$ to N **do**
7:          $B_{MM(i)}$: $\langle ID_C, \text{SPEC-MM} \rangle \to \langle ID_G, E_{UPD} \rangle$ $\{$Section 2.5.6$\}$
8:          $B_{GEN(i)}$: $\langle ID_G, \text{SPEC-G}, M_{A(i)}, M_S \rangle \to CH_{OG} \in M_{G(i)}$ $\{R_{GE}$ instances, Section 2.5.5$\}$
9:          $B_{SUB(i)}$: $\langle E_{UPD}, \text{SPEC-MP}, M_{A(i)}, CH_{OG} \rangle \to \langle M_{BA(i)}, M_{BSG} \rangle$ $\{$Section 2.5.4$\}$
10:          B_{SK}: $\langle CH_{OG}, F_N \rangle \to \mathbf{x}^{\circledast}$ $\{$Stores the best-so-far solution$\}$
11:      **end for**
12:      B_{USG}: $\langle \text{SPEC-MP}, M_{BSG}, M_{SG}^{(t)} \rangle \to M_{SG}^{(t+1)}$ $\{R_{UE}$ instances, Section 2.5.3$\}$
13:      **for** $i = 1$ to N **do**
14:          $B_{UA(i)}$: $\langle \text{SPEC-MP}, M_{BA(i)}, M_{A(i)}^{(t)} \rangle \to M_{A(i)}^{(t+1)}$ $\{R_{UE}$ instances, Section 2.5.3$\}$
15:      **end for**
16:      /* $M_{SD}^{(t)} \to M_{SD}^{(t+1)}$ is automatically updated by the dependence built by B_{CO} */
17: **end for**
18: **return** $\mathbf{x}^{\circledast}$ $\{$Returns the best-so-far solution$\}$

2.7 System characteristics

The CGO system has three characteristics: (1) the CGO framework can support a cooperative group; (2) each agent holds a customized portfolio of ESHs; and (3) The framework is driven by a multilayer script working on knowledge components in the CGO toolbox.

2.7.1 Cooperative group

In principle, the CGO framework can support three kinds of groups: (1) a *nominal group*, in which each agent

regarded as a "trajectory" (Glenberg 1997) along with learning cycles. An algorithmic example of individual learning strategies is stochastic local search (Hoos and Stutzle 2004). In a group, individual learning is essential for social learning to be useful (Laland 2004), by escaping from some maladaptive outcomes (Boyd et al. 2011).

For a group of agents, M_S might be referred as public memory (Danchin et al. 2004) or group memory (Dennis and Valacich 1993; Satzinger et al. 1999). M_{SD} contains non-private chunks that can be observed from M_A of the agents, whereas M_{SG} contains all genuine chunks that are not possessed by any agents, e.g., pheromone trails in an ant colony (Socha and Dorigo 2008) and the group memory

in *brainstorming* (Dennis and Valacich 1993). Many animals and human beings are able to learn socially by observing their peers and/or utilizing external knowledge (Danchin et al. 2004; Laland 2004).

Thus, each agent possesses a mixed cultural learning capability (Boyd et al. 2011; Curran and O'Riordan 2006; Galef 1995; Tomasello et al. 1993) that operates with both the individual and social memories. The social memory contains gradually accumulated and recombined adaptive knowledge (Boyd et al. 2011) for accelerating the learning process, whereas the individual memory preserve some promising minority patterns (Nemeth 1986) for supporting the capability of escaping from some maladaptive outcomes (Boyd et al. 2011), which is essential for the social memory to be useful (Laland 2004). The emergence of solutions in the group level might also share some essences with collective intelligence (Woolley et al. 2010).

In each cycle, the agents might be different in not only the chunks in their M_A but also the executive rows picked by their B_{MM}. Even a nominal group becomes a portfolio of heterogeneous algorithms (Huberman et al. 1997; Streeter and Smith 2008), which may achieve better overall performance by exploiting the large variance among the performance of the agents.

Allowing for cooperation among the agents may improve search performance (Huberman et al. 1997) by enabling agents to circumvent their own cognitive limitations. Compared to a nominal group, the interaction may enhance the group creativity (Paulus 2000), as shown in *brainstorming* (Dennis and Valacich 1993; Kohn and Smith 2011).

Compared to a stigmergic group, a cooperative group has two major features due to the possession of personal memories by the agents. First, the agents may explore in a parallel way, while the diversity of positive patterns, even those in minority (Nemeth 1986), can be preserved in a more reliable way. Individualism in a group may foster the group creativity by encouraging uniqueness (Goncalo and Staw 2006). Second, the cooperative mechanism in a group needs not to be designed very carefully since the public information does not have an overwhelming impact on accumulated knowledge in the system.

For the viewpoint of *population-based algorithms*, stigmergic groups, e.g., ES (Runarsson and Yao 2005), GA (Deb 2000), MA (Ong et al. 2006), ACO (Socha and Dorigo 2008), and CA (Reynolds et al. 2008), are commonly studied, where M_S might contains different chunks, e.g., an evolutionary population (Ong et al. 2006; Runarsson and Yao 2005, a pheromone matrix (Socha and Dorigo 2008), and external belief (Reynolds et al. 2008). A nominal group can be seen as a portfolio (Huberman et al. 1997; Streeter and Smith 2008) of independent local search agents (Hoos and Stutzle 2004).

Independent local search agents can only use blind disturbance (Hoos and Stutzle 2004), whereas cooperative agents can be guided by adaptive clues in the social memory, when they are trying to escape from some local valleys in the problem landscape. In a stigmergic group such as GA, the population diversity must be explicitly maintained by frequently applying blind disturbance, e.g., mutation operators, whereas a cooperative group keeps novel and diverse states in individual memory of the agents.

2.7.2 Algorithm portfolios

In the CGO system, each agent holds a portfolio (Huberman et al. 1997) of embedded search heuristics (ESHs). The meta-management behavior can be viewed as a task-switching schedule (Streeter and Smith 2008) to interleave the execution of a portfolio (Huberman et al. 1997) of ESHs across learning cycles. Any executive rows using the nodes of the same tree in the updatable graph may be cooperative by default, or they can be turned into cooperative rows by using the customized mode, if necessarily. The cooperation among ESHs may further improve performance (Huberman et al. 1997). The cooperative algorithm portfolio is strengthened in the cooperate group, since novel cooperation results can be easily diffused though the group memory, and detrimental results might be isolated in individual memory of agents.

From a user-oriented perspective, the practical problem sets are different during different periods for different users. According to the No Free Lunch (NFL) theorems (Wolpert and Macready 1997), it is impossible to obtain an omnipotent algorithm case for a sufficiently diverse set of problem instances. Thus, it is rational to tackle the problem set faced by each user during a sufficiently long period, by using a portfolio of fast-and-frugal heuristics. New heuristics, which mainly tackle some of new problems unsolved by existing heuristics, can be implemented into the portfolio over time.

According to its $E_{(IG)}$, each ESH might possess one of the four search properties: (a) *scratch search*, which has $E_{(IG)} = \varnothing$; (b) *individual learning*, which only uses the chunks in M_A; (c) *social learning*, which only uses the chunks in M_{SG}; and (d) *cultural learning* (or *socially-biased individual learning*; Galef 1995), which employs the input chunks in both M_A and M_S. In principle, the agents in a cooperative group might use mixed strategies, as long as cultural learning strategies play a nontrivial role. For example, in GSO (He et al. 2009), only scroungers use a cultural learning strategy, whereas producers and rangers employ individual learning strategies.

There is a basic paradigm shift in supporting the algorithm space. Traditional methods mainly use the

space of setting parameters for tuning/controlling the algorithm performance (Eiben et al. 1999). Each algorithm with setting parameters might support a huge algorithm space, but only a few algorithm cases are competent for some problem instances. The number of useful algorithm cases might be dropped to much less if the overlap among the performance of different algorithm cases is considered. The CGO algorithm space is mainly defined upon a portfolio of basis ESHs, in which each ESH is competent, which captures explicit/implicit domain-specific features, for some problem instances. Furthermore, CGO cases can be defined in an independent or cooperative way to stretch for solving most problem instances. Moreover, allowing heterogeneous inputs/outputs increases the chance of finding competent ESHs. The total portfolio size can be maintained to be small by adding new ESHs that are competent for new problems as well as removing obsolete ESHs.

2.7.3 Multilayer script

The CGO system is a development framework that is driven by an multilayer script. This is essential for supporting the vision of the adaptive box, since implementing many stand-alone search heuristics might require quite an effort, let alone flexibly supporting the cooperative search among heterogeneous search heuristics that are sharing customized memory elements.

In the upper three layers of the CGO script, each layer contains some elemental rows. A new row can be added into a layer to provide more choices for implementing new rows into higher layers; and an old row can be removed from a layer if it is not used by any higher layers. Thus, any competent knowledge components can be easily accumulated, whereas obsolete knowledge component can be easily removed, without leading to any risk to interfere existing algorithm cases.

Each layer might provide nontrivial knowledge for the higher layer. SPEC-MP forms a simple memory protocol ontology for the interaction between the agents and the interactive center. The corresponding updatable graph is not only useful for checking the validity of SPEC-MP, but also for defining feasible input/output chunks for each ESH. SPEC-MP also provides a nontrivial support for the stability of cooperative search among different ESHs. The actual performance of each ESH in SPEC-G provides essential knowledge for designing promising portfolios (Huberman et al. 1997; Streeter and Smith 2008) in SPEC-MM.

The implementation process may mainly occur in higher levels, once there are enough supports from lower layers. Eventually, almost all operations will take place in

SPEC-MM for finding suitable portfolios, either cooperative or not, if a sufficiently large number of ESHs are implemented.

3 Implementation for constrained optimization

For each problem F_P of a specific type, the CGO system can be concretely implemented. Here the constrained optimization problem is used for demonstrating the implementation process. We first introduce the problem, then describe problem-specific algorithmic components. Here we provide full details of these components for easily reproducing the algorithms, but it might be worthy to keep in mind that each component is a binary object with specific input/output parameters for end users.

3.1 The constrained optimization problem

THe F_P of the constrained optimization problem can be defined as follows (Deb 2000):

$$\begin{cases} \text{Minimize}: & f(\mathbf{x}) \\ \text{Subject to}: & g_j(\mathbf{x}) \in [\underline{c}_j, \bar{c}_j]_{\mathbb{R}} \quad (j \in [1, J]_{\mathbb{Z}}) \end{cases} \quad (1)$$

in which $\mathbf{x} = (x_{[1]}, \ldots, x_{[D]}) \in S_P \subset \mathbb{R}^D$ is a state within the space S_P which is a D-dimensional Euclidean space with the *boundary constraints* $x_{[d]} \in [\underline{x}_{[d]}, \bar{x}_{[d]}]_{\mathbb{R}}$ for $\forall d \in [1, D]_{\mathbb{Z}}$, $f(\mathbf{x})$ is the *objective function*, and each $g_j(\mathbf{x})$ is a *constraint function* with two constant boundary values $\underline{c}_j$ and $\bar{c}_j (\underline{c}_j \leq \bar{c}_j)$. The feasible space S_{PF} is defined as $S_{PF} = \{\mathbf{x} | g_j(\mathbf{x}) \in [\underline{c}_j, \bar{c}_j]_{\mathbb{R}}, \forall j \in [1, J]_{\mathbb{Z}}; \mathbf{x} \in S_P\}$. Any solutions in S_{PO} are located in S_{PF}, by treating all constraints as hard constraints. We define $\bar{c}_{\min} = \min_{j=1}^{J}(\bar{c}_j - \underline{c}_j)$ for convenience. If $\underline{c}_j \equiv \bar{c}_j$, the jth constraint is called an *equality constraint*, which is preprocessed into a relaxed form by using $g_j(\mathbf{x}) \in [\underline{c}_j - \varepsilon_H, \bar{c}_j + \varepsilon_H]$ with a tolerance parameter $\epsilon_H > 0$.

3.2 Quality measurement

The quality measurement is a macro subtype, i.e., $R_M = \langle R_{QE}^{V2}, R_{QC} \rangle$, which is realized as

$$R_M(\mathbf{x}_{(a)}, \mathbf{x}_{(b)}) = R_{QC}\left(R_{QE}^{V2}(\mathbf{x}_{(a)}, R_{QE}^{V2}(\mathbf{x}_{(b)})\right) \quad (2)$$

in which the *quality-encoding* (R_{QE}^{V2}) rule is a background rule that calculates the intermediate quality of the only input state into a commonly-used data structure (v_{CON}, v_{OBJ}), then the *quality-comparing* rule (R_{QC}) evaluates any two (v_{CON}, v_{OBJ}) instances and returns TRUE or FALSE.

For $R_{QE}^{V2}(\mathbf{x})$, the output element v_{OBJ} is equal to the objective function value $f(\mathbf{x})$, and the output element v_{CON} is the summarized constraint violation value, i.e.,

$$v_{CON} = \sum_{j=1}^{J} \begin{cases} 0 & \text{IF } g_j(\mathbf{x}) \in [\underline{c}_j, \bar{c}_j]_{\mathbb{R}} \\ \underline{c}_j - g_j(\mathbf{x}) & \text{IF } g_j(\mathbf{x}) < \underline{c}_j \\ g_j(\mathbf{x}) - \bar{c}_j & \text{IF } g_j(\mathbf{x}) > \bar{c}_j \end{cases} \quad (3)$$

Thus, the minimum value of v_{CON} is 0. For $\forall\, \mathbf{x}$, if there is $v_{CON} \equiv 0$, then it means $\mathbf{x} \in S_{PF}$.

3.2.1 Existing quality comparison rules

A basic usage is that various existing constraint-handling techniques may be realized by using different R_{QC} rules.

A R_{QC} rule (R_{QC}^O) returns TRUE, if there is: (a) $v_{CON(a)} < v_{CON(b)}$; or (b) $v_{CON(a)} \equiv v_{CON(b)}$ and $v_{OBJ(a)} \leq v_{OBJ(b)}$. The R_{QC}^O rule satisfies the following criteria (Deb 2000): (a) $\mathbf{x}_{(a)} \in S_{PF}$ is preferred to $\mathbf{x}_{(b)} \notin S_{PF}$; (b) between two states within S_{PF}, the one having a smaller objective function value is preferred; (c) between two states out of S_{PF}, the one having a smaller constraint violation is preferred. It has been widely used in some existing work (Lu and Chen 2008; Mezura-Montes and Coello 2005; Zhang and Xie 2003).

The *penalized* R_{QC} rule (R_{QC}^P) returns TRUE, if there is $v_{all(a)} \leq v_{all(b)}$, in which $v_{all} = v_{OBJ} + C_{AP} \cdot v_{CON}$ and $C_{AP} \geq 0$ is a static penalty coefficient. The static penalty term has been used as a popular technique (Deb 2000). However, deciding a good penalty coefficient for each specific problem instance might be a rather difficult optimization problem itself.

The *stochastic* R_{QC} rule (R_{QC}^S) returns TRUE, if there is: (a) $v_{CON(a)} < v_{CON(b)}$; or (b) $v_{OBJ(a)} \leq v_{OBJ(b)}$ as $v_{CON(a)} \equiv v_{CON(b)}$ or $U_{\mathbb{R}} < C_{PF}$, in which $C_{PF} \in [0,1]_{\mathbb{R}}$ is a setting parameter. This rule is used in the stochastic ranking technique (Runarsson and Yao 2005).

The *static-relaxing* R_{QC} rule (R_{QC}^{RS}) returns TRUE, if there is: (a) $v_{CON(a)} < v_{CON(b)}$ as $v_{CON(b)} > C_{ER}$; or (b) $v_{OBJ(a)} \leq v_{OBJ(b)}$ as both $v_{CON(a)}, v_{CON(b)} \leq C_{ER}$. Here $C_{ER} \geq 0$ is a relaxing value.

A *dynamic-relaxing* R_{QC} rule is then defined as a tuple $\langle R_{QC}^{RS}, R_{ADJ} \rangle$, where the *adjusting* rule (R_{ADJ}) dynamically adjusts the C_{ER} value of R_{QC}^{RS} (Hamida and Schoenauer 2002; Xie and Zhang 2004).

Among these R_{QC} rules, only R_{QC}^O leads to a natural landscape. However, the S_{PF} of a natural landscape is critically shaped by the boundary values of constraint functions. If the $\bar{c}_{\min}$ value of F_P is not large enough, S_{PF} may be long and narrow valleys, which can be divided into multiple segments of the ridge function class (Beyer 2001)

of unknown directions. Searching in such valleys is very challenging since improvement intervals (Salomon 1996) toward better solutions are predominantly too small.

For a landscape with the R_{QC}^{RS} rule, its quasi-feasible space is $S_{PF}' = \{\mathbf{x}|v_{CON}(\mathbf{x}) \leq C_{ER}; \mathbf{x} \in S_P\} \supseteq S_{PF}$. A good attribute is that S_{PF}' always shrinks when C_{ER} decreases, till $S_{PF}' \rightarrow S_{PF}$ for $C_{ER} \rightarrow 0$. To increase improvement intervals and to approach the optimum from both feasible and infeasible space, it is rational to sufficiently relax S_{PF}' at the early stage and gradually shrinks S_{PF}' to S_{PF}, by adjusting the C_{ER} value of R_{QC}^{RS} from large to small, during the run-time. Some dynamic adjustment techniques have been proposed (Hamida and Schoenauer 2002; Xie and Zhang 2004). The basic experience is that the adjustment pace is a key issue for the performance. In the next section (Sect. 3.2.2), we will consider a slightly modified adjusting rule for maintaining a controllable adjustment pace.

Some of these R_{QC} rules, including $R_{QC}^O, R_{QC}^P, R_{QC}^S$, and the dynamic adjustment method in (Hamida and Schoenauer 2002) have been used in existing algorithms discussed in Sect. 4.2.1.

3.2.2 Adaptive ratio-reaching adjustment

The *ratio-reaching* R_{ADJ} rule (R_{ADJ}^{RR}) adjusts $C_{ER}^{(t)}$ by using the $\bar{c}_{\min}$ value of F_P and the maximum number of cycles (T). Furthermore, it has four setting parameters: C_{RRE}, C_{RNU}, C_{RTU}, and an input state set called $\$\mathbf{x}_{FB}$. For convenience, we define $t_{TH} = \text{INT}(C_{RTU} \cdot T)$, in which $C_{RTU} \in [0,1]_{\mathbb{R}}$ and the function $\text{INT}(t)$ returns the closest integer value of its input t.

There are $C_{ER}^{(0)} = 0$ and $C_{ER}^{(1)}$ is set as the maximum v_{CON} value (Xie and Zhang 2004) of all the states in $\$\mathbf{x}_{FB}$. As $t > t_{TH}$, there is $C_{ER}^{(t)} = 0$ (Xie and Zhang 2004). Thus there are totally ($T - t_{TH}$) learning cycles left for fulfilling the final search process in S_{PF}. For $1 \leq t \leq t_{TH}$, if $c_{RNC}^{(t)} > C_{RNU}$, there is

$$C_{ER}^{(t+1)} = C_{ER}^{(t)} \cdot \left(c_{ERE}/C_{ER}^{(t)}\right)^{1/(t_{TH}-t+1)} \quad (4)$$

in which $C_{RNU} \in [0,1]_{\mathbb{R}}, c_{RNC}^{(t)}$ is the ratio of the states within the current S_{PF}' over all the states in $\$\mathbf{x}_{FB}$ (Hamida and Schoenauer 2002), $c_{ERE} = C_{RRE} \cdot \bar{c}_{\min}/2$ is an expected value of $C_{ER}^{(t_{TH})}$, and C_{RRE} is a positive constant.

Compared to the previous methods (Hamida and Schoenauer 2002; Xie and Zhang 2004), the minor modification in Eq. 4 aims in keeping a stable $c_{RNC}^{(t)}$ value, while maintaining an adaptive pace for adjusting $C_{ER}^{(t)}$ to the expected c_{ERE} value at $t = t_{TH}$.

3.2.3 Incorporate global knowledge

The equality constraints can be seen as a typical problem feature that is known in advance. The R_{QC}^{O3R} rule is a macro R_{QC} rule integrating R_{QC}^{O} and $R_{QC}^{RRR} = \langle R_{QC}^{RS}, R_{ADJ}^{RR} \rangle$ by a simple policy, i.e., R_{QC}^{RRR} is executed if $\underline{c}_{\min} \leq 2 \cdot \varepsilon_H$, otherwise R_{QC}^{O} is executed. Thus problem instances with and without equality constraints are tackled by the R_{QC}^{RRR} and R_{QC}^{O} rules, respectively.

3.3 The facilitator

Based on F_P, the facilitator is implemented from a tuple, i.e., $\langle S_P, R_{M:N}, R_{M(R)}, AUX \rangle$, where the $R_{M:N}$ rule in F_N and the $R_{M(R)}$ rule in F_R are respectively realized as $\langle R_{QE}^{V2}, R_{QC}^{O} \rangle$ and $\langle R_{QE}^{V2}, R_{QC(R)} \rangle$ (based on Eq. 2), and AUX only contains the D-dimensional Euclidean space S_P defined by the boundary constraints.

Here AUX is represented in a concise form. Following a practical setting of black-box optimization, no function details and gradient information are considered. The detail information of function values for candidate states and the boundary values for all constraint functions are encapsulated in the $R_{M(R)}$ rule.

For the facilitator, only $R_{QC(R)}$ can be assigned, but a high flexibility is still retained since $R_{QC(R)}$ can be realized in various forms (e.g., different R_{QC} rules defined in Sects. 3.2.1 and 3.2.3).

3.4 Elemental generating rules

Three R_{GE} rules are extracted from existing algorithms. In addition, boundary-handling methods are integrated into R_{GE} rules according to available knowledge.

3.4.1 Differential evolution R_{GE} rule

The *differential-evolution R_{GE} rule* (R_{GE}^{DE}) is extracted from differential evolution (DE) (Price et al. 2005). Its $E_{(IG)}$ has two chunks, i.e., $\{\mathbf{x}_P, \$\mathbf{x}_{DP}\}$. Its setting parameters include C_F, C_{CR}, and C_{CG}, in which $C_{CR} \in [0, 1]_{\mathbb{R}}$ is the crossover constant, $C_F > 0$ is the scale constant.

The R_{GE}^{DE} rule generates one state $\mathbf{x}_C$ as its $CH_{(OG)}$ by using the following steps:

(a) Create a list of states, i.e., $\{\mathbf{x}_{(a)}, \mathbf{x}_{(b)}, \mathbf{x}_{(c)}, \mathbf{x}_{(d)}\}$, by selecting from $\$\mathbf{x}_{DP}$ at random;
(b) Obtain c_{DR}, which is randomly chosen from $[1, D]$;
(c) For the dth dimension, if $U_{\mathbb{R}} < C_{CR}$ or $d \equiv c_{DR}$,

$$
\begin{aligned}
x_{C[d]} = x_{(P)[d]} &+ C_{CG} \cdot \left(x_{(g)[d]} - x_{(P)[d]}\right) \\
&+ C_F \cdot \left(x_{(a)[d]} - x_{(b)[d]} + x_{(c)[d]} - x_{(d)[d]}\right)
\end{aligned} \tag{5}
$$

in which $\mathbf{x}_{(g)} = R_{SEL}^{G}(\$\mathbf{x}_{DP})$ (defined in Sect. 2.3.1) is the state with the best quality in $\$\mathbf{x}_{DP}$, and $C_{CG} \in [0, 1]$ is the ratio between $x_{P[d]}$ and $x_{(g)[d]}$. Here $\mathbf{x}_C$ is respectively generated around $\mathbf{x}_{(g)}$ and $\mathbf{x}_P$, if C_{CG} is assigned as 1 and 0.

Its boundary-handling method is realized in a simple way: for the dth dimension, $x_{C[d]}$ is randomly chosen from $[\underline{x}_{[d]}, \bar{x}_{[d]}]_{\mathbb{R}}$ if there is $x_{C[d]} \notin [\underline{x}_{[d]}, \bar{x}_{[d]}]_{\mathbb{R}}$.

3.4.2 Particle swarm R_{GE} rule

The *particle swarm R_{GE} rule* (R_{GE}^{PS}) is extracted from the operation of each particle in PSO (Kennedy et al. 2001). Its $E_{(IG)}$ possesses four input chunks, i.e., $\{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{DP}\}$, and its $CH_{(OG)}$ is $\mathbf{x}_C$. Furthermore, its setting parameters include $C_A > 2$ and $C_B > 2$.

For the dth dimension, $\mathbf{x}_C$ is generated as

$$
\begin{aligned}
x_{C[d]} = x_{R[d]} &+ c_K \cdot \text{DIS}(x_{R[d]}, x_{O[d]}, d) \\
&+ C_A \cdot U_{\mathbb{R}} \cdot \text{DIS}\left(x_{P[d]}, x_{R[d]}, d\right) \\
&+ C_B \cdot U_{\mathbb{R}} \cdot \text{DIS}\left(x_{(g)[d]}, x_{R[d]}, d\right)
\end{aligned} \tag{6}
$$

in which $c_K = 2/(\sqrt{\varphi \cdot (\varphi - 4)} + \varphi - 2), \varphi = C_A + C_B > 4, \mathbf{x}_{(g)}$ is the state with the best quality in $\$\mathbf{x}_{DP}, U_{\mathbb{R}}$ is a real value randomly selected in $[0, 1]_{\mathbb{R}}$, and $\text{DIS}(x_{(a)}, x_{(b)}, d)$ calculates the distance between $x_{(a)}$ and $x_{(b)}$ at the dth dimension of S_P (Xie and Liu 2005), i.e.,

$$
\text{DIS}(x_{(a)}, x_{(b)}, d) = \begin{cases} \bar{\bar{x}}_{[d]} + y & \text{IF} \quad y < -\bar{\bar{x}}_{[d]}/2 \\ \bar{\bar{x}}_{[d]} - y & \text{IF} \quad y > \bar{\bar{x}}_{[d]}/2 \\ y & \text{OTHERWISE} \end{cases} \tag{7}
$$

in which $\bar{\bar{x}}_{[d]} = \bar{x}_{[d]} - \underline{x}_{[d]}$ and $y = x_{(a)} - x_{(b)}$.

Finally, $\mathbf{x}_{C[d]}$ is repaired for $\forall \mathbf{x}_C \notin S_P$ (Xie and Liu 2005), i.e.,

$$
x_{C[d]} = \begin{cases} \bar{x}_{[d]} - (\underline{x}_{[d]} - x_{C[d]})\%\bar{\bar{x}}_{[d]} & \text{IF} \, x_{C[d]} < \underline{x}_{[d]} \\ \underline{x}_{[d]} + (x_{C[d]} - \bar{x}_{[d]})\%\bar{\bar{x}}_{[d]} & \text{IF} \, x_{C[d]} > \bar{x}_{[d]} \end{cases} \tag{8}
$$

3.4.3 Social cognitive R_{GE} rule

The *social cognitive R_{GE} rule* (R_{GE}^{SC}) is extracted from the operation of an agent in social cognitive optimization (SCO) (Xie et al. 2002). Its $E_{(IG)}$ possesses two inputs chunks, i.e., $\{\mathbf{x}_R, \$\mathbf{x}_{GR}\}$, and one output chunk, i.e., $\mathbf{x}_C$. Furthermore, R_{GE}^{SC} has only one setting parameter of the integer type, i.e., $C_{NTB} > 0$. The basic idea is to learn from a good model state in a public knowledge pool.

The R_{GE}^{SC} rule produces $\mathbf{x}_C$ using the following steps:

(a) Select a model state $\mathbf{x}_{(m)}$ from $\$\mathbf{x}_{GR}$, by using the R_{SEL}^{TS} rule (as defined in Sect. 2.3.1) with the setting parameter values $C_{NTS} = C_{NTB}$ and $C_{BQ} = $ TRUE;

(b) Determine two states $\mathbf{x}_{(b)}$ and $\mathbf{x}_{(r)}$ from $\mathbf{x}_R$ and $\mathbf{x}_{(m)}$: If $R_{M(R)}$ $(\mathbf{x}_{(m)}, \mathbf{x}_R) \equiv$ TRUE, then $\mathbf{x}_{(b)} = \mathbf{x}_{(m)}$ and $\mathbf{x}_{(r)} = \mathbf{x}_R$, otherwise $\mathbf{x}_{(b)} = \mathbf{x}_R$ and $\mathbf{x}_{(r)} = \mathbf{x}_{(m)}$;

(c) Obtain the virtual promising space, called S_{pv}, which takes $\mathbf{x}_{(b)}$ as the center, and uses $\mathbf{x}_{(r)}$ to determine its range. For the dth dimension of $\mathbf{x} \in S_{pv}$, and $x_{(br)[d]} = |x_{(b)[d]} - x_{(r)[d]}|$, there is,

$$x_{[d]} \in \left[x_{(b)[d]} - x_{(br)[d]}, \, x_{(b)[d]} + x_{(br)[d]} \right]_{\mathbb{R}}; \quad (9)$$

(d) Generate $\mathbf{x}_C$ within $S_{pv} \cap S_P$ at random.

3.5 Implementation of CGO script

The CGO script is implemented over algorithmic components that are defined in previous sections: some generic components are defined in Sect. 2.3, whereas problem-specific components are defined in Sects. 3.2 and 3.4. Each setting parameter of a component instance is fixed, unless it is specially assigned as an overall script parameter. For simplicity, the script is shown in tables, but it can be easily converted into a standard format, e.g., extensible markup language (XML).

For the facilitator (realized in Sect. 3.3), SPEC-F is simply defined by assigning an R_{QC}^{O3R} instance (realized in Sect. 3.2.3) as $R_{QC(R)}$ in its $R_{M(R)}$. For the R_{ADJ}^{RR} instance in R_{QC}^{RRR}, its parameters include $C_{RRE} = 10$, $C_{RNU} = 0.5$, $C_{RTU} = 0.5$, and $\$\mathbf{x}_{FB} = \$\mathbf{x}_{DP} \in M_S$ (defined later in SPEC-MP).

Table 2 lists the elemental rows in SPEC-MP, for $M_A = \{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P\}$, $M_{SG} = \{\$\mathbf{x}_{GR}\}$, $M_{SD} = \{\$\mathbf{x}_{DP}\}$, and $M_G = \{\mathbf{x}_C\}$. Here the primary chunk interfaces include $\mathbf{x}$ and $\$\mathbf{x}$. For $|\$\mathbf{x}_{GR}|$, the number of states is $C_{NGR} = 4 \cdot N$. All genuine chunks are initialized by $R_{IE:X}^{RND}$. The dependent

chunk is $\$\mathbf{x}_{DP} = \{\mathbf{x}_{P(i)} | i \in [1, N]\}$, in which $\mathbf{x}_{P(i)}$ is in M_A of the ith agent. For illustrative purposes only, the corresponding updatable graph, which contains a single tree, is also shown in Table 2.

As shown in Table 3, SPEC-G contains four embedded search heuristics (ESHs), in which each ESH has a ID_G, an R_{GE} instance with setting parameter values, and the lists of input/output parameters E_{IG} and CH_{OG}. Both G.DE1 and G.DE2 use R_{GE}^{DE} instances, while G.PS and G.SC uses R_{GE}^{PS} and R_{GE}^{SC} instances, respectively. Besides, G.SC uses one chunk in M_{SG}, while the others uses one chunk in M_{SD}. For each ESH, its E_{IG} contains the nodes in a sub-graph of the updatable graph in Table 2. All the four ESHs use both M_A and M_S elements in their E_{IG}. For illustrative purposes only, the lists of all active genuine chunks, i.e., E_{IGG}, is also shown in Table 3.

Table 4 lists seven CGO cases. All of them are defined on four executive rows, in which each executive row is associated with a generative row in SPEC-G, according to its ID_G. Columns 2–5 in Table 4 include the updating lists in $E_{IGM} = \{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{GR}\}$. All E_{SUB} lists are defined in the default mode, and the elements in each E_{SUB} (i.e., the corresponding E_{IGG} elements shown in Table 3) are marked by "$\sqrt{}$". In the last seven columns, the seven CGO cases with different ID_C, i.e., #PS, #DE1, #DE2, #SC, #DEDE, #DEPS, and #DESC, are defined by assigning with different C_{SV} values for the four executive rows, in which each executive row is not actually used if the corresponding C_{SV} value is 0. Thus, the first four cases only use a single executive row, whereas the last three cases use two executive rows in the equal probability.

Table 2 The memory protocol rows in SPEC-MP and the corresponding updatable graph

ID_M	CH_M	R_{IE} instance	R_{UE} instance	CH_U	Updatable graph
M_A	$\mathbf{x}_O$	$R_{IE:X}^{RND}$	$R_{UE:S}^{D}$	$\mathbf{x}_R$	
M_A	$\mathbf{x}_R$	$R_{IE:X}^{RND}$	$R_{UE:S}^{D}$	$\mathbf{x}_C$	
M_A	$\mathbf{x}_P$	$R_{IE:X}^{RND}$	$R_{UE:S}^{G}$	$\mathbf{x}_C$	
M_{SG}	$\$\mathbf{x}_{GR} : C_{NGR} = 4 \cdot N$	$R_{IE:X}^{RND}$	$R_{UE:X}^{TS} : C_{NTW} = 4$	$\mathbf{x}_R$	
M_{SD}	$\$\mathbf{x}_{DP}$	$-$	$-$	$\mathbf{x}_P$	

Table 3 The generative rows in SPEC-G and the corresponding E_{IGG} lists

ID_G	R_{GE} instance	E_{IG}	CH_{OG}	E_{IGG}
G.PS	$R_{GE}^{PS} : C_A = C_B = 2.05$	$\{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{DP}\}$	$\mathbf{x}_C$	$\{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P\}$
G.DE1	$R_{GE}^{DE} : C_F = 0.5, C_{CR} = 0.1, C_{CG} = 1.0$	$\{\mathbf{x}_P, \$\mathbf{x}_{DP}\}$	$\mathbf{x}_C$	$\{\mathbf{x}_P\}$
G.DE2	$R_{GE}^{DE} : C_F = 0.5, C_{CR} = 0.9, C_{CG} = 1.0$	$\{\mathbf{x}_P, \$\mathbf{x}_{DP}\}$	$\mathbf{x}_C$	$\{\mathbf{x}_P\}$
G.SC	$R_{GE}^{SC} : C_{NTB} = 2$	$\{\mathbf{x}_R, \$\mathbf{x}_{GR}\}$	$\mathbf{x}_C$	$\{\mathbf{x}_R, \$\mathbf{x}_{GR}\}$

Table 4 Seven CGO cases supported by SPEC-MM, where $E_{UPD} \in E_{IGM} = \bigcup E_{IGG} = \{\mathbf{x}_O, \mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{GR}\}$, and each CGO case has its name ID_C (e.g., #PS) and its set of C_W values in the corresponding column

ID_G	$\mathbf{x}_O$	$\mathbf{x}_R$	$\mathbf{x}_P$	$\$\mathbf{x}_{GR}$	#PS	#DE1	#DE2	#SC	#DEDE	#DEPS	#DESC
G.PS	$\surd$	$\surd$	$\surd$		1	0	0	0	0	1	0
G.DE1			$\surd$		0	1	0	0	1	0	0
G.DE2			$\surd$		0	0	1	0	1	1	1
G.SC		$\surd$		$\surd$	0	0	0	1	0	0	1

Table 5 The executive rows of SPEC-MM for #DESC-I, an customized CGO case ("$\surd$" indicates additional elements)

ID_G	$\mathbf{x}_O$	$\mathbf{x}_R$	$\mathbf{x}_P$	$\$\mathbf{x}_{GR}$	#DESC-I
G.PS	$\surd$	$\surd$	$\surd$		0
G.DE1			$\surd$		0
G.DE2		$\underline{\surd}$	$\surd$		1
G.SC		$\surd$	$\underline{\surd}$	$\surd$	1

Table 5 describes the #DESC-I case, in which only both G.DE2 and G.SC are actually used, and the E_{SUB} lists are specified in the customized mode. Compared to #DESC, two additional elements, i.e., $\mathbf{x}_R$ for G.DE2 and $\mathbf{x}_P$ for G.SC, are marked by "$\underline{\surd}$". The two executive rows are independent in #DESC, but cooperative in #DESC-I. Thus the customized mode provides an additional dimension for further designing new algorithms with additional cooperative rows. Such a cooperation may facilitate for the search process of other embedded search heuristics (ESHs).

Each CGO case is designated by ID_C and a few setting parameters. In the current script, there are only two parameters, i.e., the number of agents (N) and the maximum number of cycles (T).

3.5.1 Informal execution process

Here an informal description is provided for the actual execution of the #DESC-I case in Table 5. The formal description of the executionc an be found in Sect. 2.

For an agent i, both rows G.DE2 and G.SC in SPEC-MM (Table 5) have the same probability to be selected by $B_{MM(i)}$. Here we assume that G.SC is selected in the current cycle. Thus there are $ID_G =$ G.SC, and $E_{UPD} = \{\mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{GR}\}$. With ID_G, B_{GEN} locates the fourth row in SPEC-G (Table 3), and executes the R_{GE}^{SC} instance with $E_{IG} = \{\mathbf{x}_R, \$\mathbf{x}_{GR}\}$ from $M_{A(i)}$ and M_S and $CH_{OG} = \mathbf{x}_C$ to $M_{G(i)}$. Afterward, the elements in E_{UPD} are processed by B_{SUB}, based on the corresponding rows in SPEC-MP (Table 2). For example, $CH_U = \mathbf{x}_C$ is sent to the cell in the buffer $M_{BA(i)}$ that is used for update $\mathbf{x}_{P(i)} \in M_{A(i)}$, and $CH_U = \mathbf{x}_R$ is sent to the cell in the buffer M_{BSG} that is used

for update $\$\mathbf{x}_{GR} \in M_{SG}$. Note that M_{BSG} might receive chunks from different agents. At the end of the cycle, each buffer cell with new content will be used to update the corresponding LTM cell by the corresponding R_{UE} rule defined in SPEC-MP (Table 2).

At next cycles, G.DE2 and G.SC in SPEC-MM might be interactively selected. As we can see, G.SC only use two elements, i.e., $E_{IG} = \{\mathbf{x}_R, \$\mathbf{x}_{GR}\}$, but it updates three elements, i.e., $E_{UPD} = \{\mathbf{x}_R, \mathbf{x}_P, \$\mathbf{x}_{GR}\}$. The extra update is marked by "$\surd$" in Table 5, and it will only have an impact when $\mathbf{x}_P$ serves as an input chunk element for G.DE2.

3.6 Discussion

We have described the solid implementation of various CGO cases. Given the CGO framework, the implementation process including two parts, i.e., realize algorithmic components in the toolbox, and organize the instances of these components by the script. There are only a few components in the toolbox. In Table 2, we have two generic chunk types, i.e., $\mathbf{x}$ and $\$\mathbf{x}$, a few generic rules, i.e., $R_{IE:X}^{RND}, R_{UE:S}^{D}, R_{UE:S}^{G}$, and $R_{UE:X}^{TS}$. These generic components might be used across different problem types. In Table 3, we have three problem-specific rules, i.e., R_{GE}^{PS}, R_{GE}^{DE}, and R_{GE}^{SC}.

The memory protocol ontology is defined on a few chunks. Based on the position and corresponding updating process in the updatable graph, nontrivial properties may emerge for these chunks. In the individual memory of each agent, $\mathbf{x}_O$, $\mathbf{x}_R$, and $\mathbf{x}_P$ are the best state found so far, the most recently found state, and the state found in the last cycle, respectively. In the group memory, $\$\mathbf{x}_{DP}$ is the collection of elite states found by the agents, and $\$\mathbf{x}_{GR}$ is a steady-state set.

The primary advantage is to define different apparent stand-alone algorithms in an efficient way. For the ESHs, G.PS is a PSO instance, G.DE1 and G.DE2 are two DE instances, and G.SC is a SCO instance. Different instances of a single rule (e.g., R_{GE}^{DE}) might be included in SPEC-E. To develop new algorithms, the main effort is put on realizing R_{GE} with heterogeneous E_{IG} and CH_{OG}. Some

possible R_{GE} subtypes that include common algorithmic operators (e.g., recombination, mutation, and local search) are discussed in Sect. 2.3.4. For example, Step c of R_{GE}^{SC} can be viewed as a recombination operator. Furthermore, as discussed in Sect. 2.7.2, some ESHs might not necessarily be cultural learning strategies (i.e., R_{GE} uses inputs in M_A and M_S). For example, a single-start (e.g., local search and mutation) search rule and the R_{GE}^{RND} rule might be respectively added to Table 3 using $E_{IG}=\{\mathbf{x}_P\}$ and $E_{IG} = \varnothing$, given $CH_{OG} = \mathbf{x}_C$. We might also change the realization of an ESH by changing the chunks used in its E_{IG} and CH_{OG}. For example, G.SC might turn into a totally different algorithm if it uses the E_{IG} of G.DE2.

Moreover, hybrid CGO cases can be formed in a combinatorial algorithmic space formed by using a portfolio of user-oriented ESHs as basis, without writing any additional code. Here #DEDE, #DEPS, and #DESC, are defined in the default mode, and #DESC-I is defined in the customized mode. We have only considered the customized mode in an expanded style that adds additional elements into the E_{UPD} lists. The customized mode might be more flexible, as long as it follows the basic principle that any elements in E_{IGM} can be probabilistically updated. The ability of supporting a large algorithmic space offering algorithm designers quick turnaround in realizing hybrid CGO cases. The portfolio design might be guided using the offline performance of individual ESHs accumulated over time.

4 Experimental results

The experiments are performed on thirteen widely-used benchmark instances ($G01$–$G13$) (Runarsson and Yao 2005) originating from real-world applications. Table 6 summarizes the diverse characteristics of the benchmark instances (Runarsson and Yao 2005). Four of the instances, i.e., $G03$, $G05$, $G11$, and $G13$, have equality constraints. By default, the tolerance value for each equality constraint is $\epsilon_H = 1E{-}4$.

The algorithm performance is measured on the mean results under given numbers of function evaluations (NFE). For each CGO case, its NFE is approximately equal to $N \cdot T$, since the evaluation times in $t = 0$ can be neglected if T is large enough. For each problem instance, 500 independent runs were performed for obtaining the mean results. Furthermore, only the runs that entered S_{PF} are taken into accounted, and the number of runs that did not enter S_{PF} is reported in parentheses.

For an algorithm case, a problem instance is regarded as *solved* if the difference between the mean result and the optimal value is smaller than 1E−5 (except for $G08$ and $G13$, which are 1E−6). All the solved results listed in following tables are emphasized in **boldface**. As

Table 6 Summary of main characteristics of the benchmark instances (Runarsson and Yao 2005): [linear inequality (LI), nonlinear equality (NE), nonlinear inequality (NI), and the number of active constraints at optimum (NA)]

F_P	D	$f(x)$ type	$|S_{PF}|/|S_P|$ (%)	LI	NE	NI	NA
$G01$	13	Quadratic	0.011	9	0	0	6
$G02$	20	Nonlinear	99.990	1	0	1	1
$G03$	10	Polynomial	0.000	0	1	0	1
$G04$	5	Quadratic	52.123	0	0	6	2
$G05$	4	Cubic	0.000	2	3	0	3
$G06$	2	Cubic	0.006	0	0	2	2
$G07$	10	Quadratic	0.000	3	0	5	6
$G08$	2	Nonlinear	0.856	0	0	2	0
$G09$	7	Polynomial	0.512	0	0	4	2
$G10$	8	Linear	0.001	3	0	3	3
$G11$	2	Quadratic	0.000	0	1	0	1
$G12$	3	Quadratic	4.779	0	0	9^3	0
$G13$	5	Exponential	0.000	0	3	0	3

comparing sub-optimal results with different algorithms (shown in Tables 10, 11, and 15), each existing result is simply <u>underlined</u> if it has no statistically significant difference from the corresponding #DESC-I result at 95 % confidence level, based on Welch's t test.

4.1 Algorithm selection

The selection process is based on the two insights in algorithm portfolio design (Huberman et al. 1997). For all tests in Sect. 4.1, there are $N = 60$ and $T = 2{,}000$, and NFE is 1.2E4.

Table 7 summarizes the mean results by four pure CGO cases, i.e., #DE1, #DE2, #PS, and #SC, in which each case uses a single ESH. Here some nontrivial knowledge about the competency of the corresponding ESHs may be obtained from their offline performance.

For the instances without equality constraints, the f^{*1} values are the true optimal solutions. For those instances with equality constraints, the f^{*1} values are the optimal solutions obtained as $\epsilon_H = 1E{-}4$.

As shown in Table 7, #DE1, #DE2, #PS, and #SC consistently achieved the optimal solutions in six, eleven, five, and five of the instances, respectively. #DE2 achieved the best search performance among the four CGO cases. For the two instances $G01$ and $G02$, DE1 and #SC were able to achieve very good results, whereas #DE2 and #PS did not obtain good enough results.

Table 8 gives the mean results by #DEDE, #DEPS, #DESC, and #DESC-I, in which each hybrid case employs two ESHs. Here #DEPS is included since it represents an existing algorithm called DEPSO (Zhang

Table 7 Results by four pure CGO cases using a single ESH

F_P	f^{*1}	#DE1	#DE2	#PS	#SC
$G01$	−15.00000	**−15.00000**	−14.78906	−14.90595	**−15.00000**
$G02$	−0.80362	−0.80091	−0.62628	−0.64812	−0.79764
$G03$	*−1.00050*	−0.99493	**−1.00050**	−1.00045	N/A
$G04$	−30,665.5387	**−30,665.5387**	**−30,665.5387**	**−30,665.5387**	**−30,665.5387**
$G05$	*5,126.49671*	(500)	**5,126.49671**	5,137.3522(8)	N/A
$G06$	−6,961.81388	**−6,961.81388**	**−6,961.81388**	**−6,961.81388**	**−6,961.81388**
$G07$	24.30621	24.79860	**24.30621**	25.14686	24.41236
$G08$	−0.095825	**−0.095825**	**−0.095825**	**−0.095825**	**−0.095825**
$G09$	680.63006	681.03268	**680.63006**	680.65376	680.64244
$G10$	7,049.24802	7,211.44153	**7,049.24802**	7,456.49008	7,166.74353
$G11$	*0.74990*	**0.74990**	**0.74990**	**0.74990**	N/A
$G12$	−1.00000	**−1.00000**	**−1.00000**	**−1.00000**	**−1.00000**
$G13$	*0.053942*	(500)	**0.053942**	0.073435	N/A

Relaxed optimal values are italicized (which are obtained in the relaxed setting epsilon = 1E−4)

Table 8 Results by four hybrid CGO cases using two ESHs

F_P	#DEDE	#DEPS	#DESC	#DESC-I	St. dev.
$G01$	−14.99531	−14.96719	**−15.00000**	−14.99997	2.205E−05
$G02$	−0.79712	−0.69590	−0.79287	−0.79006	1.255E−02
$G03$	**−1.00050**	**−1.00050**	**−1.00050**	**−1.00050**	1.489E−10
$G04$	**−30,665.5387**	**−30,665.5387**	**−30,665.5387**	**−30,665.5387**	2.942E−10
$G05$	**5,126.49671**	**5,126.49671**	**5,126.49671**	**5,126.49671**	9.346E−12
$G06$	**−6,961.81388**	**−6,961.81388**	**−6,961.81388**	**−6,961.81388**	3.277E−11
$G07$	**24.30621**	**24.30621**	**24.30621**	**24.30621**	3.305E−07
$G08$	**−0.095825**	**−0.095825**	**−0.095825**	**−0.095825**	5.835E−16
$G09$	**680.63006**	**680.63006**	**680.63006**	**680.63006**	2.855E−12
$G10$	7,049.24822	7,049.24812	7,049.24812	7,049.24813	1.370E−04
$G11$	**0.74990**	**0.74990**	**0.74990**	**0.74990**	6.001E−15
$G12$	**−1.00000**	**−1.00000**	**−1.00000**	**−1.00000**	0.000E−00
$G13$	**0.053942**	**0.053942**	**0.053942**	**0.053942**	2.114E−16

and Xie 2003). For #DESC-I, the standard deviation (*St. Dev.*) is provided.

All the four hybrid cases achieved the optimal solutions for ten instances, which may mean that they inherited most of the merit from G.DE2. Moreover, they all achieved better results than #DE2 for both $G01$ and $G02$. The portfolio may benefit from the negative correlation among the performance of individual algorithms (Huberman et al. 1997). Besides, both DESC and DESC-I performed better results than #DEDE in $G01$ and $G10$, and #DEPS in $G01$ and $G02$, respectively.

For the four instances with equality constraints, Table 9 gives the optimal solutions (f^{*2}) and the mean results obtained by three CGO cases, i.e., #DE, #DESC, and #DESC-I, as the allowed tolerance value is reduced to $\epsilon_H = 1E−8$. All the three CGO cases found the optimal solutions for $G03$ and $G11$. Furthermore, #DE2 and #DESC-I respectively found the optimal solutions for $G05$

and $G13$. For $G05$, the result of #DESC-I was slightly worse than the optimal solution, but it achieved a much better result than #DE2.

For an instance with equality constraints, a smaller ϵ_H leads to a more accurate optimal solution. Compared to the real optimal solutions as $\epsilon_H = 0$, the f^{*2} results are the same, whereas the f^{*1} results still have the differences that are not negligible, under the given arithmetic precisions. It is meaningful that #DESC-I was able to achieve near-optimal solutions for all the four instances, even as the using of a smaller ϵ_H might significantly increase the problem difficulty.

For the instances with equality constraints, F_R is adjusted dynamically by using a dependent chunk, i.e., $\$x_{DP}$. In #DESC, only one executive row, i.e., G.DE2, may update x_P, which is a root chunk of $\$x_{DP}$, in the M_A of each agent. Thus the progress of the other executive row, i.e.,

Table 9 Results for instances with equality constraints as $\epsilon_H = 1\text{E}{-}8$

F_P	f^{*2}	#DE2	#DESC	#DESC-I	St. dev.
G03	-1.00000	**-1.00000**	**-1.00000**	**-1.00000**	1.748E$-$05
G05	5,126.49811	**5,126.49811**	5,126.50203	5,126.49812	8.143E$-$05
G11	0.75000	**0.75000**	**0.75000**	**0.75000**	4.334E$-$15
G13	0.053950	0.054720	0.055489	**0.053950**	2.799E$-$09

G.SC, is totally ignored. Actually, #DESC did not find better results than #DE2. However, in #DESC-I, G.SC updates $\mathbf{x}_P$, while G.DE2 updates $\mathbf{x}_R$ as well. Such a mutual interaction ensures that the search progress of both the executive rows are taken into account. As shown in Table 9, #DESC-I was able to achieve a much better result than #DE2 in G13.

The difference between #DESC and #DESC-I is in that the two ESHs in #DESC are independent, whereas in #DESC-I they are cooperative due to the additional updating elements. Compared #DESC-I to #DESC, such an interaction significantly enhanced the performance for G05 and G13, as shown in Table 9. Thus, for an algorithm portfolio, the overall performance might be further tuned through low-level cooperative search among individual algorithms (Huberman et al. 1997).

4.2 Comparison with existing algorithms

The performance of two CGO cases, i.e., #DESC-I:S and #DESC-I:L, was compared to that of existing algorithms. For the two CGO cases, #DESC-I:S has $N = 50$ and $T = 1{,}000$, and thus its NFE is 5.0E4; while #DESC-I:L has $N = 70$ and $T = 3{,}000$, and thus its NFE is 2.1E5.

4.2.1 Existing algorithms

Here we briefly describe basic features of ten algorithms, including their diverse algorithmic types and constraint-handling techniques (some of them can be represented by R_{QC} rules discussed in Sect. 3.2.1). For more details, please refer to corresponding literature.

GA$_{SAFF}$ (Farmani and Wright 2003) is a GA with the self-adaptive fitness formulation. Specifically, each individual is assigned an infeasibility value, i.e., the normalized sum of all constraint violation values, and the two-stage penalty functions are then applied in relation to boundary solutions.

OEA (Liu et al. 2007) is an organizational evolutionary algorithm. The individuals in a population are structured into some organizations, in which all evolutionary operations are applied for simulating the interaction among the organizations. For handling constraints, the static penalty term R_{QC}^P, which uses the penalty coefficient tuned for each problem instance, is considered.

CDE is a cultured differential evolution (Becerra and Coello 2006), i.e., a CA algorithm hybridized with a DE population, or a DE algorithm integrated with a belief space of CA. In CDE, CA uses the external knowledge in its belief space to influence the DE search operations. Its constraint-handling method can be regarded as the natural quality measurement using the R_{QC}^O rule.

PSO$_{SAV}$, or called SAVPSO (Lu and Chen 2008), is a PSO variant, in which each particle adjusts its velocity self-adaptively, according to the run-time information, for searching within S_{PF}. Its constraint-handling method can be described by the quality measurement using R_{QC}^O.

SIMP$_\alpha$, or called αSimplex (Takahama and Sakai 2005), is an α-constrained simplex method. The α-constrained method uses a satisfaction level in order to indicate how well a search point (state) satisfies the constraints. The simplex method uses multiple simplexes to avoid the situation that a single simplex might lose its affine independence. For each simplex, the worst point is mutated, either by a boundary mutation if the point is feasible, or by another mutation to increase the satisfaction level of the point. The α-level is dynamically increased for the problem instances with equality constraints.

ES$_{ATM}$, or called ATMES (Wang et al. 2008), integrates a (μ, λ)-ES with a constraint-handling method called adaptive tradeoff model, which tries to achieve a good tradeoff between feasible and infeasible spaces during different stages of a search process, by taking advantage of the valuable run-time information. For equality constraints, the dynamic adjustment used in (Hamida and Schoenauer 2002) is adopted.

ES$_{SM}$ is a simple multimembered ES (Mezura-Montes and Coello 2005), a $(\mu{+}\lambda)$-ES variant with two basic modifications, i.e., a panmictic combined recombination technique to improve its exploitation capability and a diversity mechanism that keeps best infeasible solutions in the population. Its constraint-handling method can be seen as using R_{QC}^O for the quality measurement. For equality constraints, the dynamic adjustment used in (Hamida and Schoenauer 2002) is considered.

ES$_{RY05}$ (Runarsson and Yao 2005) is a (μ, λ)-ES variant hybridized with differential variation. Its constraint-handling method is the stochastic ranking technique, which is equivalent to applying the R_{QC}^{SR} rule for quality measurement.

SAMO-GA and SAMO-DE (Elsayed et al. 2011) are two self-adaptive multi-operator (SAMO) based algorithms that using multiple genetic and DE operators, respectively. Each search operator has its own sub-population, and the sub-populations are changed by a self-adaptive learning strategy during the evolution process.

4.2.2 Comparison

Tables 10 and 11 list the results obtained by #DESC-I:S and #DESC-I:L, and the existing algorithms.

For the instances with equality constraints, the tolerance value $\epsilon_H = 1E-4$ was considered in all the experiments, except for the experiments by #DESC-I:L, ES_{ATM}, ES_{SM}, and PSO_{SAV}, which used $\epsilon_H = 1E-8$, $5E-6$, $4E-4$, and $1E-3$, respectively.

The NFEs of ES_{ATM}, ES_{SM}, and OEA were 2.4E5. The NFEs of ES_{RY05} and GA_{SAFF} were 3.5E5. The NFEs of $SIMP_\alpha$, CDE, and PSO_{SAV} were 2.9E5–3.3E5, 1.0E5, and 5.0E4, respectively.

For NFE, #DESC-I:L is less than ES_{ATM}, ES_{SM}, ES_{RY05}, GA_{SAFF}, OEA, and $SIMP_\alpha$, while #DESC-I:S is further less than or similar to CDE and PSO_{SAV}.

As for consistently achieving optimal solutions, the two cases #DESC-I:S and #DESC-I:L were successful in eight and twelve instances, while ES_{ATM}, ES_{SM}, ES_{RY05}, GA_{SAFF}, OEA, $SIMP_\alpha$, CDE, and PSO_{SAV} were successful in seven, six, ten, zero, seven, ten, seven, and five instances, respectively. For $G02$, #DESC-I:L performed better than all the existing algorithms.

Compared to PSO_{SAV}, #DESC-I:S was dominating. Furthermore, #DESC-I:S outperformed ES_{SM}, OEA, GA_{SAFF}, and CDE, although it spent a much less NFE. Compared to EA_{SM}, #DESC-I:S achieved better results in

six instances ($G05$, $G06$, $G07$, $G09$, $G10$, and $G13$), and similar results in five instances. #DESC-I:S dominated GA_{SAFF} in all the instances, except $G02$. Compared to OEA, #DESC-I:S achieved better results in four instances ($G05$, $G07$, $G09$, and $G10$), and similar results in six instances. Compared to CDE, #DESC-I:S achieved better results in five instances ($G02$, $G03$, $G05$, $G11$, and $G13$), and similar results in five instances.

Other five algorithms are compared to #DESC-I:L. For ES_{ATM}, the tolerance value for the instances with equality constraints is $\epsilon_H = 5E-6$. As demonstrated in Table 11, #DESC-I:L worked efficiently on harder instances with $\epsilon_H = 1E-8$. Compared to ES_{ATM} on the other nine instances, #DESC-I:L achieved better results in $G02$, $G07$, $G09$, and $G10$, and obtained similar results in five easy instances. Compared to ES_{RY05}, #DESC-I:L achieved better results in $G02$, $G10$, and $G13$, and found similar results in all other instances. Compared to $SIMP_\alpha$, #DESC-I:L performed better in $G02$, $G07$, and $G13$, and found similar results in all other instances. Compared to SAMO-GA, #DESC-I:L performed better in five instances and worse in $G02$. Compared to SAMO-DE, #DESC-I:L performed better in $G07$ and $G10$ and worse in $G02$.

Overall, #DESC-I:L found better results over the existing algorithms on the classic benchmark set, with a less NFE (except for CDE and PSO_{SAV}) and a tighter ϵ_H for problems with equality constraints.

4.3 Additional tests

In this section, additional tests are performed for demonstrating the run-time behavior and the role of landscape tuning of the hybrid CGO case #DESC-I:L.

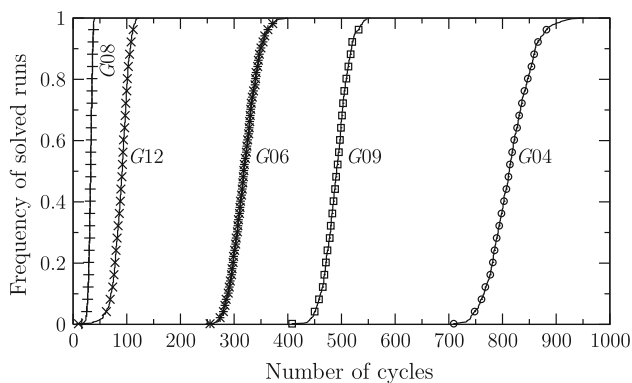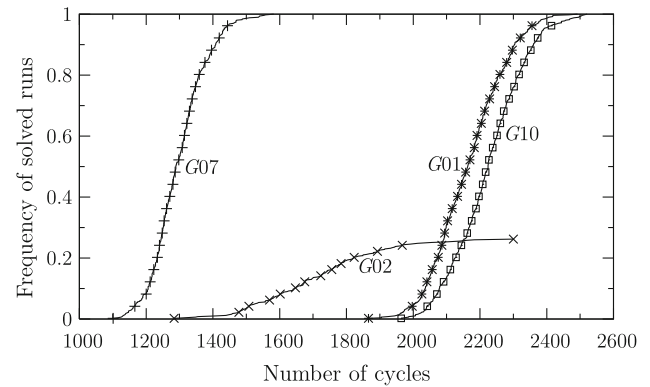**Table 10** Mean results by #DESC-I:S and four existing algorithms

F_P	#DESC-I:S	St. dev.	GA_{SAFF}	OEA	CDE	PSO_{SAV}	ES_{SM}
$G01$	−14.99373	3.32E−03	−14.9993	**−15**	−14.999996	−14.715104	**−15.000**
$G02$	−0.76896	3.38E−02	−0.77512	−0.782518	−0.724886	−0.740577	−0.785238
$G03$	**−1.00050**	7.64E−04	−0.99930	**−1.000**	−0.788635	−1.003367	**−1.000**
$G04$	−30,665.5387	8.58E−08	−30,659.41	−30,665.539	−30,665.539	−30,665.538672	−30,665.539
$G05$	**5,126.49671**	2.83E−08	N/A	5,127.048	5,207.410651	5,202.362681	5,174.492
$G06$	**−6,961.81388**	3.28E−11	−6,961.769	**−6,961.814**	**−6,961.814**	**−6,961.813875**	−6,961.284
$G07$	24.30765	2.09E−03	27.83	24.373	**24.306210**	24.988731	24.475
$G08$	**−0.095825**	5.95E−16	−0.092539	**−0.095825**	**−0.095825**	**−0.095825**	**−0.095825**
$G09$	**680.63006**	1.06E−11	680.97	680.632	**680.630057**	680.655378	680.643
$G10$	7,049.66674	1.27E+00	7,760.54	7,219.011	7,049.248266	7,173.266104	7,253.047
$G11$	**0.74990**	6.00E−15	0.7546	**0.750**	0.757995	**0.749002**	**0.75**
$G12$	**−1.00000**	0.00E+00	−0.99972	**−1**	**−1.000000**	**−1**	**−1.000**
$G13$	0.055977	7.94E−01	N/A	0.053969	0.288324	0.552753	0.166385

Table 11 Mean results by #DESC-I:L and four existing algorithms

F_P	#DESC-I:L	St. dev.	$SIMP_\alpha$	ES_{ATM}	ES_{RY05}	SAMO-GA	SAMO-DE
$G01$	**−15.00000**	6.67E−08	**−15.00000**	**−15.000**	**−15.000**	**−15.0000**	**−15.0000**
$G02$	−0.79080	1.10E−02	−0.78419	<u>−0.790148</u>	−0.782715	−0.79605	−0.79874
$G03$	**−1.00000**	2.84E−07	**−1.00050**	**−1.000**	**−1.001**	**−1.0005**	**−1.0005**
$G04$	**−30,665.5387**	2.94E−10	**−30,665.5387**	**−30,665.539**	**−30,665.539**	**−30,665.5386**	**−30,665.5386**
$G05$	**5,126.49811**	2.29E−11	**5,126.49671**	5,127.648	**5,126.497**	5,127.976	**5,126.497**
$G06$	**−6,961.81388**	3.27E−11	**−6,961.81388**	**−6,961.814**	**−6,961.814**	**−6,961.81388**	**−6,961.81388**
$G07$	**24.30621**	2.24E−10	24.30626	24.316	**24.306**	24.4113	24.3096
$G08$	**−0.095825**	1.00E−15	**0.095825**	**−0.09825**	**−0.095825**	**−0.095825**	**−0.095825**
$G09$	**680.63006**	2.91E−12	683.63006	683.639	**680.630**	683.634	**680.630**
$G10$	**7,049.24802**	3.28E−08	**7,049.24802**	7,250.437	7,049.250	7,144.40311	7,059.81345
$G11$	**0.75000**	4.22E−15	**0.74990**	**0.75**	**0.750**	**0.7499**	**0.7499**
$G12$	**−1.00000**	0.00E+00	**1.00000**	**−1**	**−1.000**	**1.0000**	**1.0000**
$G13$	**0.053950**	4.78E−16	0.066770	0.053959	0.066770	0.054028	**0.053942**

4.3.1 Run length distribution

The run-length distribution (RLD) (Hoos and Stutzle 2004) is suitable for characterizing the run-time behavior of a stochastic algorithm case on each problem instance. Figures 2, 3 and 4 show the frequency of solved runs along with the number of cycles for #DESC-I:L on all thirteen instances. All the instances were 100 % solved, except for $G02$. The steepness of each RLD discloses nontrivial information for the variance of the run length of each algorithm case. For the problem instances with equality constraints, their RLDs are influenced by the threshold cycle, i.e., $c_{TH} = C_{RTU} \cdot T = 1{,}500$. Here, $G03$ and $G11$ were mostly solved before c_{TH}, whereas $G05$ and $G13$ were solved after c_{TH} and most of the solutions were obtained quite soon after c_{TH}, although there is a longer tail for $G05$. For all instances without equality constraints, their RLDs are steep, except for $G02$. Since the success frequency is

Fig. 3 RLDs for #DESC-I:L on $G01$, $G02$, $G07$, and $G10$
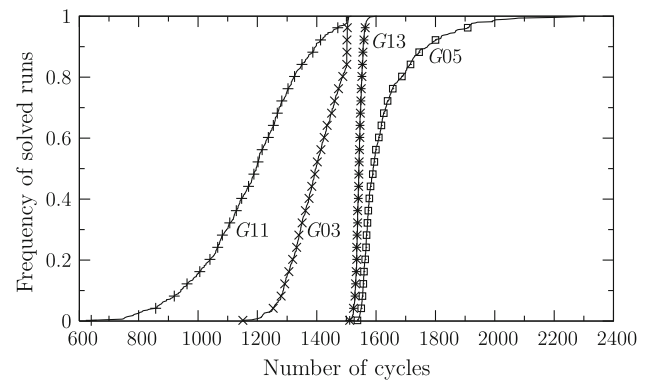
Fig. 4 RLDs for #DESC-I:L on $G03$, $G05$, $G11$, and $G13$

sufficiently high, using independent runs can further improve the performance (Hoos and Stutzle 2004).

RLD gives a quite accurate estimation of competency of an stochastic algorithm. In addition, RLD might be easily approximated using a Weibull distribution (Hoos and

Fig. 2 RLDs for #DESC-I:L on $G04$, $G06$, $G08$, $G09$, $G12$

Stutzle 2004) with only two parameters. In other words, the information of algorithm behavior might be compressed and stored for possible analysis and learning.

4.3.2 Landscape tuning

In the facilitator, R_{ADJ}^{RR} essentially encoding the knowledge on handling equality constraints for adaptively shaping the problem landscape. Because $\$\mathbf{x}_{FB} = \$\mathbf{x}_{DP}$ is a run-time chunk in M_S, only three other parameters might be used for tuning the landscape. Note that R_{ADJ}^{RR} does not account for landscape tunings on the problem instances without equality constraints.

Tables 12, 13 and 14 report the average results for #DESC-I:L using different C_{RRE}, C_{RNU}, and C_{RTU} values. The algorithm achieved optimal results in a large parameter space. C_{RRE} should not be too large, since the relaxation loses its usage. C_{RNU} should not be close to 1 to ensure that Eq. 4 is executed. C_{RTU} should not be too small or too large. Before $t_{TH} = \text{INT}(C_{RTU} \cdot T)$, the adaptive ratio-reaching technique plays the role for ensuring a suitable pace to reach the neighborhood of the optimal solution; after t_{TH}, R_{QC}^{RRR} is equivalent to R_{QC}^{O}, which is useful for fine search.

We expect that any global features that can explicitly tuning the problem landscape might be encoded in the group facilitator. For a better understanding the capability of each R_M rule, it is certainly useful to analyze the influence for different parameter combinations on different problems. It might be more robust and effective to use an ensemble of constraint handling techniques (Mallipeddi and Suganthan 2010), including those R_{QC} rules in Sect. 3.2.1.

Furthermore, the essential role of landscape tuning is only providing transformed problem instances. The majority of efforts for achieving better performance should still be placed on the customization in the algorithmic space formed from the portfolio of ESHs.

4.3.3 Expanded problem set

We continue to evaluate on an expanded problem set from the CEC06 competition (Liang et al. 2006). For end users, this might be viewed as the situation of their problems changing over time. Table 15 give the results by #DESC-I:L, SAMO-GA, and SAMO-DE. Here #DESC-I:L is running at $N = 80$. Thus for all the algorithms, NFE = 2.4E5.

Compared to SAMO-GA, #DESC-I:L performed better in six instances and only worse in $G17$ and $G18$. Compared to SAMO-DE, #DESC-I:L achieved better results in three instances ($G14$, $G19$, $G23$), and comparable results in three instances ($G15$, $G16$, and $G24$). In addition, the median solutions of #DESC-I:L achieved or approached closely to the optimal solutions.

Table 12 Results for #DESC-I:L using different C_{RRE} values on the instances with equality constraints

F_P	$C_{RRE} = 1\text{E}0$	$C_{RRE} = 1\text{E}1$	$C_{RRE} = 1\text{E}2$	$C_{RRE} = 1\text{E}3$	$C_{RRE} = 1\text{E}4$
$G03$	**−1.00000**	**−1.00000**	**−1.00000**	−0.99991	−0.99860
$G05$	**5,126.49811**	**5,126.49811**	**5,126.49811**	5,126.49823	5,126.49917
$G11$	**0.75000**	**0.75000**	**0.75000**	**0.75000**	**0.75000**
$G13$	**0.053950**	**0.053950**	**0.053950**	**0.053950**	0.054721

Table 13 Results for #DESC-I:L using different C_{RNU} values on the instances with equality constraints

F_P	$C_{RNU} = 0.00$	$C_{RNU} = 0.25$	$C_{RNU} = 0.50$	$C_{RNU} = 0.75$	$C_{RNU} = 1.00$
$G03$	**−1.00000**	**−1.00000**	**−1.00000**	**−1.00000**	−0.03707(414)
$G05$	**5,126.49811**	**5,126.49811**	**5,126.49811**	5,126.50580	5,313.93466
$G11$	**0.75000**	**0.75000**	**0.75000**	**0.75000**	1.00707(40)
$G13$	**0.053950**	**0.053950**	**0.053950**	**0.053950**	(500)

Table 14 Results for #DESC-I:L using different C_{RTU} values on the instances with equality constraints

F_P	$C_{RTU} = 0.00$	$C_{RTU} = 0.25$	$C_{RTU} = 0.50$	$C_{RTU} = 0.75$	$C_{RTU} = 1.00$
$G03$	−0.96073	−0.99699	**−1.00000**	**−1.00000**	**−1.00000**
$G05$	5,126.49823	5,126.49812	**5,126.49811**	**5,126.49811**	(500)
$G11$	**0.75000**	**0.75000**	**0.75000**	**0.75000**	**0.75000**
$G13$	0.085691	0.055809	**0.053950**	**0.053950**	(500)

Table 15 Results by #DESC-I:L, #DESC-I:L2, SAMO-GA, and SAMO-DE

F_P	#DESC-I:L	St. dev.	*Median*	#DESC-I:L2	St. dev.	Median	SAMO-GA	SAMO-DE
G14	**−47.76489**	5.00E−15	**−47.76489**	**−47.76489**	3.00E−15	**−47.76489**	−46.47318	−47.68115
G15	**961.71502**	5.00E−15	**961.71502**	**961.71502**	0.00E+00	**961.71502**	961.71509	**961.71502**
G16	**−1.905155**	0.00E+00	**−1.905155**	**−1.905155**	0.00E+00	**−1.905155**	−1.905154	**−1.905155**
G17	8,887.26933	4.17E+01	**8,853.53967**	8,891.23276	4.24E+01	**8,853.53967**	8,853.8871	**8,853.5397**
G18	−0.82591	7.79E−02	**−0.866025**	−0.85036	5.25E−02	**−0.866025**	−0.865545	−0.866024
G19	32.6912	8.52E−02	32.66595	**32.65559**	2.01E−07	**32.65559**	36.427463	32.75734
G21	202.98569	2.96E+01	**193.72451**	193.76478	8.90E−01	**193.72451**	246.09154	193.77138
G23	−382.93683	4.86E+01	−400.02517	−385.49695	6.42E+01	**−400.05510**	−194.76034	−360.81766
G24	**−5.508013**	0.00E+00	**−5.508013**	**−5.508013**	0.00E+00	**−5.508013**	**−5.508013**	**−5.508013**

Previous CGO cases only use naive search operators, for the sake of simplicity in description. The performance might be benefited from other superior operators. In Table 15, #DESC-I:L2 is slightly tuned from #DESC-I:L by using $C_{CG} = 0.5$ for G.DE2. In other words, the DE operator is tuned from best/2/bin into current-to-best/2/bin, where "current-to-best" is a more advanced sub-strategy in DE operators (Elsayed et al. 2013; Zhang and Sanderson 2009). Compared to #DESC-I:L, this minor change achieved statistically significant improvement on three instances (G18, G19, and G20). In addition, all median solutions of #DESC-I:L2 achieved the optimal solutions.

The advance of #DESC-I:L2 might be largely attributed to better preservation of the population diversity, since "current-to-best" relies less on the currently best state. The cooperative group itself provides an implicit mechanism to preserve diversity in the memory of agents. However, it is still beneficial to prevent states becoming too similar. In a human idea-generating group, redundant states are often automatically filtered. Thus, CGO cases might be further improved by considering some explicit strategies to control redundant states, e.g., the speciation method and R_{GE}^{RND} used in multimethod (Vrugt et al. 2009) and multi-operator algorithms (Elsayed et al. 2011, 2012, 2013).

In addition, the CGO system might also be benefited from including advanced search operators (e.g., those used in SAMO-DE) to further improve the performance on some problem instances, e.g., G17 and G18.

4.4 Summary

The algorithm selection process was consistent with two nontrivial insights in algorithm portfolio design (Huberman et al. 1997; Streeter and Smith 2008). First, combining competent strategies into a portfolio may improve the overall performance by exploiting the negative correlation among the offline performance of individual strategies

(e.g., #DE2 and #SC). Second, customized cooperative search (#DESC-I) further tuned the portfolio performance than when the individual strategies are independent (#DESC). The two insights might help users to quickly find an effective algorithm case in a large algorithmic space.

We then compared the performance of #DESC-I with existing algorithms in diverse paradigms and with different constraint-handling techniques. With less computational times, CGO cases achieved competitive performance as compared to existing algorithms.

The behavior of #DESC-I was demonstrated using the run length distributions. We also performed a systematic test to show the impacts in tuning the problem landscape. An expanded problem set were also tested, and a slightly tuned CGO case was also demonstrated.

5 Related work and discussion

In Sect. 2.7, we have discussed related work on two main elements, i.e., the cooperative group paradigm and the algorithm portfolio design, of the CGO system. The implementation itself, shows the relation with existing algorithm paradigms, e.g., PSO, DE, and SCO, and the role of shaping landscape in handling constraints. The essential role of the CGO system, however, is a development framework for realizing customized algorithm portfolios with low-level hybridization.

Many conceptual frameworks (Milano and Poli 2004; Raidl 2006; Taillard et al. 2001; Talbi 2002) have been proposed for formalizing (hybrid) metaheuristics. In adaptive memory programming (AMP) (Taillard et al. 2001) and a refined version called the multiagent metaheuristic architecture (MAGMA) (Milano and Poli 2004), metaheuristics are defined as a set of algorithmic components operating on a shared memory. They provide a unified view for various metaheuristics, e.g., iterative LS, ACO, and GA. Using a single memory, however, limits the capability to support a cooperative group. There is also a

unified view (Raidl 2006) that hybrid metaheuristics can be built on a toolbox of components, based on a common pool template. In (Talbi 2002), a taxonomy has been proposed to distinguish low and high levels, and relay and teamwork models for hybrid metaheuristics. The hybridization among stand-alone algorithms are often limited in the high level, and heterogeneous algorithms can only independently run or light-weighted cooperate through a shared medium. From this taxonomy, hybrid CGO cases belong to a mix of low and high levels, since individual ESHs can work in a self-containing way, while the cooperative search at the low level is emphasized. The CGO framework not only supports simple relay and teamwork models based on the individual and/or group memory, but also the low-level cooperation among heterogeneous ESHs on customized memory elements. In addition, the memory protocol for individual and group memory provides a natural support for quarantining detrimental results.

Various software frameworks have been proposed for realizing hybrid metaheuristics. As surveyed in (Parejo et al. 2012), 33 software frameworks have been identified, and 10 of them have been selected for comparison. Typical examples include HeuristicLab, ParadisEO, and JCLEC, etc. Here we discuss the relations on toolbox elements, and design of pure and hybrid metaheuristics.

These frameworks contain a toolbox of low-level components for the code reuse. In ParadisEO, low-level components are defined as *Helpers* in some categories, i.e., evolutionary helpers (e.g., transformation, selection, and replacement operations), local search helpers (e.g., generic and problem-specific classes to local search metaheuristics), and some special helpers (e.g., the management of parallel and distributed models). In HeuristicLab, the toolbox contains atomic *operators* (e.g., mutation, crossover, and selection operators) working on data structures called *scopes*, and micro operators might be defined using a operator graph. In JCLEC, individuals and elementary operations are represented by *IIndividual* and *ITool* interfaces (e.g., *IProvider*, *ISelector*, *IRecombinator* and *IMutation* respectively for individual initialization, selection, recombination and mutation operations). The toolbox in the CGO system follows the same principle for the code reuse. There are chunks and four primary rule interfaces (R_M, R_{IE}, R_{UE}, and R_{GE}). Association interfaces are provided in macro components. For example, R_{SEL} might be used in $R_{UE:X}$ and R_{GE}, and R_{QC} and R_{ADJ} are used in some macro R_M rules. The R_{GE} rule provides a unified entrance for various kinds of search operators, e.g., the operators extracted from DE, PSO, and SCO. Typical operators, e.g., local search and recombination, might be provided in macro R_{GE} rules. Heterogeneous R_{GE} rules might also be implemented using arbitrary input/output chunks. Generic low-level hybridization models, e.g., relay (Talbi 2002),

can be implemented in macro rules. All these low-level features might be realized in existing software frameworks, but a unified R_{GE} interface provides a basic step to support heterogeneous ESHs. In addition, the use of R_M provides a way to utilize the global knowledge on the problem landscape, and might relieve the requirement and difficulty of realizing too many competent components in the toolbox.

Algorithm paradigms are then realized using low-level components. In JCLEC, Algorithms are coded from the *ISystem* and *IAlgorithm* interfaces. In ParadisEO, The run of the metaheuristics are coded into *Runners* by invoking the helpers to perform specific actions on their data, and pure and hybrid metaheuristics are coded into *Solvers*. In HeuristicLab, some algorithms might be designed by using an operator graph, and complex algorithms are created by writing code. Each of these algorithm paradigms might use a configuration file to define parameters and some operators of specific interfaces. These frameworks support different paradigms, e.g., stochastic local search, GA, PSO, and DE, but these algorithms are coded separately. Each algorithm paradigm can only support a very limited algorithmic space to limit the flexibility to adapt to user-specific problem sets, and the configuration file can only provide very limited operator replacements. Hybridization between paradigms often needs to extensively write code by realizing a new paradigm. Thus, there is often a steep learning curve for existing frameworks (Parejo et al. 2012).

For the CGO system, the framework can be seen as a concrete and unified algorithm paradigm, and the script is its configuration file. The setting in a cooperative group with flexible memory protocol facilitates the realization of heterogeneous ESHs. The effort of designing a stand-alone algorithm (e.g., PSO, DE, and SCO) is reduced to implement a few algorithmic components and then organize them into an ESH using the script. Furthermore, hybridization can be realized by only defining customized algorithm portfolios in the CGO script, without writing any additional code. Based on the insights from algorithm portfolio design, the capability to adapt to specific problem sets might be benefited from negative correlations among the offline performance of individual ESHs. If user-specific problem sets change over time, new heuristics that mainly tackle some of new problems unsolved by existing heuristics can be added into the portfolio, and obsolete ESHs can be removed, if necessarily. There are two kinds of users for a CGO system. Advanced users might focus on realizing algorithmic components in the toolbox (and even improving the CGO framework for highly advanced users), and then designing new ESHs using the script. Basic users might only need pay attention to design customized portfolios based on offline performance of ESHs, and do not need advanced knowledge on algorithmic components and framework details.

It is also interesting to discuss about the similarities and differences with existing multimethod (Vrugt et al. 2009), multi-operator (Elsayed et al. 2011, 2012, 2013), and ensemble methods (Mallipeddi et al. 2010a, b; Mallipeddi and Suganthan 2010). These algorithms provides high-performance realizations in combining the strengths of different search operators (e.g., ES, GA, DE, and PSO) and constraint-handling techniques. These operators might be significant sources for the CGO toolbox. Furthermore, a distinguished feature of these algorithms is that multiple operators compete the offspring generation in sub-populations with adaptively varied sizes, to favor some operators that exhibit higher reproducive success, over the evolution process. Compared to these methods, the CGO system provide a flexible way to support customized low-level cooperation among heterogeneous ESHs that interact on multiple chunks in the private memory of individual agents.

5.1 Possible extensions

To reduce the complexity for description, the current CGO system is realized in the naive form. For future work, some possible extensions are discussed.

Many extensions do not require any changes in the framework. As shown in previous tests, the system can benefit from advanced algorithmic components that tailor to specific problem structures. For large-scale problems, many practical ESHs might only work on partial problems. The agents might still benefit from their cooperative search if the partial problems handled by different ESHs are sufficiently overlapped (otherwise agents might only rely on local reactive strategies; Liu et al. 2002).

It is also useful to formally define commonly-used macro components for simplifying the reusing of algorithmic components and implementing new ESHs.

Some extensions require more or less changes on the CGO framework. Here we list two possible extensions, then discuss possible changes on the framework.

First, the facilitator might provide multiple R_M rules to support an ensemble of constraint handling techniques (Mallipeddi and Suganthan 2010). Second, the manual process used in the experiments might be regarded as a greedy maximum set cover strategy, and the online selection of ESHs in the portfolio is purely proportionally, where individual ESHs might be favored by providing relatively larger C_W values. It is also possible to utilize some intelligent algorithm selection methods (Birattari et al. 2002; Smith-Miles 2008; Xie and Zhang 2004), by inferring from the integration of run-time information and offline performance of individual ESHs in the portfolio.

To support online self-adaptation, the basic intuition (Elsayed et al. 2011; Mallipeddi and Suganthan 2010;

Vrugt et al. 2009) is to favor operators that contribute more for the overall performance. The CGO framework might incorporate three basic changes: (1) each agent associates newly generated chunks with some tags. The tags might include the identifiers of components in usage (e.g., ESH and R_M). (2) The facilitator provides some evaluations (Elsayed et al. 2011; Mallipeddi and Suganthan 2010; Vrugt et al. 2009) on the contributions of the chunks associated with different tags. (3) Based on the evaluation information, each agent updates the weights for algorithmic components in usage. These changes, nevertheless, might only cause few changes in the script.

6 Conclusions

The cooperative group optimization (CGO) system is a generic framework to combine the advantages of the cooperative group and low-level portfolio design for realizing CGO algorithms. In the cooperative group, the agents not only search the problem landscape in a parallel way by using diverse knowledge in their individual memory, but also cooperate with their peers by diffusing novel knowledge through the group memory. The search process might also be accelerated through adaptive landscape shaping by a passive group leader.

The CGO framework is driven by the CGO script that assembles algorithmic components in the CGO toolbox. The CGO script is based on a multilayer design to facilitate the accumulation of knowledge over time. Based on the CGO system, implementing a stand-alone embedded search heuristic (ESH) only needs to implement a few algorithm components, and customized portfolios can be defined using the script without writing any additional code. Possible cooperative portfolios form a large algorithmic space. The offline performance of individual ESHs might be used for help adapting the framework to user-specific problem sets, based on the nontrivial insights from algorithm portfolio design. Advanced users might focus on realizing algorithmic components and then defining new ESHs using the script, whereas basic users might only pay attention to design customized portfolios on existing ESHs.

Finally, we have discussed several aspects of the proposed CGO system that warrant further investigation.

References

Anderson JR (2005) Human symbol manipulation within an integrated cognitive architecture. Cognit Sci 29(3):313–341

Barkat Ullah ASSM, Sarker R, Cornforth D, Lokan C (2009) AMA: a new approach for solving constrained real-valued optimization problems. Soft Comput 13(8-9):741–762

Becerra RL, Coello CAC (2006) Cultured differential evolution for constrained optimization. Comput Methods Appl Mech Eng 195(33–36):4303–4322

Beyer HG (2001) On the performance of (1, λ)-evolution strategies for the ridge function class. IEEE Trans Evol Comput 5(3):218–235

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Genetic and evolutionary computation conference. Morgan Kaufmann, New York, pp 11–18

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 11:4135–4151

Boyd R, Richerson PJ, Henrich J (2011) The cultural niche: Why social learning is essential for human adaptation. Proc Natl Acad Sci 108:10918–10925

Cahon S, Melab N, Talbi EG (2004) ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics. J Heurist 10:357–380

Chen X, Ong YS, Lim MH, Tan KC (2012) A multi-facet survey on memetic computation. IEEE Trans Evol Comput 15(5):591–607

Curran D, O'Riordan C (2006) Increasing population diversity through cultural learning. Adapt Behav 14(4):315–338

Danchin É, Giraldeau LA, Valone T, Wagner R (2004) Public information: from nosy neighbors to cultural evolution. Science 305(5683):487–491

Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186(2–4):311–338

Dennis A, Valacich J (1993) Computer brainstorms: more heads are better than one. J Appl Psychol 78(4):531–537

Edgington T, Choi B, Henson K, Raghu T, Vinze A (2004) Adopting ontology to facilitate knowledge sharing. Communi ACM 47(11):85–90

Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124–141

Elsayed S, Sarker RA, Essam DL (2011) Multi-operator based evolutionary algorithms for solving constrained optimization problems. Comput Oper Res 38:1877–1896

Elsayed S, Sarker RA, Essam DL (2012) On an evolutionary approach for constrained optimization problem solving,. Appl Soft Comput 12(10):3208–3227

Elsayed S, Sarker RA, Essam DL (2013) An improved self-adaptive differential evolution algorithm for optimization problems. IEEE Trans Ind Inf 9(1):89–99

Ericsson KA, Kintsch W (1995) Long-term working memory. Psychol Rev 102(2):211–245

Farmani R, Wright J (2003) Self-adaptive fitness formulation for constrained optimization. IEEE Trans Evol Comput 7(5):445–455

Galef BG (1995) Why behaviour patterns that animals learn socially are locally adaptive. Anim Behav 49(5):1325–1334

Gigerenzer G, Selten R (2001) Bounded rationality: the adaptive toolbox. MIT Press, Cambridge

Glenberg AM (1997) What memory is for. Behav Brain Sci 20(1):1–55

Goncalo JA, Staw BM (2006) Individualism–collectivism and group creativity. Org Behav Human Decis Process 100:96–109

Hamida SB, Schoenauer M (2002) ASCHEA: new results using adaptive segregational constraint handling. In: Congress on evolutionary computation. IEEE, Honolulu, pp 884–889

He S, Wu Q, Saunders JR (2009) Group search optimizer: an optimization algorithm inspired by animal searching behavior. IEEE Trans Evol Comput 13(5):973–990

Hinton GE, Nowlan SJ (1987) How learning can guide evolution. Complex Syst 1:495–502

Hoos HH, Stutzle T (2004) Stochastic local search: foundations and applications. Elsevier, Burlington

Huberman BA, Lukose RM, Hogg T (1997) An economics approach to hard computational problems. Science 275(5296):51–54

Jin Y, Olhofer M, Sendhoff B (2002) A framework for evolutionary optimization with approximate fitness functions. IEEE Trans Evol Comput 6(5):481–494

Kennedy J, Eberhart RC, Shi Y (2001) Swarm intelligence. Morgan Kaufmann, San Mateo

Kohn NW, Smith SM (2011) Collaborative fixation: effects of others' ideas on brainstorming. Appl Cognit Psychol 25(3):359–371

Laland KN (2004) Social learning strategies. Learn Behav 32(1):4–14

Lau HC, Wan WC, Halim S, Toh K (2007) A software framework for fast prototyping of meta-heuristics hybridization. Int Trans Oper Res 14(2):123–141

Leonard NE, Shen T, Nabet B, Scardovi L, Couzin ID, Levin SA (2012) Decision versus compromise for animal groups in motion. Proc Natl Acad Sci 109(1):227–232

Liang JJ, Runarsson TP, Mezura-Montes E, Clerc M, Suganthan PN, Coello CAC, Deb K (2006) Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization. Tech. rep., Nanyang Technological University, Singapore

Liu J, Han J, Tang YY (2002) Multi-agent oriented constraint satisfaction. Artif Intell 136(1):101–144

Liu J, Tsui KC (2006) Toward nature-inspired computing. Commun ACM 49(10):59–64

Liu J, Zhong W, Hao L (2007) An organizational evolutionary algorithm for numerical optimization. IEEE Trans Syst Man Cybern Part B 37(4):1052–1064

Lu H, Chen W (2008) Self-adaptive velocity particle swarm optimization for solving constrained optimization problems. J Global Optim 41(3):427–445

Mallipeddi R, Mallipeddi S, Suganthan PN (2010a) Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput 11(2):1679–1696

Mallipeddi R, Mallipeddi S, Suganthan PN (2010b) Ensemble strategies with adaptive evolutionary programming. Inf Sci 180(2):1571–1581

Mallipeddi R, Suganthan PN (2010) Ensemble of constraint handling techniques. IEEE Trans Evol Comput 14(4):561–579

Mezura-Montes E, Coello CAC (2005) A simple multimembered evolution strategy to solve constrained optimization problems. IEEE Trans Evol Comput 9(1):1–17

Milano M, Poli A (2004) MAGMA: a multiagent architecture for metaheuristics. IEEE Trans Syst Man Cybern Part B 34(2):925–941

Nemeth CJ (1986) Differential contributions of majority and minority influence. Psychol Rev 93(1):23–32

Omran MGH, Engelbrecht AP (2009) Free search differential evolution. In: IEEE congress on evolutionary computation. IEEE, Trondheim, pp 110–117

Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst Man Cybern Part B: Cybern 36(1):141–152

Parejo JA, Ruiz-Cortes A, Lozano S, Fernandez P (2012) Metaheuristic optimization frameworks: a survey and benchmarking. Soft Comput 16(3):527–561

Paulus PB (2000) Groups, teams, and creativity: the creative potential of idea-generating groups. Appl Psychol 49(2):237–262

Platon E, Mamei M, Sabouret N, Honiden S, Van Parunak H (2007) Mechanisms for environments in multi-agent systems: survey and opportunities. Auton Agents Multi Agent SystAgents and Multi-Agent Systems 14(1):31–47

Price K, Storn RM, Lampinen JA (2005) Differential evolution: a practical approach to global optimization. Springer, NY

Raidl GR (2006) A unified view on hybrid metaheuristics. In: International conference on hybrid metaheuristics. Gran Canaria, pp 1–12

Reynolds RG, Peng B, Ali MZ (2008) The role of culture in the emergence of decision-making roles: an example using cultural algorithms. Complexity 13(3):27–42

Runarsson TP, Yao X (2005) Search biases in constrained evolutionary optimization. IEEE Trans Syst Man Cybern Part C 35(2):233–243

Salomon R (1996) Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. BioSystems 39(3):263–278

Satzinger JW, Garfield MJ, Nagasundaram M (1999) The creative process: the effects of group memory on individual idea generation. J Manage Inf Syst 15(4):143–160

Smith-Miles K (2008) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv 41(6), Art. No. 6

Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. Eur J Oper Res 185(3):1155–1173

Stadler PF, Happel R (1999) Random field models for fitness landscapes. J Math Biol 38(5):435–478

Streeter M, Smith SF (2008) New techniques for algorithm portfolio design. In: Conference in Uncertainty in Artificial Intelligence, pp. 519–527. AUAI, Helsinki, Finland

Taillard ED, Gambardella LM, Gendreau M, Potvin JY (2001) Adaptive memory programming: a unified view of metaheuristics. Eur J Oper Res 135(1):1–16

Takahama T, Sakai S (2005) Constrained optimization by applying the alpha constrained method to the nonlinear simplex method with mutations. IEEE Trans Evol Comput 9(5):437–451

Talbi EG (2002) A taxonomy of hybrid metaheuristics. J Heurist 8(5):541–564

Tomasello M, Kruger A, Ratner H (1993) Cultural learning. Behav Brain Sci 16(3):495–511

Ventura S, Romero C, Zafra A, Delgado JA, Hervas C (2008) JCLEC: a Java framework for evolutionary computation. Soft Comput 12(4):381–392

Vrugt JA, Robinson BA, Hyman JM (2009) Self-adaptive multi-method search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259

Wagner S (2009) Heuristic optimization software systems—modeling of heuristic optimization algorithms in the heuristiclab software environment. Phd thesis, Johannes Kepler University, Linz

Wang Y, Cai Z, Zhou Y, Zeng W (2008) An adaptive tradeoff model for constrained evolutionary optimization. IEEE Trans Evol Comput 12(1):80–92

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Woolley AW, Chabris CF, Pentland A, Hashmi N, Malone TW (2010) Evidence for a collective intelligence factor in the performance of human groups. Science 330(6004):686–688

Xie XF, Liu J (2005) A compact multiagent system based on autonomy oriented computing. In: IEEE/WIC/ACM international conference on intelligent agent technology. IEEE, Compiegne, pp 38–44

Xie XF, Liu J (2009) Multiagent optimization system for solving the traveling salesman problem (TSP). IEEE Trans Syst Man Cybern Part B Cybern 39(2):489–502

Xie XF, Zhang WJ (2004) SWAF: swarm algorithm framework for numerical optimization. In: Genetic and evolutionary computation conference (GECCO). Springer, Seattle, pp 238–250

Xie XF, Zhang WJ, Yang ZL (2002) Social cognitive optimization for nonlinear programming problems. In: International conference on machine learning and cybernetics. IEEE, Beijing, pp 779–783

Zhang J, Sanderson AC (2009) JADE: adaptive differential evolution with optional external archive. IEEE Trans Evol Comput 13(5):945–958

Zhang WJ, Xie XF (2003) DEPSO: hybrid particle swarm with differential evolution operator. In: IEEE international conference on systems, man, and cybernetics. IEEE, Washington, DC, pp 3816–3821