Multiagent Optimization System for Solving the Traveling Salesman Problem (TSP)

Xiao-Feng Xie, Member, IEEE, and Jiming Liu, Senior Member, IEEE

Abstract—The multiagent optimization system (MAOS) is a nature-inspired method, which supports cooperative search by the self-organization of a group of compact agents situated in an environment with certain sharing public knowledge. Moreover, each agent in MAOS is an autonomous entity with personal declarative memory and behavioral components. In this paper, MAOS is refined for solving the traveling salesman problem (TSP), which is a classic hard computational problem. Based on a simplified MAOS version, in which each agent manipulating on extremely limited declarative knowledge, some simple and efficient components for solving TSP, including two improving heuristics based on a generalized edge assembly recombination, are implemented. Compared with metaheuristics in adaptive memory programming, MAOS is particularly suitable for supporting cooperative search. The experimental results on two TSP benchmark data sets show that MAOS is competitive as compared with some state-of-the-art algorithms, including the Lin–Kernighan–Helsgaun, IBGLK, PHGA, etc., although MAOS does not use any explicit local search during the runtime. The contributions of MAOS components are investigated. It indicates that certain clues can be positive for making suitable selections before time-consuming computation. More importantly, it shows that the cooperative search of agents can achieve an overall good performance with a macro rule in the switch mode, which deploys certain alternate search rules with the offline performance in negative correlations. Using simple alternate rules may prevent the high difficulty of seeking an optimal rule that is efficient for a large data set.

Index Terms—Cooperative systems, multiagent systems, optimization methods, traveling salesman problems (TSPs).

I. INTRODUCTION

The traveling salesman problem (TSP) [1], [2] is a classic combinatorial optimization problem. Although it can be easily formulated, it exhibits various interesting aspects of hard computational problems and has often served as a touchstone for novel approaches [3]–[6]. Moreover, TSP has various applications, such as very large scale integration (VLSI) design [7], rearrangement clustering [8], predicting protein functions [9], etc.

Due to the NP-hardness in a strong sense [10], various heuristic procedures [1], [2], [11], such as construction and local search (LS) strategies, have been applied for solving TSP instances with near-optimal solutions in reasonable amounts of running time. Representative LS strategies include 2-opt [12], and 5-opt [13], the Lin–Kernighan heuristic (LK) [12], etc. Some simple metaheuristics, such as restart [14], [15] and iterated LS (ILS) [16]–[19], are also widely used.

Recombination search (XS) has been first introduced in a genetic algorithm (GA) [20]. For TSP, domain-specific XS strategies have been proposed from position- and order-based [21]–[23] to edge-based variants [24]–[29].

Various metaheuristics have been proposed, which may use common search strategies, such as construction, LS, XS, etc., as their components for solving TSP. Typical examples include ILS [16]–[19], GAs [20], [30], ant colony optimization (ACO) [31], [32], particle swarm optimization (PSO) [33], [34], etc.

Most state-of-the-art algorithms use certain LS heuristics, particularly LK variants, as the major search components during the runtime. For example, the LK–Helsgaun (LKH) [15] itself is an LK variant. In genetic LS (GLS) [27], [30], [35], [36], XS strategies are chained with LS strategies for producing high-quality solutions. Hence, among existing XS strategies, edge assembly crossover (EAX) [28], [37] attracts our interest since it had demonstrated its efficiency in certain selected TSP instances without resorting to the power of LS.

The multiagent optimization system (MAOS) [38] is a nature-inspired method, which addresses the self-organization of agents working with limited declarative knowledge [40] and simple procedural knowledge under ecological rationality [41]. Specifically, agents explore in parallel [33], [42], based on socially biased individual learning (SBIL) [43], [44] and indirectly interact with other agents through sharing public information [45] organized in the environment (ENV) [46], [47]. Recently, MAOS has been applied to some hard problems [38], [48]. For TSP, the experiments on MAOS with an EAX variant have produced some primary encouraging results [48].

In this paper, MAOS is refined to implement simple and efficient knowledge components for solving TSP. In Section II, a simplified MAOS with limited memory specifications and simplified rules is described in a formalized form. In Section III, some existing knowledge components for TSP, particularly search behaviors and related auxiliary data structures, are described. In Section IV, MAOS is implemented with simple and efficient components for solving TSP. In Section V, the experimental results by MAOS cases on some TSP instances are presented and compared with some existing algorithms. In the last section, this paper is concluded.

II. MAOS

In this section, a multiagent optimization framework, in symbolic representation, is described. Afterward, a simplified MAOS version is defined based on the framework.
A. Symbolic Representation

The general problem-solving capability arises from the interaction of declarative and procedural knowledge [49], [50]. Basic declarative and procedural knowledge components are represented in units called chunks and rules, respectively [49]. A chunk aggregates a small amount of information, and a rule specifies a certain action on certain information.

Moreover, macro components or modules may be organized by a combination of elemental knowledge components.

Each knowledge component is an object that can be defined by its actual type and certain setting parameters. Moreover, each action in a rule or a macro component has an interface with input/output parameters. Each component or parameter has a name. Each parameter has a formal type. Each type [51] is used for designating a specific organization structure.

For any two types, one type is defined as compatible with another type if it is an instance, i.e., a child type, of another type. Hence, the actual type of each valid parameter value must be compatible with the formal type of the parameter. Two types are not considered as compatible if they are different, unless there is an explicit declaration.

For a chunk, its actual type is a specific data structure. For a rule (R), its actual type is represented as $R_{\text{L\_NAME}}$ [38], where the tags, i.e., L\_KEY and L\_NAME, designate the high-level interface with specific formal input/output parameters and the low-level realization, respectively. Any rules with a same L\_KEY are compatible with a parameter in the formal type $R_{\text{L\_KEY}}$. Here, $R_{\text{L\_KEY}}$ is also called an abstract rule.

For each module, the relations between its elemental components and their input/output parameters are hard-coded. Hence, a whole system can be symbolically described by all its setting parameters in scripts, where each actual parameter value may be an instance of a knowledge component or a value in a primitive data type. However, the number of setting parameters in such a system is not necessarily high since many components and parameters can be fixed in default values.

Such a system has the following features: 1) It allows for reusing and understanding of existing knowledge components; 2) the system may be locally improved by preferring those nondominated components with simplicity and efficiency, and 3) some knowledge components can be incrementally embedded into the system by using certain macro modes.

B. Multiagent Framework

The multiagent framework consists of a group of $N_{\text{FA}}$ autonomous agents. Each agent has a compact solving ability. The agents cooperatively search in an ENV [47], [48] for achieving a common intention of finding high-quality solution(s), based on the internal representation (IR) [48] of an optimization task, and related world knowledge [50].

The framework runs in iterative learning cycles. By running as a Markov chain process, the system behavior in the $t$th cycle only depends on the system status in the $(t-1)$th cycle. For convenience, during a run, $t = 0$ and $t > 0$ are called the initialization and the runtime stages, respectively.

Moreover, a chunk is called a runtime chunk if it is long-term stored, retrieved and updated during the runtime. Each runtime chunk aggregates certain particularities of potential solutions [52] of IR [48]. All rules and modules that are applied on the runtime chunks are also called behaviors.

1) Problem Representation: The IR [48] of an optimization task encapsulates certain basic knowledge of the task, which will be further processed for making solution(s) available [50]. Specifically, IR consists of a landscape for providing global structural information and certain auxiliary knowledge components associated with local structural information.

The landscape paradigm has been used for search in general [53]. Formally, a landscape is represented by a tuple, i.e., ($S_R$, $R_M$). $S_R$ is called the representation space, containing all potential solutions to be searched, where each is called a state ($\vec{x}$). The quality-measuring rule ($R_M$) measures which one has a better quality between any two valid states in the $S_R$.

For a normal task, there is a cost function $f_C(\vec{x})$, which is to be minimized, for every state $\vec{x} \in S_R$. Then, the quality can be measured by a simple $R_M$ rule [48], where $\forall \vec{x}_a, \vec{x}_b \in S_R$, if there is $f_C(\vec{x}_a) \leq f_C(\vec{x}_b)$, then $\vec{x}_a$ has a better quality than $\vec{x}_b$, and $R_M$ returns TRUE; else, it returns FALSE.

2) Agent: Each agent is a socially situated autonomous entity [42], [54]. Here, autonomy [54] indicates that an entity is able to control its own declarative and procedural knowledge components. Moreover, each agent is capable of generating new promising states, subjects to the limitations of available knowledge.

Each agent has several basic characteristics, as follows. First, each agent possesses a personal long-term memory (LTM), called $M_A$. The memory stores some private runtime chunks. For an entity, each chunk in its LTM is persistent during its lifetime unless the memory is modified by itself. Most real-world animals possess LTM. The personal LTM supports individual learning [43], [44], which allows each entity to acquire knowledge during its lifetime by utilizing its past experience. In a group, individual learning allows agents to explore, in parallel, novel experiences in a large diversity.

Second, each agent refers to runtime chunks in public knowledge ($M_S$), which is organized in an ENV from the outside world of the agent [46]. Many species, including human beings [55], have evolved the capability for enhancing their group fitness [45] through social learning [56], [57], i.e., obtaining adaptive knowledge from their communities. Social learning also has its ecological significance for circumventing the inhibitory effects of individual learning [44].

Third, each agent can affect the outside world [46], which is realized by simply sharing a part of its knowledge to the ENV, i.e., at least one of the chunks in its $M_A$ is nonprivate [45] and can be accessed by the outside world in the read-only mode.

Finally, the law of behavior is SBIL [43], [44] or cultural learning [58], i.e., a mix of reinforced practices of lifetime experiences and available public information [45], which has been adopted by many species for adaptive foraging [44]. As one of the fast-and-frugal heuristics in ecological rationality [41], it presents the following: 1) gains most of the advantages of both individual and social learnings [56] and 2) boosts collective properties by allowing adaptive experiences to be accumulated over many generations [56].
The solving capability is owned by a rudimentary central executive (CE) [48], [59] module. Specifically, CE performs a generate-and-test behavior [42], [60] for searching promising states, which contains a generating part, a testing part, and a solution-extracting part, under the support of a buffer memory, called $M_G$, which will be cleaned at the end of each cycle. Under the law of SBIL, the generating part creates new chunks and stores it into $M_G$ by using the knowledge in both $M_A$ and $M_S$. Afterward, the testing part updates $M_A$ by using the chunks in $M_G$. The solution-extracting part just extracts all state(s) in $M_G$, which has no influence on the efficiency of the problem-solving process. Moreover, it is assumed that the generating part performs a rather deliberative behavior, while the testing part only produces a rather reactive behavior.

3) ENV: The ENV [46], [47] plays significant roles for supporting the solving capability of all agents.

First, ENV maintains all available world knowledge [50] and physical supports [47]. For example, it keeps a central clock that helps synchronize the behaviors, if necessary.

Second, the ENV serves as a task-dependent domain, namely: 1) It holds the IR of the optimization task, which encapsulates all problem-dependent knowledge, and 2) it recognizes candidate solution(s) for the task, which is simply realized by storing a quasi-solution state ($\vec{x}$), with the best quality among all promising states generated by agents for each single-objective optimization problem.

Finally, ENV also manages resources and services [47] for all agents according to their protocols. First, it provides the initial chunks in $M_A$ of all agents by a memory-initialization module so that each agent itself can be designed in a simple way. Second, it organizes the corresponding $M_S$ for every agent according to all available public knowledge [45], [61] by an interactive center (IC), through a protocol with all agents.

4) Interaction: Here, each agent may only directly interact with IC, in which the interaction protocol between them is based on an interface of two read-only information flows, i.e., 1) the nonprivate knowledge of the agent shared to IC, and 2) the corresponding $M_S$ of the agent referred from IC. In principle, each agent and IC can be designed independently once the corresponding protocol is defined. It also implies that agents may be realized in a heterogeneous way, while the interaction protocol should be defined in a way that the realization of IC is not too complicated, particularly as there are multiple agents.

During the runtime, IC uses the nonprivate knowledge shared by all agents to update its private runtime chunks, and such knowledge will be organized as the $M_S$ of each agent. Thus, agents indirectly interact with their peers through the medium role of IC. In fact, IC may act as a social memory [62], [63], where agents can share their past experiences for achieving emergent behavior by the self-organization of them [39].

C. Simplified System

Fig. 1 shows a simplified MAOS, which includes one of the agents and the ENV it roams. The agents are homogeneous in the sense that they have the same internal structure, which means that all agents are interacted with the ENV in the same way.

Given an IR, the chunks in three memories, including $M_A$, $M_G$, and $M_S$, and, afterward, the nonprivate chunk(s) in $M_A$, are specified in advance. Then, other components of MAOS may be implemented in a rather independent way.

1) Basic Knowledge Components: As an extremely limited version, both $M_A$ and $M_G$ contain only one chunk for holding the states $\vec{x}_A$ and $\vec{x}_G$, respectively, and $M_S$ contains only one chunk, i.e., the state set $X_S$, which contains a set of states in $S_R$. Then, $\vec{x}_A$ is exactly the only nonprivate knowledge to be accessed by IC.

The IC is designed in a centralized way [48]. It summarizes a state set, i.e., $X_S = \{\vec{x}_{A(i)} | i \in [1, N_P]_2\}$, where each $\vec{x}_{A(i)}$ is referred from the $M_A$ of the $i$th agent, by a basic coordinating rule ($R^{CO}_G$), and then, the same $X_S$ is shared by all the agents.

The memory-initialization module simply generates totally $N_P$ states in the $M_A$ of agents. Here, it is reduced to an initial search rule ($R_{IS}$), which generates each state independently.

The solution-extracting part is reduced as a trivial rule, which simply exports each $\vec{x}_G$ directly.

The generating rule ($R_G$) is the major search rule during $t > 0$. It generates $\vec{x}_G$ by using two input sources, i.e., $\vec{x}_A$ and $X_S$.

The testing rule ($R_T$) simply replaces $\vec{x}_A$ by the generated $\vec{x}_G$ once a specific criterion is satisfied. Due to its simplicity, the $R_T$ may determine nontrivial properties of the $\vec{x}_A^{(t)}$ [48].

There are two simple $R_T$ versions: 1) the direct $R_T(R^D_G)$, which does the replacement directly, and b) the better $R_T(R^E_G)$, which does the replacement as $R_M(\vec{x}_G^{(t)}, \vec{x}_A^{(t)}) \equiv TRUE$. Hence, $\vec{x}_A^{(t)}$ stores the most recent and the best states ever obtained by an agent as $R^D_G$ and $R^E_G$ are used, respectively.

The usage of search rules is a direct way of obtaining potential solutions. From a narrow perspective, a search rule is defined as a rule outputting a potential new state in $S_R$.

Both $R_{IS}$ and $R_G$ can be considered as search rules working during the runtime, respectively. However, $R_T$ is not a search rule because it does not produce any new state.

The $R_G$ rule may be decomposed into a filter and a search rule, where the filter may serve as a simple knowledge lens [64] for the purpose of organizing the desired declarative knowledge while suppressing the irrelevant.

A macro mode is called generic if it can embed with some kernel and auxiliary rules, and the type of such a macro rule is the intersection of the types of all kernel rules.
As a macro search rule, each kernel rule is a search rule. With macro search modes, we may only concern with those basic search rules. Then, various advanced macro search rules can be developed by applying different macro modes multiple times to embed existing search rules into them.

2) Execution Process: The simplified MAOS version is executed as follows.

At the initialization stage \((t = 0)\), the states in \(M_A\) of all agents are to be supplied by the \(R_{IS}\) rule in the ENV, i.e.,

\[
\text{IR} \overset{R_{IS}}{\rightarrow} \left\{ \vec{x}_{A(i)}^{(0)} | i \in [1, N_P] \right\}
\]

where \(\vec{x}_{A(i)}^{(0)}\) is the \(\vec{x}_A\) of the \(i\)th agent at \(t = 0\).

During the runtime stage \((t > 0)\), each cycle contains two sequential clock steps \([48]\), i.e., C_PRE and C_RUN steps.

At the C_PRE step, the ENV itself is activated for executing some preparation operations by its components, if necessary. Specifically, the \(R_{IS}\) in IC is executed to organize the \(\vec{x}_S^{(0)}\).

Then, at the C_RUN step, every agent is activated once for executing its rules. For the \(i\)th agent, it performs that

\[
\vec{x}_A^{(t)}(i) \overset{X_S}{\rightarrow} \vec{x}_G^{(t)}(i) \overset{R_G}{\rightarrow} \vec{x}_M^{(t+1)}(i)
\]

For each exported \(\vec{x}_G\), if there is \(R_{IS}^O(\vec{x}_G, \vec{x}_M) \equiv \text{TRUE}\), then the \(\vec{x}_G\) is recognized as the quasi-solution state, i.e., \(\vec{x}_M\).

The system is terminated if an optimal solution is found or if a specific overall cutoff criterion \(R_{CCCO}\) is satisfied.

In summary, the simplified version has one overall setting parameter, i.e., \(N_P\), and four rules, i.e., \(R_{IS}, R_G, R_T\), and \(R_{CCCO}\). Each rule may also have certain setting parameters. Moreover, some rules may be decomposed into suitable macro forms in order to utilize some existing knowledge components.

D. Related Works and Advantage of MAOS

Here, we compare the current MAOS to some related works. Then, we have a discussion on the advantage of MAOS.

1) Previous MAOS Versions: The current framework is developed based on both the compact \([38]\) and extended \([48]\) versions. Since the compact version, a multiagent framework with a rudimental symbolic representation is formed. In the extended version, there are four changes: 1) IR is introduced for general search, including TSP; 2) clock steps are used for synchronizing behaviors in each clock; 3) a CE is taken into account for situating the generate-and-test behavior during the runtime; and 4) a fixed memory specification is applied for avoiding the complex management on declarative knowledge elements, as well as supporting an efficient search strategy.

In the current framework, there are four refinements: 1) The symbolic representation is associated with type system for easy reusing and understanding of the knowledge components; 2) a memory-initialization module is formally used, since the initialization may also have a nontrivial effect on the solving process; 3) an IC is formally introduced for managing the public information so that IC and agents can be flexibly designed according to an interaction protocol between them; and 4) a buffer memory \((M_G)\) is defined for formally separating the three parts of the generate-and-test behavior.

In the simplified version, \(M_G\) only contains one state; thus, the testing behavior is simpler than both the previous versions, and the generating behavior itself simply becomes a search rule. Moreover, using generic macro search rules to embed some basic search rule is particularly stressed.

2) Population-Based Algorithms: GA \([20]\), ACO \([31], [32]\), and PSO \([33], [34]\) are three major examples.

GA simulates the evolution process. Each individual is a basic entity only containing a chromosome, which is not operated by the individual itself. Apparently, a population of these chromosomes forms the public information. Then, “the invisible hand” manipulates the population by using some evolutionary operators, such as selection, crossover, mutation, inheritance, etc. Instead, MAOS simulates the cultural learning process by a group of autonomous agents. Moreover, the simplified MAOS may directly use a rich pool of existing search operators proposed by GA researchers, if necessary.

ACO simulates the stigmergy mechanism in ant colonies. Each ant can be seen as an agent with simple behaviors; however, it differs from the agent in MAOS in that it does not possess a personal declarative memory. The pheromone trails laid down by the ants serve as the public information and can be implemented in the IC of MAOS with a suitable protocol. Without memory, an ant may only perform reflex behaviors. It may be interesting to know if any efficient strategies may be introduced once each ant possesses its personal memory.

PSO is a swarm-intelligence-based method. It has been implemented into the compact MAOS version \([38]\), in which each particle is an agent. Each particle uses three chunks in its personal memory, and these chunks are associated with the search behavior based on a social–psychological principle. However, PSO is not supported in the simplified MAOS since it is still difficult to find competitive PSO variants for solving TSP.

3) Advantage of MAOS for Cooperative Search: As a simple multiagent system, MAOS may possess the potentials of parallelism, robustness, scalability, etc. \([65]\). Moreover, it is a bottom–up way to study social strategies \([66]\). However, here, we are only concerned with the advantage of MAOS in facilitating cooperative searches as compared with existing metaheuristics in adaptive memory programming (AMP) \([52]\).

Cooperative search \([67]–[69]\) consists of a search performed by multiple algorithms which share certain chunks during the runtime. With cooperation, statistical correlations among the performance of the individual algorithms will be introduced, which helps the cooperative portfolio, particularly as the offline performance of the individual algorithms are in negative correlations \([67]\), to achieve better results than when the individual algorithms are independent of each other \([67]\).

Many existing metaheuristics, including ILS, ACO, GA, etc., can be unified under the name of AMP \([52]\) or a more sophisticated version called multiagent metaheuristic architecture (MAGMA) \([68]\). It should be mentioned that an agent in MAGMA could be actually reduced to a search behavior since it does not possesses a personal memory. With a symbolic representation, AMP may be seen as rule-based heuristics. However, it is difficult to represent MAOS in AMP since AMP lacks a way of distinguishing the owners of runtime chunks.
In AMP/MAGMA, all the runtime chunks are stored as public information, and they have the primary importance to all search behaviors. It means that AMP/MAGMA only supports the cooperative search of its behaviors on public information. Thus, it should be very careful in adding/removing a behavior, since the change may have a significant impact on the whole system.

In MAOS, for each agent with autonomy, the runtime chunks from its personal memory and from the public information in IC have the primary and secondary importance, respectively. Thus, cooperative search may emerge in two different levels, i.e., the agent and the group levels.

For the agent level, the cooperation can be achieved if an agent deploys several alternate search rules, particularly in a macro realization of the generating part, during the runtime. From a perspective of multiple sequential learning cycles, the individual algorithms, i.e., the alternate rules, cooperate with others on the $M_A$ of each agent in an interleaving way.

For the cooperation, the sharing information in the personal memory of an agent has the primary importance for the agent. For an agent in MAOS, the alternative search rules may be embedded independently for identifying, in advance, if their offline performances are in negative correlations. Moreover, the testing part may prevent certain high-risk results into its personal memory. Furthermore, the risk will be isolated in the agent only if an unexpected result occurs.

For the group level, the cooperation is applied on the group of agents. Each agent itself can be seen as an individual algorithm, and the agents cooperate with their peers only on public information in IC in a parallel way.

For the cooperation, the sharing information in IC has the secondary importance to each agent in the group. Thus, it does not have a great impact on the whole group, even as to add/remove heterogeneous agents. While the agents explore in parallel, the search process may be accelerated by their cooperation.

The group level is an inherent mechanism within MAOS. However, the agent level requires an agent using macro modules explicitly, as in the simplified MAOS version, which may be easily supported by the symbolic representation.

### III. TSP Domain Knowledge

The implementation of MAOS depends on the problem domain knowledge, which is described as follows. First, the IR of TSP is defined. Second, some auxiliary data structures, which are eligible for improving the search efficiency, are introduced. Third, some related search rules are described.

#### A. IR

For TSP [2], there is a graph with $V$ nodes (or cities) and a cost matrix $D_O = (d_{jk})$, in which $d_{jk}$ is the length of an edge that connects between cities $j$ and $k$, $j, k \in [1, V]$. Here, we are only concerned with the symmetric TSP, which has $d_{jk} = d_{kj}$ for the edges. The objective is to find a minimal-cost Hamiltonian tour, which passes through each node once and only once.

The representation space $(S_R)$ contains all possible states, where each state $\bar{x}$ represents a tour, which is a permutation $(x(1), x(2), \ldots, x(V))$ of the integer values from 1 through $V$. The cost function $f_C(\bar{x})$ to be minimized is represented as

$$f_C(\bar{x}) = d_{x(V), x(1)} + \sum_{j=1}^{V-1} d_{x(j), x(j+1)}.$$  

Normally, the distance metric is defined as the number of different edges between any two tours [70].

The cost matrix $D_O$ itself is the sole source associated with local structure information. For example, it can be used for building some candidate sets and for determining the relative cost changes for certain local moves, which are performed on valid tours, intermediate reference structures (REFs), etc.

#### B. Auxiliary Data Structures

Auxiliary data structures, e.g., reference structures (REFs) and candidate sets, are eligible for improving search efficiency.

1) REF: Some search strategies may use infeasible intermediate REFs, which are strongly associated with local structural information. For TSP, each REF may contain one or more disjointed graph elements in certain patterns, such as tour segment [25], [27], subcycle [11], [15], [28], stem-and-cycle (S&C) structure [71], etc.

Each REF can be modified into a valid tour by suitable reactive operations. Some graph elements can be preprocessed into simple elements. For example, each S&C structure can be transformed into a subcycle or segment. The REF in multiple tours segments (REF$_{SM}$) can be reconnected by randomly [29] or greedily [25], [27] inserting all missing edges. The REF in disjointed subcycles (REF$_{CM}$) can be turned into a valid tour by iteratively joining any two subcycles [15], [28].

2) Candidate Set: For TSP, a complete graph has totally $(V^2 - V)/2$ edges; however, most of the edges will not occur in good tours [2]. Thus, it is reasonable to focus on promising edges. One simple way is to use an edge set, which is called a candidate set ($E_C$) [15], as a static sparse graph for restricting the search.

A simple example of $E_C$ is the nearest neighbor subgraph (NNS) [2], which keeps the nearest $k_{nn}$ ($k_{nn} > 0$) neighbors for each node as candidates, based on the input matrix.

Usually, a minimum 1-tree shares many edges with an optimal tour. The length of the minimum 1-tree is often used as the lower bound [72], since it is a relaxation of TSP. In Lagrangian relaxation [15], [2], there is a penalized matrix $D_\pi = (d_{jk}^\pi)$, where $d_{jk}^\pi = d_{jk} + \pi_j + \pi_k$, by associating each node $j$ with a value $\pi_j$ in a penalty array ($\pi$). Intuitively, the transformed problem with $D_\pi$ may be easily solved if it has a smaller length of the minimal 1-tree, which can be improved by applying a subgradient optimization [72] to $\pi$. However, it might be rather time consuming to obtain a good $\pi$ array [15], which requires one to solve minimum 1-trees many times, and each needs a time complexity $O(V^2)$ by Prim’s algorithm.

Helsgaun have estimated the chances of a given edge being a member of a high-quality tour by using an $\alpha$ matrix [15]. The $\alpha$ matrix can be calculated [15] in a time complexity $O(V^2)$ and
a space complexity $O(V)$ based on a given input matrix, where each edge $(j, k)$ has a corresponding $a_{jk}$ value, which is equal to the minimum length as a 1-tree is required to contain this edge minus the length of a minimal 1-tree. Thus, it is intuitive that an edge is more promising to belong to an optimal tour if it has a smaller $\alpha$ value [15].

Two $\alpha$ matrices, i.e., $D_\alpha$ and $D_{\alpha\kappa\pi}$, are calculated by using $D_\alpha$ and $D_\kappa$ as the inputs, respectively. For example, the $E_C$ in the LKH [15] uses $D_{\alpha\kappa\pi}$ as its input matrix.

C. Types of Search Rules

Some types of search rules, which are frequency-used in existing algorithms, are defined based on the differences in their input parameters and internal features.

Scratch search rule ($R_{SS}$) generates a new state from scratch. Hence, it actually has an equivalent type with the $R_{TS}$ rule.

As a child type of $R_{SS}$, construction search rule ($R_{CS}$) simply assigns the value of each part of a state once and only once. Perturbation search rule uses a perturbation search rule as the kernel rule. An example is an Inver-over operator [24].

Incumbent search rule uses one incumbent state as the input and searches its neighborhoods. It has two child types. The first is perturbation search ($R_{PS}$), also called mutation in GA [20] or kick–move in ILS [18]. The second is the LS rule ($R_{LS}$). The difference is that $R_{LS}$ intends to improve the quality, which is capable of exploiting a landscape, while $R_{PS}$ tries to move to a state with a distance away from the current state, which is useful for escaping from local optimum.

The XS rule ($R_{XS}$), which is similar to the crossover in GA [20], uses two parent states. An implicit advantage is that the distance between $\vec{x}_O$ and each parent may be adaptively controlled by the distance of two parents.

Set-guided search rule ($R_{SGS}$) uses two parents; one is a state, and another is a state set. It has an equivalent type with the $R_G$ rule. An example is an Inver-over operator [24].

All the aforementioned types can be easily embedded into the $R_G$ rule with corresponding filters on their input information.

D. Generic Macro Search Rules

The chained macro mode has been used in some existing algorithms. Here, a macro rule contains a chain of two rules, i.e., a kernel and a $R_{LS}$ rule, where a chain means that the input of a latter rule is exactly the output of a former rule.

An apparently, chained with the $R_{LS}$ rule introduces a greedy search feature for the macro rule. A simple way of realizing a high-quality $R_{SS}$ rule is to use an $R_{CS}$ as the kernel rule [30]. In GLS [27], [30], [35], [36], the major search rule is a chained macro search rule by using an $R_{XS}$ rule, which is normally not inherently greedy, as the kernel rule.

It may also increase the search capability if the kernel rule is an incumbent search rule, particularly as the $R_{LS}$ rule does not allow uphill moves. For example, in ILS [18], the major search rule uses a perturbation search rule as the kernel rule.

E. Basic Search Rules

1) Construction Search: The construction search rule ($R_{CS}$) starts with a partial tour and adds the remaining nodes one by one until the tour is complete. Here, two basic requirements are considered.

First, if there are multiple tours to be constructed, the diversity among these constructed tours should be large to provide enough information for search strategies.

The randomized $R_{CS}$ rule ($E_{CS,R}$) is a simple $R_{CS}$ version, which constructs a state $\bar{x}$ in the $S_R$ at random.

Second, each constructed tour should have a small distance to a good tour to contain positive information [15].

The probabilistic $R_{CS}$ rule ($E_{CS,P}$) uses a candidate set ($E_{CS}$) and $D_\kappa$ as its inputs. A probability matrix is initialized as $p_{ij} = \frac{1}{d_{ij}^2}$, $i, j$. Then, each state is constructed by adding a city as the next city based on roulette selection from the remaining cities in ($E_{CS}$) or else the remaining city with the maximum probability once all the cities in ($E_{CS}$) were used.

2) LS: Well-studied LS strategies include $r$-opt variants, such as 2-, 3- [12], 5-opt [13], [15], etc., and multistage variants [10]. Some LS strategies may also use infeasible Refs, e.g., the two-stage moves with infeasible 2- [11], [13] and 3-opt [15], and the ejection chain algorithm [71] uses S&C for generating ejection moves.

As a simple LS, 3-opt systematically examines sequential exchanges (at most, three edges) in a positive gain criterion [12]. The full 3-opt version requires the examination of the $O(V^3)$ exchanges. In this paper, two 3-opt versions are considered. The $R_{LS}$ rule simply adopts a static candidate set ($E_{CS}$) for restricting the moves, and the $R_{XS}$ rule incorporates one more speed-up technique, i.e., don’t look bits [73] associated with each node.

3) XS: The XS rule ($R_{XS}$) generates a state $\bar{x}_O$ by using two parent states, i.e., $\bar{x}_{P1}$ and $\bar{x}_{P2}$.

Each tour is a permutation, as in some other problems, such as flow-shop scheduling [29], quadratic assignment problem [32], etc. Some order-based XS strategies try to preserve the order and position information, such as cycle crossover [23], partially mapped crossover [21], etc. Certain rules, e.g., maximal preservative crossover (MPX) [22], may also use edges as auxiliary information.

For TSP, each tour may serve as a restrict sparse graph [74]. In fact, edge information is more straightforward for guiding the search than order and position information [29].

Edge-based XS strategies try to utilize the positive edge information. For convenience, an edge is called external if it is not contained in the union edge set of both parents ($E_U$). One basic idea is that any edge in ($E_U$) can be freely introduced as the edge of $\bar{x}_O$. An extreme example is edge exchange crossover (EXX) [26], which does not add any external edges. Another basic idea is that good external edges should be also taken in account since ($E_U$) does not necessarily contain all the edges belonging to, at least, an optimal tour, even as ($E_U$) comes from many high-quality states [74]. Certainly, the quality of $\bar{x}_O$ may be rather low if the probability of introducing unpromising external edges is high.

Many edge-based XS strategies work in two sequential steps: 1) selects a subset of the edges in ($E_U$) into a REF, and 2) modifies the REF into $\bar{x}_O$. Only the second step may introduce certain external edges into $\bar{x}_O$. 
Most existing strategies use the REF in multiple tour segments (REFSM), although they may work under different classes, such as intersection, union, partition, etc.

The intersection class directly uses the intersection edge sets of both parents as the REF. A famous example is distance-preserving crossover [27].

The union class uses the union edge sets of parents for constructing tour segments. In edge recombination [29], tour segments are constructed by considering node degrees.

The partition class is realized as follows. First, the nodes are partitioned into different groups. For each parent tour, each group is associated with a subgraph with corresponding edges connecting the nodes within the group. Second, the REF is formed by a union set of all groups from different parents. A typical example is natural crossover [25].

A few edge-based XS strategies utilize the intermediated REF in disjointed subcycles (REFCM). Most of them belong to the family of the EAX [28].

The probability of introducing low-quality edges by joining subcycles in REFCM is much lower than that by connecting segments in REFSM, as both kinds are realized in greedy mechanisms, since the former has more choices. Joining may be applied on all nodes in subcycles, while connecting can only happen on the two end nodes of each segment.

IV. IMPLEMENTATION

For convenience, certain template(s) may be specified in advance, and certain basic MAOS case(s) can be defined by implementing all rules; then, all related cases can be described by applying local modification(s) on the basic cases.

Two simple improving heuristics are taken into account, based on a generalized edge assembly recombination (GEAX) for enhancing the overall search performance.

A. GEAX

The GEAX is a generalization of EAX [28], [37], which is realized as follows.

Step 1) Design an auxiliary set, called AB-Set, where each is called an AB-cycle (EAB) [28], which is an even-length cycle with A-Edges (EA) and B-Edges (EB), by tracing different edges of xP1 and xP2 alternately. It is realized in two substeps.

Step 1.1) Build the AB-library by using xP1 and xP2. As each AB-Cycle is identified, it is stored into the AB-Library, and the edges that compose it are no longer used [28]. This procedure is repeated until no AB-Cycle could be extracted or the number of AB-Cycles has achieved an upper value (NABL).

Step 1.2) Design the AB-set by an AB-selecting rule (RSAB), which selects some AB-Cycles from the AB-Library.

Step 2) Generate multiple candidate states by using both the xP1 and the AB-Set, where each AB-Cycle in the AB-Set is used for guiding the kick–move on xP1 in two substeps [28].

Step 2.1) Kick the xP1 into a REFCM structure, called E_O, which contains disjointed subcycles covering all nodes, by using each AB-Cycle. First, xP1 is copied as the initial E_O. Then, all A and B-Edges in the AB-Cycle are removed from and added to the E_O. Thus, all the edges introduced in the E_O, i.e., the B-Edges, are selected from the E_P2, based on the definition of the AB-Cycle.

Step 2.2) Modify the REFCM structure into a valid tour. It is achieved by joining two subcycles iteratively, which is accelerated by a candidate set (ECXS). While there are multiple subcycles, the subcycle with the minimal number of edges is selected as the base one, and then, a greedy two-exchange with maximum local gain is used for merging it with any other one, where an edge is systematically selected in the ECXS.

Step 3) Choose one state among the candidate states as the xO by a state-competing rule, which is simply realized by associating each candidate state x with the competing function (fcomp) to be minimized [48], i.e.,

\[
\text{f}_{\text{comp}}(x_c) = \frac{f_C(x_c) - f_C(x_{p1})}{\text{DIS}(x_{p1}, x_c)C_D}
\]

where DIS(xp1, xc) reflects the distance between xp1 and xc, and C_D is the diffusion coefficient, which may determine the velocity of information diffusion [48]. The DIS(xp1, xc) function is approximately estimated as [EABL]/2, i.e., a half of the number of edges in the corresponding AB-Cycle [37].

The fcomp integrates both the high-quality criterion [75] and the distance requirement [18], [27], [70]. If C_D = 0, it represents the high-quality criterion only, as in EAX-1AB [28] and some of its variants [75], [76]. As C_D < 0, it also prefers high-quality states that are far away from the incumbent state, as in the fitness-distance-based diversification [18]. As C_D > 0, it prefers high-quality states closer to the incumbent state. If C_D = 1, it represents the criterion employed in EAX-DIS [37]. In [48], C_D = 1.5 is used for slowing down information diffusion.

The inherent greedy feature is achieved by two aspects. First, xp1 is kicked into REFCM by introducing the edges in xP2 (Step 2.1), while the REFCM can be turned into a valid tour with a few promising external edges (Step 2.2). Second, there is an iterative child generation (ICG) [18], [75] (Step 2), i.e., the competition among multiple independent candidate trials. Such a simple competing algorithm portfolio may reduce the heavy-tailed distributions of individual trials [14], [67]. In fact, ICG has shown its efficiency in ILS [18], and it may perform better than 2-opt as it works with EAX [75].

In summary, GEAX has four variable parts, i.e., C_D, NABL, RSAB, and ECXS, where NABL is fixed as 100 in this paper.

The random GEAX (GEAX.R) uses the random RSAB rule (R^R_{SAB}), where totally N_MT AB-Cycles are randomly
chosen from the AB-Library and any repeated AB-Cycles are discarded.

Some existing EAX variants can be represented by GEAX cases. For example, EAX-Dis [37] is the GEAX.R case with $C_D = 1$ if the potential difference in $E_{C, XS}$ is not considered.

### B. Simple Improving Heuristics

Under limited time and resources, animals make inferences about specific domains in the real world with simple heuristics by utilizing certain imprecise knowledge [41].

1) *Sorting-Based AB-Cycle Selection:* The first heuristic is applied into the process of ICG [18], [75] so as to utilize imprecise positive clues before generating candidate states.

For GEAX, some AB-Cycles are obtained in Step 1.1, where each AB-Cycle leads to a candidate state. However, the random $R_{SAB}$ rule does not utilize any information available at the stage.

Each AB-Cycle $E_{AB}$ is associated with the function $f_{	ext{clue}}$

$$f_{	ext{clue}}(E_{AB}) = \sum \text{len}(E_B) - \sum \text{len}(E_A)$$  \hspace{1cm} (5)

where $\text{len}(E)$ represents the sum of length of all edges in $E$. Hence, the $f_{\text{clue}}$ value is the delta length of total edges between $s_{P1}$ and the intermediate $\text{REFCM}$ structure to be generated by the $E_{AB}$ at Step 2.1. Intuitively, an AB-Cycle with a smaller $f_{\text{clue}}$ value may carry an imprecise positive clue, although some new edges will be included later to modify the $\text{REFCM}$ into a valid tour at Step 2.2.

The sorting $R_{SAB}$ rule $(R_{SAB}^S)$ is proposed at Step 1.2, where totally $N_{MS}$ AB-Cycles in the AB-Library with the minimal $f_{\text{clue}}$ values are selected as the members of AB-Set. Certainly, if $N_{MS}$ is larger than the size of the AB-Library, then all the elements in the AB-Library become the members of the AB-Set.

The sorting GEAX (GEAX.S) simply differs from GEAX.R in that the former uses $R_{SAB}^S$ instead of $R_{SAB}^G$ at Step 1.2) of GEAX.

2) *Switch Macro Mode:* The second heuristic is the generic macro search rule in switch mode. The switch macro mode takes multiple kernel rules in the same $I_{KEY}$ as the alternate rules, which may have different $I_{NAME}$ tags, and only runs one of them, which is selected by a deploying rule ($R_{DEP}$) [42], i.e., a simple task-switching schedule [77], at each activation.

If $R_G$ or at least one of its components (e.g., $R_{XS}$) is realized in the switch mode, then agents may run alternative $R_G$ rules at each cycle. For each agent, all the alternate $R_G$ rules cooperate, at least, by its personal knowledge during multiple cycles. Thus, the switch macro mode supports the cooperative search of multiple alternative search rules in the agent level.

### C. Standard Template

The standard MAOS template, which employs $N_P$ compact agents, has the following specific components.

The $R_{IS}$ rule is simply a macro scratch search rule in chained mode. It is organized as a tuple $(R_{CS}, R_{LS})$, i.e., a chain of a construction search rule $(R_{CS})$ and an LS rule $(R_{LS})$.

The $R_G$ rule is decomposed as a tuple $(R_{SP}, R_{XS})$, which contains a state-picking rule $(R_{SP})$ and an XS rule $(R_{XS})$. For the $R_{XS}$ rule, a basic requirement is that it is a greedy strategy. The $R_{XS}$ rule uses two input sources, i.e., $x_{P1}$ and $x_{P2}$, where $x_{P1} = x_A$ and $x_{P2}$ is filtered from $X_S$ by the $R_{SP}$ rule. For the $R_{XS}$ rule, $x_{P1}$ and $x_{P2}$ can be seen as the incumbent state and the guiding information, respectively.

Moreover, the macro search rule in switch mode is applied on the $R_{XS}$ rule. The macro $R_{XS}$ rule contains an $R_{DEP}$ rule and several alternate $R_{XS}$ rules. If there is only one alternate $R_{XS}$ rule, then the macro $R_{XS}$ is reduced to the alternate $R_{XS}$.

Organized by simple components with specific roles, the MAOS template tends to tailor to the “big valley” [70] in the TSP landscape, which suggests that better local minima tend to have a smaller distance to the closest optimum by sharing common partial structures [17], [19], i.e., edges or segments.

At the initialization stage, the $R_{CS}$ is to construct states with both quality and diversity, and the $R_{LS}$ is to improve the quality of each state for reaching the “big valley.” Then, during the runtime, as the basic intuition behind ILS [16]–[18], agents explore in parallel, where each high-quality state $x_{A(i)}$ may serve as the incumbent state for searching global optimum in its neighborhood with a short distance.

Moreover, the high-quality tours in $X_S$ tend to concentrate on a very small subspace around the optimal tour(s) [32], which may lead to two positive effects: 1) The union of edges in high-quality tours may be seen as a restrict graph. The graph may cover most or even all edges in an optimal tour while it is quite sparse, due to the large number of shared edges among high-quality tours [74]. The sparse restrict graph may serve as a dynamic guiding information for facilitating the search process. It has been shown in ILS [16] that cost-restricted kicks can be more effective than blind kicks, particularly for large instances, and b) the edge set may represent pseudobackbone frequencies [19], although it is NP hard to find the exact backbone [78]. Compared with the pheromone matrix [31], [32], the edge set provides not only the frequency of edges but also the frequency of many tour segments, for belonging to high-quality tours. Through picking the partial information by the $R_{SP}$ rule, it is able to utilize the pseudobackbone frequencies implicitly.

### D. Standard Case

The standard case (#STD) is defined based on the standard template, where the $N_P$ will be specified at each time it is used.

For $R_{IS}$, it is implemented as $(R_{SC}, R_{LIS})$. For $R_T$, $R_{IT}$ is used. For the macro $R_{XS}$ rule, the GEAX.R is employed as the alternate $R_{XS}$ rule, which has $C_D = 1.0$ and $R_{SAB}$ with $N_{MS} = 20$. Basically, GEAX.R is equivalent to EAX-Dis [37] if the difference in $E_{C, XS}$ is not considered.

For $R_{DEP}$, the randomized $R_{DEP}$ rule $(R_{DEP}^R)$ is realized by selecting one of the alternate rules at random [42].

An identical candidate set $(E_{C, STD})$ is employed for restricting the search of all related behavioral rules, i.e., $R_{CS}$, $R_{LCS}$, and $R_{XS}$. Specifically, $E_{C, STD}$ is realized in the NNS mode [2], which has $k_{nn} = 20$ and has $D_0$ as its default input matrix.

For $R_{SP}$, the randomized $R_{SP}$ rule $(R_{SP}^R)$ is considered, which picks one of the states from the given $X_S$ at random.
For $R_{CCO}$, the $R_{MC}$ rule is used, which is terminated if the maximum number of cycles ($T_{MAX}$) is achieved or if no further improvement on the solution quality occurs for $T_{CON}$ cycles. Here, $T_{MAX}$ and $T_{CON}$ are fixed as 500 and 100, respectively.

For each tour, the most straightforward Array representation [79] is used by both the $R_{CS}$ and $R_{LS}$ rules, while the doubly linked list representation [79], which has a similar time complexity with the Array representation, is used by GEAX since it is convenient for representing the REF$_{CM}$ structure.

V. EXPERIMENTAL RESULTS AND DISCUSSION

The MAOS is coded in JAVA, and was run by JRockit JVM 1.5 on a 1.5-GHz Itanium 2 processor with 1-G memory.

Two benchmark sets are used; the first is all 21 instances with $1000 \leq V < 3000$ nodes in the TSP library (TSPLIB) [80], and the second is all 22 instances with $3000 \leq V < 6000$ in the VLSI data set [7]. Using a set of instances with nodes in a range may bring a lesser bias on the average performance than by using some selected instances. For each instance, ten independent runs ($N_R$) were performed.

There are three indices for the performance of an algorithm. The first is the success rate ($p_s$), which is estimated by $N_S/N_R$, where $N_S$ is the number of times optimal states were found. The second is the relative percentage deviation (RPD) above the best known solution ($f^*$). The third is the running time ($t_r$) in seconds, which may be influenced by different machine configurations. Both $p_s$ and RPD are related to the solution quality, although they are based on different viewpoints.

A. Contributions of Components and Parameters

The contributions of the MAOS components are evaluated on the TSPLIB instances by using RPD–Time diagrams for indicating the Pareto efficiency of both performance indices. For each case, $N_P = 10, 30, 50, 100$, and 300 are tested.

Fig. 2 shows the results by #STD using different $R_T$ and $R_{LS}$ components. For $R_T$, #STD/RT.D simply uses $R_T^D$ instead of $R_T^T$, which removes the quality control. It produces similar performance with #STD, which means that the employed $R_{XS}$ has an inherent greedy feature in a strong sense.

For initialization, the #STD/CS.RND uses $R_{CS}^R$ instead of $R_{CS}^P$, and #STD/LS.3OC uses $R_{LS}^{3OC}$ instead of $R_{LS}^{3OS}$. In fact, the tuple $(R_{CS}^R, R_{LS}^{3OC})$ has been used in [48] for initialization, where $R_{LS}^{3OC}$ has shown its advantage as compared with the 2-opt [48]. Fig. 2 shows that #STD achieves highly dominating performance than both #STD/LS.3OC and #STD/CS.RND.

Table I lists the average information at $t = 0$ for all three cases under $N_P = 10$, 100, and 1000, where $R_{DP}$ and $t_{r0}$ are RPD and $t_r$ at $t = 0$, respectively, and $r_{t0} = t_{r0}/t_r$. It can be seen that #STD achieves much lower $r_{t0}$ than both of the others. It reduces both $R_{DP}$ and $t_{r0}$ significantly by using $R_{CS}^P$ instead of $R_{CS}^R$, which indicates the high advantage of using suitable $R_{CS}$ heuristics, even with simple input information. With $R_{LS}^{3OS}$, which simply incorporates don’t look bits [73] into $R_{LS}^{3OC}$, the initialization requires much less $t_{r0}$, although it also produces worse $R_{DP}$. Certainly, it still leaves a large space for improving the $R_{LS}$ since the initialization costs more than 32% of the total running time. For investigating the performance of $R_{XS}$, it is expected that the $R_{LS}$ to be introduced may reduce $r_{t0}$, while it is not too sophisticated.

Here, the $t_{r0}$ is mainly consumed by the $R_{LS}$. The Array representation may be replaced by the two-level tree [79], which may drop the time complexity of per Flip from $O(V)$ to $O(\sqrt{V})$, while having no impact on its outputs. It has also been declared that the “segment list” version [30] may make it twofold faster than the conventional two-level tree. Moreover, more sophisticated $R_{LS}$ may be considered, concerning the overall performance. For example, 5-opt [13] may achieve better performance than 3-opt, which has been the basic move in LH [15] and PHGA [30]. The chained LK (CLK) [16], which has been used in some algorithms [36], may also be considered.

Figs. 3 and 4 show the results by using different $N_{MT}$ values for $R_{SAB}$ and different $C_D$ values for GEAX.R, respectively.

Fig. 3 shows that the performance of #STD is quite stable for $N_{MT}$ varied from 10 to 50 and only becomes worse as $N_{MT}$ is smaller, such as $N_{MT} = 5, 3$, and 1. Hence, the choice of $N_{MT}$ is quite flexible. Moreover, it also suggests the advantage of ICG [18], [75].

Fig. 4 shows that varying $C_D$ from $-1.0$ to 2.0 does not introduce a great difference, although the case with $C_D = -1.0$ seems slightly worse, which may be due to two possible reasons. The first is that the multiagent framework may facilitate the preservation of the diversity of positive clues in public information [58], as explored by the agents in parallel, where each of them possesses a private memory and only picks a piece of public information for biasing its search. The second is that the GEAX worked in the biased style may keep a local diffusion effect, since most AB-Cycles are in small sizes and some edges may be brought back by the greedy joining mechanism. Aside from that, the biasing rules are used in both $R_T$ and $R_{LS}$ to produce different random sequences. The general advantage of #STD for $N_{MT} = 1000$ is confirmed. For $C_D = 2.0$, the performance of #STD is quite stable.
from this, a larger $C_D$ value may be considered for achieving better performance if a longer running time is available.

Fig. 5 shows the results by using candidate sets ($E_{C:ST}$) in the NNS mode, which are derived under four input matrices, i.e., $D_O(\#STD)$, $D_\alpha(\#STD\alpha)$, $D_\pi(\#STD\pi)$, and $D_{\alpha\&\pi}(\#STD_{\alpha\&\pi})$, respectively. The penalty array ($\vec{\pi}$) for obtaining $D_\pi$ of each TSBLIB instance comes from the LKH 1.3 [15]. It shows that using $D_\alpha$, $D_\pi$, and $D_{\alpha\&\pi}$ as inputs of candidate sets may bring on a better performance than using $D_O$. However, it should be careful to use $\vec{\pi}$ by counting the high time complexity for finding a suitable one [15].

B. Performance Improvements

For further investigating the performance of MAOS cases with improving heuristics, $\#STD\alpha$ is taken as a basic case.

The case $\#STD\alpha|S$ simply uses GEAX.S, which has $N_{MS} = 10$ for the $R_{SAB}$ rule, as the $R_{XS}$ rule to be used.

The case $\#STD\alpha|S&R$ deploys two alternate $R_{XS}$ rules, i.e., the GEAX.R in $\#STD\alpha$ and the GEAX.S in $\#STD\alpha|S$, by the $R_{DEP}$ rule. It is equivalent to a macro $R_G$ rule with two alternate $R_G$ rules, where each uses one of the $R_{XS}$ rules.

Fig. 6 and 7 show the results of four MAOS cases, i.e., $\#STD$, $\#STD\alpha$, $\#STD\alpha|S$, and $\#STD\alpha|S&R$, which are applied on the TSPLIB and VLSI instances, respectively.

First, it shows that the $\#STD\alpha|S$ performs quite well on both RPD and $t_r$ for the TSPLIB instances. Hence, the sorting criterion by $R_{SAB}$ may indeed catch certain positive clues for achieving fewer trials in ICG.

Second, it also shows that $\#STD\alpha|S$ produces worse results for the VLSI data set as $N_P$ is large. Hence, the MAOS cases with GEAX.R and GEAX.S are in negative correlations [67], where each is particularly good at one of the data sets.

Thirdly, it demonstrates the efficiency of $\#STD\alpha|S&R$ on both data sets. The $\#STD\alpha|S&R$ case achieves a similar performance with the better one of $\#STD\alpha$ and $\#STD\alpha|S$. As $N_P$ is larger than 100, the cooperative search by GEAX.R and GEAX.S even produces better performance than the search by GEAX.R or GEAX.S alone for the VLSI data set.

Such a cooperative search may imply a different viewpoint for designing algorithms, since the offline performance of each
alternate rule can be accumulated during the solving process. Instead of seeking for an omnipotent rule for a full set of tasks, which might be extremely difficult, if not impossible, the macro rule for deploying simple alternate rules in negative correlations may be considered. Moreover, only a part of the new alternate rules need to be implemented for covering those new tasks that are difficult for existing alternate rules, as the domain structure features of tasks may change over time.

C. Comparisons With Other Algorithms

It is difficult to compare the running times of various algorithms on different executing conditions, including both machine configurations and programming platforms.

A simple way to roughly compare the difference on various machine configurations is to execute a same benchmark code [1]. Table II summarizes the running times of the C code of a greedy heuristic [1]. The $r_{CPU}$ values mean that the 1.5-GHz Itanium 2 is about 2.82 and 2.72 times faster than the 933-MHz Pentium III [30] and the 500-MHz Alpha [1], respectively.

Moreover, both the JAVA and C benchmark codes of SciMark 2.0 were executed on the 1.5-GHz Itanium 2 by using JRockit 1.5 and GCC 3.3.3 (with −O2 option) and obtained the composite scores of 170.63 and 185.08 (higher is better), respectively. It indicates that the performance gap between JAVA and C is really close. For simplicity, the small gap is not considered in the comparisons.

For the two CPUs with $r_{CPU}$ values in Table II, we compare the normalized running times. For other CPUs compared in this paper, we simply compare the real running times; however, the readers may also achieve a rough normalization since such CPUs are faster than the 933-MHz Pentium III in Table II.

The performance of MAOS on the TSPLIB instances are compared with three algorithms, i.e., LKH [15], IBGLK [19], and PHGA [30]. All the algorithms are coded in C. The times of IBGLK and PHGA were normalized to 500-MHz Alpha [19] and 933-MHz Pentium III [30], respectively, and the time of LKH was measured on a 933-MHz Pentium III [30].

The IBGLK [19] combines iterated LK algorithms with the pseudobackbone frequencies by considering two techniques: 1) construct pseudobackbone by using samples of local minima and 2) alternate between BGLK and regular LK once each of them fails. It has outperformed the three LK versions.

The LKH [15] has been one of the most powerful LK variants, which considers a powerful set of two-stage moves and utilizes $D_\alpha & \pi$ as the input matrix of a candidate set. The results tested in [15] did not count the time for obtaining suitable $\vec{\pi}$ arrays and missed one of the instances, i.e., d2103. Hence, the results tested in [30], which were performed on the UNIX version of LKH 1.3 [15], are considered.

The PHGA [30] is a multipopulation hybrid GA, which uses a variant of the MPX [22]. Some speed-up techniques, such as candidate set, don’t look bits [73], and the two-level tree [79] tour representation, are used. Moreover, PHGA represents tours in the “segment list” version [30] of the two-level tree and uses Helsgaun’s 5-opt [15].

Table III summarizes the performance indices of LKH [30], IBGLK [19], and PHGA [30], for the TSPLIB instances.

Table IV lists the results of #STD, #STD, and #STD, with $N_P = 300, 500, and 1000 for the TSPLIB instances, where the $T_{RUN}$ means the actual running cycles. With $N_P = 1000$, MAOS can achieve $p_s > 0$ for all the instances and $p_s = 100\%$ for 19 instances.

When comparing Table IV with Table III, the MAOS with $N_P = 300$ achieves performance 1.44 times higher for the $p_s$, 6.71 times lower for the RPD, and 2.10 times faster for the normalized running time than that of LKH; the MAOS with
Table IV compares the results of LKH-V10 and #STDα|S&R cases on VLSI data set. For tackling large-scale instances, it may be interesting to incorporate MAOS with some edge-locking approaches [5], [81], which may reduce those instances into smaller ones by locking certain edges, e.g., the common segments in tours [30].

VI. CONCLUSION

The MAOS is organized with a group of agents self-organizing in the ENV, where they indirectly cooperate with

\[ N_P = 500 \] achieves performance 46.92 times lower for the RPD and 16.64 times faster for the normalized running time than that of PHGA.

Table V compares the results of LKH-V10 [7] and the #STDα|S&R with \( N_P = 150, 1000 \) for the instances in VLSI data set, where RPD_B means the best RPD. LKH-V10 [7] is the best of ten runs of LKH, each with 10 iterations, on AMD Athlon 1900+ processor (1.6 GHz) [7]. The MAOS with \( N_P = 1000 \) achieves performance 3.00 times lower for the RPD and 9.21 times faster for the real running time than that of LKH-V10.

Table VI compares the results of the #STDα|S&R with \( N_P = 300 \) and two prominent EAX variants, i.e., EAX-Dis [37] and HeSEA [36]. Only HeSEA [36] integrates an advanced LS strategy, i.e., a CLK [16] with an ICG in family competition. The comparison is not complete since the two EAX variants were only tested on five and seven selected instances for the 21 TSPLIB instances, with \( 1000 \leq V < 3000 \). For EAX-Dis, it is coded in C++ and was run on a 1.7-GHz Xeon processor. For HeSEA, it is coded in C and was executed on a 1.2-GHz Pentium IV processor. For the five selected TSP instances, #STDα|S&R achieves performance 1.08 times higher for the \( p_s \), 4.83 times lower for the RPD, and 5.17 times faster for the real running time than that of EAX-Dis [30]. For the seven selected TSP instances, #STDα|S&R obtains slightly worse \( p_s \) than HeSEA on u1432 and u2152 and the same \( p_s (= 1.00) \) as HeSEA on other five instances, and achieves performance 4.67 times faster than HeSEA for the real running time.

Unlike many state-of-the-art metaheuristics [15], [27], [30], including some EAX variants, such as HeSEA [36], current MAOS versions do not use explicit LS strategies, particularly the LK variants, during the runtime. In fact, The MAOS cases are good at certain TSP instances, while LS strategies may be good at some other TSP instances. For example, LKH is good at u2319, u1432, pr1002, etc., while the MAOS cases are good at si1032, d1291, vm1748, etc.

There are two ways to implement LS strategies into MAOS. The first is to chain \( R_{XS} \) with \( R_{LS} \), as in GLS. The second is to consider \( R_{LS} \) rules as alternate rules in the macro \( R_{XS} \) rule in switch mode, since \( R_{LS} \) can be seen as a special \( R_{XS} \), which uses only one parent as the incumbent state.

Due to the machine memory limitation, the MAOS cases are evaluated on some TSP instances with \( V < 6000 \) only. For tackling large-scale instances, it may be interesting to incorporate MAOS with some edge-locking approaches [5], [81], which may reduce those instances into smaller ones by locking certain edges, e.g., the common segments in tours [30].

VI. CONCLUSION

The MAOS is organized with a group of agents self-organizing in the ENV, where they indirectly cooperate with
others through an IC. In the simplified MAOS version, each agent possesses extremely limited declarative knowledge and simple procedural rules. Moreover, using generic macro search rules to embed with some basic search rule is addressed. MAOS may be particularly suitable for supporting cooperative search.

The implementation of MAOS focuses on utilizing simple and efficient components for tailoring to the problem domain. Moreover, two simple improving heuristics are taken into account based on a GEAX.

The experimental results on both the TSPLIB instances and the VLSI data set demonstrate that MAOS is competitive with some state-of-the-art algorithms, including LKH, PHGA, and IBGLK, in both the solution quality and the running time.

The characteristics of the components and parameters of MAOS are investigated based on the overall performance. The advantage of two simple improving heuristics is also shown. In particular, with the macro rule in switch mode, which deploys alternate rules with offline performance in negative correlations, agents may achieve better performance than with each alternate rule alone. It may prevent the difficulty of finding an omnipotent rule for tackling a full set of tasks.

Future works may be performed on the following topics: 1) search for new alternate rules that are particularly good for some difficult problem instances; 2) study online strategies for adaptive deployment of alternate rules during the runtime; and 3) apply MAOS to other hard computational problems.

### References


