

How Autonomy Oriented Computing (AOC) Tackles a Computationally Hard Optimization Problem

Xiao-Feng Xie

Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
xfxie@comp.hkbu.edu.hk

Jiming Liu

Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
jiming@comp.hkbu.edu.hk

ABSTRACT

The hard computational problems, such as the traveling salesman problem (TSP), are relevant to many tasks of practical interest, which normally can be well formalized but are difficult to solve. This paper presents an extended multiagent optimization system, called MAOS_E, for supporting cooperative problem solving on a virtual landscape and achieving high-quality solution(s) by the self-organization of autonomous entities. The realization of an optimization algorithm then can be described in three parts: a) encode the representation of the problem, which provides the virtual landscape and possible auxiliary knowledge; b) construct the memory elements at the initialization stage; and c) design the *generate-and-test* behavior guided by the law of *socially-biased individual learning*, through tailoring to the domain structure. The implementation is demonstrated on the TSP in details. The extensive experimental results on real-world instances in TSPLIB show its efficiency as comparing to other algorithms.

Categories and Subject Descriptors

I.2.8 [Problem Solving, Control Methods, and Search]: heuristic methods; I.2.11 [Distributed Artificial Intelligence]: multiagent systems

General Terms

Algorithms, Experimentation, Theory

Keywords

Multiagent system, Autonomy Oriented Computing (AOC), Traveling Salesman Problem (TSP), global optimization, search, cooperative problem solving, emergent and collective behavior.

1. INTRODUCTION

The hard computational problems are ubiquitous in scientific and engineering fields [26], which often are conceptually simple and can be cast as a search through a space of alternatives [24][37].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06, May 8–12, 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005...\$5.00.

The landscape paradigm originated in theoretical biology is used for search in general [25]. Suppose the *representation space* (S_R) contains all the potential solutions, where each is called a *state* \vec{s} , and the *solution space* (S_O) is a set of states with reasonable quality. Normally, the S_O/S_R is quite small since that the size of S_R expands exponentially with the size of problems [24]. For the optimization task to find $\vec{s} \in S_O$ with high probability, typical challenges include: a) little *a priori* knowledge is available; and b) total computational resource and time are bounded.

The traveling salesman problem (TSP) [40] is a classic NP-hard problem. Although the TSP is easily formulated, it exhibits various aspects of hard computational problems and has often served as a touchstone for new problem solving methods [29][54]. The researches [10][17][53] have indicated the phase transitions under certain order parameter(s), however, it seems most real-world instances are still located at the hard computational region.

Autonomy oriented computing (AOC)[31] addresses the modeling of autonomy in the entities of a complex system and the self-organization of them in achieving a specific goal. The complex behaviors can emerge from the interactions of autonomous entities following simple rules. Recently, the AOC based methods have been applied to solve various problems [30][47][51].

In this paper, the compact multiagent optimization system (MAOS_C) [51], which is originally proposed for handling with the numerical optimization problem (NOP) only, is extended for supporting the cooperative search on a virtual landscape by a society of compact agents with limited memory capacity and simple behavioral rules under ecological rationality [18]. Each agent only has moderate problem solving capability by comparing with two extremes: a) the reflex agent, such as *ant* [12], which has no declarative memory and can only produce reflex behaviors in the environment; b) the cognitive architecture, such as ACT-R [1], which is rather sophisticated under unbounded rationality. Besides, the cooperation [13] is necessary because no single agent has sufficient knowledge on the virtual landscape. Here the agents collaborate with the others by indirect interactions, which are implemented through the communication medium role of the environment [31], instead of by sophisticated negotiation [27].

The paper is organized as follows. In Section 2, the extended multiagent optimization system (MAOS_E) in two levels is introduced in details. In Section 3, the implementation is demonstrated on the TSP, which addresses exploiting the problem features in search. In Section 4, the extensive experimental results by MAOS_E, using real-world instances from the TSPLIB, are

compared with those of some existing algorithms [45][54]. In the last section, this paper is concluded.

2. THE MULTIAGENT SYSTEM

The extended multiagent optimization system is built in two levels: a) the symbolic level for supporting basic problem solving; and b) the multiagent framework for specifying the organization structure and operation mechanism based on the symbolic level.

2.1 The symbolic level

The general problem solving capability arises from the consequence of the interaction of declarative and procedural knowledge [1]. As in the information process system [37], the declarative knowledge is represented in *symbol structures*, called T_INFO elements, and the procedural knowledge is represented in *elementary information processes*, called behavioral rules.

Each T_INFO element is described by a tuple $\langle I_RC, I_TYPE, I_CON \rangle$, where I_RC is the retrieve cue, I_TYPE indicates a certain search space, and I_CON is its state.

The memory [19][37] is used for storing T_INFO elements. Each element can be retrieved from a memory according to its I_RC.

The memory must be updated if it is to be useful [19]. But updating is not encoding a new T_INFO element. Instead, only the I_CON is subjected to change. Then a T_INFO element can be referred as a trajectory that comprise of many instances. For specified time t , a T_INFO instance is expressed as $I_TYPE_{I_RC}^{(t)}$.

Each behavioral rule is described by a tuple $\langle I_NAME, I_KEY \rangle$, where I_KEY indicates a virtual interface for handling specific T_INFO elements, and I_NAME indicates the real operations, which can be controlled by specifying setting parameters within a rule-parameter space. A behavioral rule is expressed as $R_{I_KEY}^{I_NAME}$.

2.2 The multiagent framework

As an AOC-based model [31], the framework consists of a society of N agents where they cooperate in a sharing environment (E) to realize a common intention of finding high-quality solution for an optimization task. Since the optimization task can be well-defined, the sophisticated negotiation [27] among agents is not necessarily. Instead, an agent is communicated with the others by indirect interactions, which are implicit implemented through the communication medium role of the environment (E).

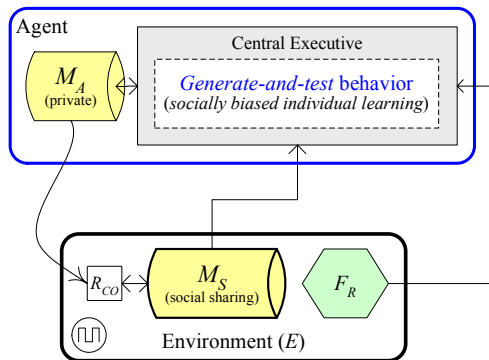


Figure 1. An agent and the environment (E) it roams

Without loss of generality, the agents are homogenous in the sense that they have the same organized structure. Figure 1 shows one of the agents and the environment it roams. All other agents are interacted with the environment in same way.

2.2.1 Environment

The environment serves as the domain in which agents roam [31].

Firstly, it contains an internal representation (F_R) [36][41] of the optimization task, which encapsulates accessible rudimental knowledge, includes a virtual landscape [25] and some auxiliary information associated with problem structure.

For the landscape, a quality evaluation rule (R_M) is used for measuring which one has better quality between any two states (\bar{s}) in the representation space (S_R). The quality evaluation implies the *intention* to attain the state that is better than another.

For normal problems, there is a cost function $f(\bar{s})$ for each state $\bar{s} \in S_R$. Then the evaluation is realized by a $R_M^{\square}(\bar{s}_a, \bar{s}_b)$ rule: for $\forall \bar{s}_a, \bar{s}_b \in S_R$, if $f(\bar{s}_a) \leq f(\bar{s}_b)$, then the \bar{s}_a is better than the \bar{s}_b , and R_M^{\square} returns TRUE, else it returns FLASE.

Secondly, the environment holds a socially-shared memory (M_S), called *collective memory* [38], which serves as the blackboard for all agents by creating a shared past. The information flow from the agents to the M_S is supported by the *coordinating* behavior (R_{CO}) as in Fig. 1. In this sense, the environment also serves as an accumulation pool for the emergent collective behavior instead of a pure information provider as in cognitive architectures [1].

Thirdly, the environment keeps a central clock that helps synchronizing the behaviors of the total system, if necessary [31].

In fact, the environment can be seen as a pseudo autonomous entity for supporting the indirect cooperation of agents.

2.2.2 Agent

Each agent is a socially situated autonomous entity. Autonomy [31] is an attribute of a self-governed, self-directed entity with respect to its own status, free from the explicit control of another entity. The essence is that each entity is able to make decisions for itself, subject to the limitations of the available information.

The central executive (CE), the most important component of working memory model [3][43] in term of its general impact on cognition, is responsible for manipulating the local behavior and behavioral rules of an agent, which govern how it should act or react to the available information on the representation of problem while determine the next status to which the entity will transit.

The agent has two *belief* sources. Firstly, it possesses a private memory, called M_A , which has limited capacity and can only be modified by agent itself. The M_A is a fundamental mental component for supporting individual learning. Secondly, it can access to the M_S , which is socially available from the environment.

The law of behavior is *socially biased individual learning* (SBIL) as adopted by many species in real world [16], which is a mix of reinforced practice of own experience and socially available information. The SBIL is suggested to be a heuristic in ecological rationality [8][18], which a) gains most of the advantages of both individual and social learning; and b) facilitates the emergent and

collective properties by allowing learned knowledge to be accumulated from one generation to the next.

Due to lack of enough knowledge on the landscape to be searched, the essential behavioral rule is *generate-and-test* rule (R_{GT}). The essential components of an R_{GT} rule include a generating part (R_G), a solution-extracting part (R_S) and a testing part (R_T) [51]. The R_S part has no influence on the problem-solving process. It just exports potential solution(s) during a run. Without loss of generality, it can be supposed that only the R_G part can create new information that contains new potential solution(s), and the R_T part only produces simple reflex behavior, which may determine nontrivial properties for some T_INFO elements in memory.

2.2.3 Working process

The framework runs in two phases. The first phase is *memory initialization* phase, which initializes the T_INFO elements in M_S and in the M_A of all agents according to the F_R . The second phase is *generate-and-test* phase. For a run, the number of cycles is T . The behavior of system in the t th ($t \in [1, T]_{\mathbb{Z}}$) cycle only relies on the status of system in the $(t-1)$ th cycle. By running as a Markov chain process, the system can be analyzed in each cycle.

Each cycle contains two clock steps: the C_PRE and the C_RUN.

At the C_PRE step, the coordinating behavior (R_{CO}) at the environmental level, which summarize the information submitted by all the agents into the M_S , is executed, i.e.

$$\{M_{A(i)}^{(t)} \mid i \in [1, N]_{\mathbb{Z}}\}, M_S^{(t)} \xrightarrow{R_{CO}} M_S^{(t+1)} \quad (1)$$

For simplicity, here only the observational information is taken into account, i.e. the T_INFO elements in all agents with same I_RC are collected into a T_INFO element in M_S . It is significant since *observational learning* [5] can lead to cumulative evolution of knowledge that no single agent could invent on its own.

At the C_RUN step, if an agent is activated, where its *generate-and-test* rule is executed for generating new information by estimating the distribution of promising space according to available information in M_A and M_S and updating its M_A , i.e.,

$$M_A^{(t)}, M_S^{(t)} \xrightarrow{R_{GT}} \begin{cases} \{\bar{s}^{(t)}\} \\ M_A^{(t+1)} \end{cases} \quad (2)$$

Naturally, there are two run modes: a) *full-run* mode, where all agents are activated; b) *partial-run* mode, where only N_{ACT} agents ($N_{ACT} \in [1, N]_{\mathbb{Z}}$), which are selected at random, are activated. Under same agent activated times, the system in *partial-run* mode often converges fast since it allows fast information diffusion.

3. IMPLEMENTATION FOR THE TSP

For solving the TSP, the implementation is realized in three steps: a) define the representation of the TSP (F_R); b) initialize the memory; and c) design suitable generate-and-test behavior. According to *no free lunch* theorem [50], the domain knowledge of the problem must be embedded at the implementation stage.

3.1 Representation of the TSP (F_R)

The representation (F_R) provides a basic interface for accessing the essential domain information of the TSP, which includes a

TSP landscape and a candidate set for accelerating the local search (LS), the basis of many good heuristic methods.

3.1.1 The TSP landscape

The TSP landscape is quite straightforward. It is a weighted complete graph with V nodes (or called cities) and a cost matrix $D=(d_{ij})$, where d_{ij} represents the length of an edge that connects between cities i and j ($i, j \in [1, V]_{\mathbb{Z}}$). Here we only concern with the symmetric TSP, which have $d_{ij} = d_{ji}$ for every pair of nodes.

The TSP then is to find a minimal-cost Hamiltonian tour that passes through each node once. Each state \bar{s} is a complete tour, which can be represented as a permutation (i_1, i_2, \dots, i_V) of the integer values from 1 through V , where the cost function is

$$f(\bar{s}) = d_{i_1 i_2} + d_{i_2 i_3} + \dots + d_{i_V i_1} \quad (3)$$

The *distance* of two states is defined as the number of edges in which they differ [15][29][34].

In this paper, the original TSP landscape is used in order to allow a fair evaluation of the performance of algorithms. However, it should be mentioned that the transformation of the cost matrix always be an approach for improving the solving performance.

One kind of methods does not change the relative order of the states. As in Lagrangean relaxation [23][40], the length of every tour is increased by $2 \cdot \sum \pi_i$ as we associate with each node i a penalty value π_i and use the transformed cost matrix $D'=(d'_{ij})$, where $d'_{ij} = d_{ij} + \pi_i + \pi_j$. Of course, the order for the same segments of a tour may be changed under different cost matrix.

The other kind of methods indeed changes the global landscape. As in *search space smoothing* [21], the original landscape is transformed into a series of gradually smoother landscapes, which the smoothest one is solved at first and the solutions are used for guiding the following landscapes until the original one.

3.1.2 Candidate set (E_C) for local search

Local search (LS), also known as neighborhood search, has been shown very effective for hard computational problems, including the TSP. A basic LS algorithm, under a predefined candidate set of neighborhood description, starts from an arbitrary complete tour and repeatedly improves the current tour till it is trapped into a local minimum. 2-Opt, 3-Opt and Lin-Kernighan (LK) algorithm [29] are representative LS methods.

The complete graph for a symmetric TSP has $(V^2-V)/2$ edges. However, it is intuitively that most of the possible edges will not occur in good tours because they are simply too long [40]. It is therefore reasonable to restrict major attention to promising edges, or called the candidate set, for accelerating local search (LS) methods. The examples of candidate sets include the k ($k \geq 1$) nearest neighbor subgraph, and the Delaunay graph, etc [40]. Of course, it is often expected the candidate set to be smaller so as not to result in a substantial increase in running time.

Helsgaun [23] estimates the chances of a given edge being a member of a good tour by using minimum 1-trees [40]. A 1-tree is a spanning tree with an additional edge combined two nodes with degree 1. The minimum 1-tree is often used for estimating the lower bound on the optimum since it is a relaxation of the TSP. The estimation, or called Held-Karp bound [22], can be improved

substantially on the 1-tree with Lagrangean relaxation [40] by applying *subgradient optimization* [22] for determining a certain transformation of the cost matrix with the suitable set of penalties, which each π_i is associated with each node i .

In α -nearness measure [23], each edge has a corresponding α -value, which is the minimum length when a 1-tree is required to contain the specified edge based on a transformed cost matrix. Intuitively, the more accurately Held-Karp bound achieved by the transformed cost matrix, the better estimation of α -values for the edges. Then the candidate edges of each node are sorted in ascending order of their α -values. Usually the minimum 1-tree shares many edges with an optimal tour. It is intuitive that the smaller α is for an edge, the more promising is this edge.

3.1.3 The features of the TSP landscape

The landscape is often studied in various ways. The first way is to understand the landscape in terms of simple parameters [44], in particular, to depict the phase transitions according to certain order parameters [10][17][24][53]. For the TSP, most researches are focused on the matrix D . Chessman et al. [10] investigated the standard deviation of the D as an order parameter by using a backtrack algorithm. Zhang [53] went deep into the precision of the elements in the D by using a branch-and-bound algorithm. The phase transitions in many hard computational problems have revealed interesting regularities across different problems and algorithms [24]. However, most real-world TSP instances are still classified as hard at the existing phase diagrams.

The second way is to understand the landscape based on the hints accumulated from the solving experience of existing algorithms.

Most modern LS variants [2][23][32][45] are essentially under the umbrella of iterated LS (ILS) [32], or called chained LS (CLS) [2], where the basic idea is to modify the current tour by applying a *kick-move* on a previous found tour instead of independently generated one. The intuition behind using chained starting tours is that the strong positive correlation between solution cost and the distance from the closest optimal tour, i.e., better local minima tend to have smaller distance to the closest global optimum by sharing many common partial structures [7][32][54]. The observations have led to the “big valley” hypothesis [7], which argues that the high-quality tours tend to concentrate on a very small subspace around the optimal tour(s) [46].

Moreover, the inherent features of a set of high-quality tours have been exploited in several ways. The first is reduction [29], which locks the edges in the intersection of the edge sets (\underline{E}_i) of the tours for speeding up the subsequent search. However, this method is brittle [54], especially as a locked edge is not part of good tour. The second is tour merging [11], which is to look for a high-quality tour in a restrict graph (G_R) consisting of the union of the edge sets (\underline{E}_U) of the tours through a “branch-width” algorithm. In fact, the number of edges in a G_R can be quite few due to the large number of shared edges among high-quality solutions. However, optimizing over a G_R can be hard as the original TSP instance. Besides, although a G_R may contain most of edges of an optimal tour, it may not ensure containing at least one optimal tour, then the tour merging may be failed to achieve optimum. The third is backbone guided search [54], which makes using of global information embedded in the pseudo-backbone frequencies for changing the landscape. However, the local search may be trapped into a local minimum unaffected by the changed landscape.

In this paper, the TSP landscape is exploited in the following ways: a) the LS method is applied at the initialization (or early) stage in order to reach the “big valley” as soon as possible [7]; b) The new tour(s) are generated based on the *kick-move* on a previous found tour as in the iterative LS [32]; c) an edge set (\underline{E}_U), which is represented in a set of high-quality tours, are used as the guiding information for the *kick-move*. The basic idea is that, for the edges to be introduced, most of them come from the \underline{E}_U and a few of promising edges are safely introduced by greedy search. Moreover, to use a \underline{E}_U with repeated edges instead of a restrict graph may utilize the pseudo-backbone frequencies implicitly; and d) the diversity among tours is maintained carefully: firstly, each starting tour is conserved in the private memory of each agent; secondly, the partial instead of the full information in \underline{E}_U , normally be a single tour, are used for guiding the *kick-moves*; and thirdly, the diffusion distance for each starting tour is restricted by an additional testing criterion on multiple trails.

3.2 Memory initialization

For simplicity, We use one T_INFO element in M_A , i.e. a tour called $\bar{s}_p^{(t)}$. Hence the element in M_S is $\underline{\bar{s}}_p^{(t)} = \{ \bar{s}_{p(i)}^{(t)} \mid i \in [1, N]_{\mathbb{Z}} \}$.

Following a random initialization, each tour is improved by a LS method, indicated as LS_{MI} . The default choice for LS_{MI} is a 3-Opt with $k_{NN}=20$ neighborhood nodes in the candidate set.

3.3 The generate-and-test behavior

For solving the TSP, the generate-and-test behavior can be seen as a *kick-move* on \bar{s}_a , the starting tour to be improved, as in ILS [32].

To improve the chances for escaping from the local optimal tours, the double-bridge (DB) [29][32], a non-sequential *kick-move*, is normally used [2]. Helsgaun [23] has extended it into a *2-stage kick-move*, i.e., any infeasible 2- or 3-Opt move (producing an intermediate solution consisting sub-tours) followed by any 2- or 3-Opt move, which produces a valid tour (by merging the sub-tours). The infeasible intermediate solution is significant so as to alter the global shape of a tour [2]. However, the blind mutation should be avoided since it may be ineffective and inefficient.

The *kick-move* guided by a high-quality edge set (\underline{E}_U) has been exploited extensively. Many of them [15][28][34][42] can be classified as the *tour-guided 2-stage kick-move* (TG2KM): guided by a tour \bar{s}_b in \underline{E}_U , the tour \bar{s}_a is first modified into an infeasible intermediate solution, normally consist disjoint segments or sub-tours, which are then repaired by ingenious greedy mechanisms while considering available information, such as the \underline{E}_C .

Many edge-based recombination operators, such as Voronoi Quantized Crossover (VQX) [42], Natural Crossover (NX) [28], Distance Preserving Crossover (DPX) [15], Edge Assembly Crossover (EAX) [34][35], etc., can be taken as the examples.

Guiding by a tour instead of mutating blindly brings an implicit advantage by utilizing the distance between \bar{s}_a and \bar{s}_b . Normally, the larger distance, the large perturbation on \bar{s}_a is then allowed. It brings an adaptive control of the balance between exploration and exploitation along with the execution cycles.

Using a single tour instead of the whole information available is simply because the former has inherent mechanism for preventing premature convergence by locally diffusion of information in a

self-organized agent network while the latter needs sophisticated mechanism although it provides more complete information.

Normally, new promising edges can only be safely introduced when repairing the intermediate solution in greedy way. Hence the unexpected edges that are introduced into the intermediate solution, which are not existed in the \bar{s}_b and the set \underline{E}_C , should be as less as possible. The following R_{GT} rule, called $R_{GT}^{\underline{EA}}$, is extended from the EAX [34], because it meets the requirement.

3.3.1 The generating behavior

The generating part (R_G) is applied for generating new tour(s).

The preliminary operation is to build the mappings between the external and internal information sources, i.e., from the $\bar{s}_p^{(t)}$ and the $\underline{SS}_p^{(t)}$ to the starting tour \bar{s}_a and the guiding edge set \underline{E}_U . Here $\bar{s}_a = \bar{s}_p^{(t)}$ so as to preserve information naturally by using the private memory; and $\underline{E}_U = \underline{SS}_p^{(t)}$ so as to make use of the cumulative pool. Moreover, the \bar{s}_b is a tour selected from the $\underline{SS}_p^{(t)}$ at random so as to utilize the pseudo-backbone frequencies implicitly.

By defining the G_{AB} as a graph constructed by merging \bar{s}_a and \bar{s}_b , the edges on G_{AB} are then divided into AB-Cycles [35], where an AB-cycle is defined as an even-length sub-cycle on G_{AB} generated by tracing different edges of \bar{s}_a and \bar{s}_b alternately. As an AB-Cycle is identified, then the edges making up the AB-Cycle are removed from G_{AB} and the AB-Cycle is stored into the AB-Lib if it contains not only common edges of \bar{s}_a and \bar{s}_b . The procedure is repeated till no AB-Cycle could be extracted.

Each AB-cycle in AB-Lib can be used as the guiding information for applying a *kick-move* on \bar{s}_a in two steps [35][49].

The first step is to generate an intermediate solution, i.e., by removing edges of \bar{s}_a in the AB-cycle from \bar{s}_a and adding edges of \bar{s}_b in the AB-cycle to \bar{s}_a . The result is a set of disjoint sub-tours. It is obviously that all the changed edges of \bar{s}_a are selected from \bar{s}_b . Besides, the edges in \bar{s}_a will be unchanged in the intermediate solution if the edges that connected to a node in \bar{s}_b belong to the edges that connected to the same node in \bar{s}_a .

The second step is to modify the intermediate solution into a valid tour by merging its sub-tours. Of course, it is a choice to apply the 2-Opt for merging sub-tours [9]. For reducing the computational cost, a LS version, which can be seen as the first round of 2-Opt, is used [34]: two sub-tours are greedy merged, by deleting one edge from each sub-tour and adding two edges, where one of them is a node selected at random in the candidate set \underline{E}_C , to connect the nodes in both sub-tours with degree 1.

Of course, to find a good tour by simple one trail is not always easy. Fortunately, there already are many AB-cycles in the AB-Lib, which allows us to perform a multi-trail mechanism (MTM), or called iterative child generation (ICG) [35][49], i.e. to generate N_{MTM} candidate tours, where each is based on an AB-Cycle selected in AB-Lib at random. The idea was also exploited by the Fitness-Distance-Based Diversification (FDD) [45] and the brood selection mechanism [48]. The default value for N_{MTM} is set as 30.

3.3.2 The testing behavior

The testing part (R_T) is used for maintaining the M_A , which is similar as updating the starting solution in iterated LS, based on the competition among the generated candidate tours.

The smaller difference between two information sources for the R_G part, the higher probability for falling into one of the local minima associated with the guiding information [7][15]. Hence, it is critically to maintain the diversity of information intentionally since the high-quality solutions are already rather close.

For the competition of generated tours, it may easily get trapped in specific regions of the TSP landscape if we only consider the essential high-quality criterion [49]. The FDD mechanism [45] hence utilizes an additional criterion, i.e., high quality as well as rather distant from the starting tour.

If \bar{s}_a is located at a local valley, then a tour better than \bar{s}_a is located at another valley of the TSP landscape. Here if only such tours are allowing for competition, then it is no long necessarily to keep large distance from the current solution intentionally.

It is expected that the private information in M_A has strong influence on the generated states, so as to keep a local diffusion effect. Hence large adjustments are possible, but are much less probable than small adjustments [4]. For a generated tour \bar{s}_c with better quality than \bar{s}_a , the cost function is transformed as follows:

$$f'(\bar{s}_c) = (f(\bar{s}_c) - f(\bar{s}_a)) / (DIS(\bar{s}_a, \bar{s}_c))^{C_D} \quad (4)$$

where $DIS(\bar{s}_a, \bar{s}_c)$ is a function to reflect the increasing of distance between \bar{s}_a and \bar{s}_c , and C_D is called *diffusion coefficient*. As in [34], the $DIS(\bar{s}_a, \bar{s}_c)$ function is defined as half of the number of edges in the corresponding AB-cycle to reduce computational cost for calculating the exact distance.

If $C_D=0$, $f'(\bar{s}_c)$ represents the high-quality criterion, as in EAX-1AB [34][35] or in brood selection mechanism [48]. If $C_D=1$, $f'(\bar{s}_c)$ represents the criterion as in EAX-Dis [35]. If C_D is very large, the \bar{s}_c will be selected from the tours with smallest DIS value. In fact, the C_D determines the diffusion velocity of information. The default setting is $C_D=1.5$.

The selected \bar{s}_c is then used for replacing the $\bar{s}_p^{(t)}$ in M_A . Hence \bar{x}_p is a steady-state solution satisfying the above testing criterion.

4. EXPERIMENTAL RESULTS

The experiments are carried out on a set of real-world instances with $1000 \leq V < 3000$ nodes in TSPLIB [39].

The results are compared with the iterative LS (ILS-FDD) [45] and the iterated backbone guided LK algorithm (IBGLK) [54].

The ILS-FDD generates a new starting tour for ILS with Fitness-Distance-Based Diversification (FDD) [45]. Their 3-Opt uses some standard speed-up techniques under a candidate list of 40 nearest neighbors. The ILS version has achieved best performance while comparing to three other algorithms, i.e., the genetic algorithm (GA) using DPX-crossover (GA-DPX) [15], the repair-based GA [48], and the MAX-MIN ant system (MMAS) [46].

The IBGLK [54] combines pseudo-backbone utilization with the iterated LK algorithms by considering two issues: the first issue is to construct unbiased samples of local minima for constructing pseudo-backbone; and the second issue is to alternate between BGLK and regular LK once each of them fails. For LK algorithm, length-20 nearest nodes, *don't look bits* [6], and the *two-level tree* [14] tour representation, are used. The IBGLK version has outperformed than three versions of LK algorithms [54], i.e. the *ILK-1-run*, the *ILK-5-run*, and the *ILK-10-run*.

Table 1 gives the setting parameters for the standard case of MAOS_E, called #Std, where \underline{E}_C indicate the candidate set. Table 2 gives five other cases by changing one parameter value based on the template of #Std. As same as in [45], 25 trails are run on pr1002, pc1173, d1291, and 10 trails are run on other instances.

Table 1. The parameter setting for the case #Std of MAOS_E

Parameter	N	N_{ACT}	\underline{E}_C	LS_{MI}	C_D	k_{NNI}	N_{MTM}
Value	300	300	α -measure	3-Opt	1.5	20	30

Table 2. The parameter changes for other cases (based on #Std)

Case	#NA100	#N400	#D1.0	#NN	#2Opt
Changes	$N_{ACT}=100$	$N=400$	$C_D=1.0$	$\underline{E}_C=nearest$	$LS_{MI}=2-Opt$

Table 3 shows the results for average percentage deviation (Δ_{avg}) from the optimum by ILS-FDD [45], IBGLK [54], and the MAOS_E cases. Besides, the average CPU-time (sec) for the ILS-FDD and the IBGLK, and the average cycles (\bar{T}) for the MAOS_E cases, are listed. We also list the run-time (sec) for the case #Std and #400. The result for si1032 is just for reference and is not used in the following discusses since it is not tested by ILS-FDD. At the last row, the average values for all instances are calculated.

For the MAOS_E cases, the calculation time mainly contains two parts: the first is N times of LS executions at the initialization stage; the second is $N_{ACT} \cdot \bar{T}$ times of R_{GT} executions by agents. The less R_{GT} executions is normally preferable, although there are great deviations for the costs between different R_{GT} executions.

Both the ILS-FDD and the IBGLK are coded in ‘‘C’’. The ILS is run on 450MHz P3 processor, and the IBGLK runtime is normalized for a 500MHz Alpha. The MAOS_E is coded in JAVA, and is run on 2.8GHz and 3GHz P4 processors in a public cluster. In fact, to compare the computational cost is not so easy since we can only count run-time instead of CPU-time by JAVA. However, the average run-time of the case #Std is 16.2% of that of the IBGLK. Hence the case #Std can be at least comparable with the IBGLK even by only considering the speed difference of CPUs.

The case #Std can achieve the optimum of thirteen instances in all trails, which is better than both the ILS-FDD and the IBGLK. Besides, here we use *worse than/equal to/better than* (W/E/B) analysis. For #Std vs. the ILS-FDD, the W/E/B is 2/6/12, which means that for the results of #Std, 2 instances was worse than, 6 instances were equal to, and 12 instances were better than that of ILS-FDD. For #Std vs. the IBGLK, the W/E/B is 3/6/11.

The effects of parameters are also concerned. From the number of solved instances, the #Std is slightly better than #D1.0, #NN, and #2Opt. However, as the parameter C_D is increased from 1.0 to 1.5, the average \bar{T} is increased by 14.1% due to the reduced diffusion velocity. Hence for the parameter C_D , the balance between the

quality and the cost must be minded. As the α -measure instead of the *nearest* measure for the candidate set is used, the average \bar{T} is reduced by 19.3%, which due to the better estimation of promising edges. Although 3-Opt itself is normally slower than the 2-Opt, the average \bar{T} can be reduced by 42.4%. Hence for the LS_{MI} , the 3-Opt is normally preferable than 2-Opt as the \bar{T} is large.

Then the *partial-run* mode is checked by two cases: #NA100 and #N400. For the #NA100 and the #Std, the time to be costed at the initialization stage is same. Although the average Δ_{avg} is slight worse, the R_{GT} execution times of #NA100 is 50.8% of that of #Std. For #N400, it can achieve quite good performance in reasonable run-time (sec), where the fails only occurs in five instances, which are mainly due to the enriched socially-shared information and preserved diversity among the larger society.

5. CONCLUSIONS

This paper has described an extended multiagent optimization system (MAOS_E) based on autonomy oriented computing, which consists of a society of agents under ecological rationality where they cooperate in a sharing environment to search on a virtual landscape, for solving hard computational problems. Although each agent only possesses limited memory and simple behavioral rules, the collective properties can emerged from their cooperation.

From the viewpoint of computation, each agent is a stochastic algorithm. Although the agents have same organized structure, they become heterogonous since their memories are initialized with different contents. If the agents do not utilize the information in M_S , then the system becomes a *portfolio of algorithms* [20][26]. The studies [20][26][33] have indicated the portfolios may have unequivocal advantage than any of its individual member algorithms running alone for dealing with hard computation problems. In MAOS_E, the agents are cooperated through the information located in M_S . Then the multiagent system becomes a *cooperative algorithm portfolio*. The cooperation between agents will introduce statistical correlations between their performance, which is capable of reducing both the expected time and the solving risk when manifested in negative correlations [26].

Under the MAOS_E, the realization of a problem solver then can be divided into three parts: a) encode the representation of the problem into the landscape and the possible auxiliary knowledge; b) construct the memory elements at the initialization stage; and c) design the *generate-and-test* behavior guided by the law of *socially-biased individual learning*. The implementation has been demonstrated on the TSP, where all three parts are realized by exploiting the problem features. The extensive experimental results on real-world instances in TSPLIB show its efficiency as comparing to the ILS-FDD and the IBGLK.

One possible extension of current work is to apply the agents for solving other well-studied hard computational problems, which are key components for many practical tasks, by utilizing inherent features. Another possible extension is to study rational strategies for agents to deploy multiple R_{GT} rules under certain knowledge.

The agents may also be adapted to the cooperative distribute problem solving (CDPS) [13][52] by requesting the guiding information from other agents. Of course, the negotiation mechanisms then may be required, especially as the agents use various virtual landscapes for involved task.

6. REFERENCES

- [1] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. An integrated theory of the mind. *Psychological Review*, 111: 1036-1060, 2004.
- [2] Applegate, D., Cook, W., and Rohe, A. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15: 82-92, 2003.
- [3] Baddeley, A. Exploring the central executive. *The Quarterly Journal of Experimental Psychology A*, 49: 5-28, 1996.
- [4] Bak, P. *How Nature Works: The Science of Self-Organized Criticality*. Berlin: Springer, 1996.
- [5] Bandura, A. *Social Foundations of Thought and Action: a Social Cognitive Theory*. NJ: Prentice-Hall, 1986.
- [6] Bentley, J. L. Fast algorithms for geometric traveling salesman problems. *ORSA J. Computing*, 4: 387-411, 1992.
- [7] Boese, K. D., Kahng, A. B., and Muddu, S. A new adaptive multi-start technique for combinatorial global optimizations. *Operation Research Letters*, 16: 101-113, 1994.
- [8] Boyd, R. and Richerson, P. J. *The Origin and Evolution of Cultures*. New York: Oxford University Press, 2005.
- [9] Chan, C. H., Lee S. A., Kao C. Y., et al. Improving EAX with restricted 2-opt. *Genetic and Evolutionary Computation Conference*, Washington DC, USA, 1471-1476, 2005.
- [10] Cheeseman, P., Kanefsky, B., and Taylor, W. M. Where the really hard problems are. *International Joint Conference on Artificial Intelligence*, San Mateo, CA, 331-337, 1991.
- [11] Cook, W. and Seymour, P. Tour merging via branch-decomposition. *INFORMS J. Computing*, 15: 233-248, 2003.
- [12] Dorigo, M., Maniezzo, V., and Colomi, A. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Systems, Man, and Cybernetics - Part B*, 26: 1-13, 1996.
- [13] Durfee, E. H., Lesser, V. R., and Corkill, D. D. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1: 63-83, 1989.
- [14] Fredman, M. L., Johnson, D. S., McGeoch, L. A., and Ostheimer, G. Data structures for traveling salesmen. *Journal of Algorithms*, 18: 432-479, 1995.
- [15] Freisleben, B. and Merz, P. New genetic local search operators for the traveling salesman problem. *International Conference on Parallel Problem Solving from Nature*, Berlin, Germany, 890 - 899, 1996.
- [16] Galef, B. G. and Laland, K. N. Social learning in animals: Empirical studies and theoretical models. *BioScience*, 55: 489-499, 2005.
- [17] Gent, I. P. and Walsh, T. The TSP phase transition. *Artificial Intelligence*, 88: 349 - 358, 1996.
- [18] Gigerenzer, G. *Adaptive Thinking: Rationality in the Real World*. New York: Oxford University Press, 2000.
- [19] Glenberg, A. M. What memory is for. *Behavioral and Brain Sciences*, 20: 1-55, 1997.
- [20] Gomes, C. P. and Selman, B. Algorithm portfolios. *Artificial Intelligence*, 126: 43-62, 2001.
- [21] Gu, J. and Huang, X. F. Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics*, 24: 728-735, 1994.
- [22] Held, M. and Karp, R. M. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18: 1138-1162, 1970.
- [23] Helsgaun, K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126: 106-130, 2000.
- [24] Hogg, T., Huberman, B. A., and Williams, C. P. Phase transitions and the search problem. *Artificial Intelligence*, 81: 1-15, 1996.
- [25] Hordijk, W. A measure of landscapes. *Evolutionary Computation*, 4: 335-360, 1996.
- [26] Huberman, B. A., Lukose, R. M., and Hogg, T. An economics approach to hard computational problems. *Science*, 275: 51-54, 1997.
- [27] Jennings, N. R., Sycara, K., and Wooldridge, M. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1: 7-38, 1998.
- [28] Jung, S. and Moon, B. R. Toward minimal restriction of genetic encoding and crossovers for the two-dimensional Euclidean TSP. *IEEE Transactions on Evolutionary Computation*, 6: 557-565, 2002.
- [29] Lin, S. and Kernighan, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21: 498-516, 1973.
- [30] Liu, J., Han, J., and Tang, Y. Y. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136: 101-144, 2002.
- [31] Liu, J., Jin, X. L., and Tsui, K. C. *Autonomy Oriented Computing (AOC): From Problem Solving to Complex Systems Modeling*. Kluwer Academic Publishers, 2005.
- [32] Martin, O. C., Otto, S. W., and Felten, E. W. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11: 219-224, 1992.
- [33] Maurer, S. M., Hogg, T., and Huberman, B. A. Portfolios of quantum algorithms. *Physical Review Letters*, 87: 257901, 2001.
- [34] Nagata, Y. The EAX algorithm considering diversity loss. *International Conference on Parallel Problem Solving from Nature*, Birmingham, UK, 332-341, 2004.
- [35] Nagata, Y. and Kobayashi, S. Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. *International Conference on Genetic Algorithms*, East Lansing, MI, USA, 450-457, 1997.
- [36] Newell, A. The knowledge level. *AI Magazine*, 2: 1-20, 1981.
- [37] Newell, A. and Simon, H. A. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
- [38] Olick, J. K. and Robbins, J. Social memory studies: From "collective memory" to the historical sociology of mnemonic practices. *Annual Review of Sociology*, 24: 105-140, 1998.

- [39] Reinelt, G. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3: 376-384, 1991.
- [40] Reinelt, G. *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin: Springer, 1994.
- [41] Romney, A. K., Boyd, J. P., Moore, C. C., et al. Culture as shared cognitive representations. *Proc. Natl. Acad. Sci. USA*, 93: 4699-4705, 1996.
- [42] Seo, D. I. and Moon, B. R. Voronoi quantized crossover for traveling salesman problem. *Genetic and Evolutionary Computation Conference*, New York, 544-552, 2002.
- [43] Smith, E. E. and Jonides, J. Storage and executive processes in the frontal lobes. *Science*, 283: 1657-1661, 1999.
- [44] Stadler, P. F. and Schnabl, W. The landscape of the traveling salesman problem. *Physics Letter A*, 161: 337-344, 1992.
- [45] Stützle, T., Grün, A., Linke, S., and Rüttger, M. A comparison of nature inspired heuristics on the traveling salesman problem. *International Conference on Parallel Problem Solving from Nature*, Paris, France, 661-670, 2000.
- [46] Stützle, T. and Hoos, H. H. MAX-MIN ant system. *Future Generation Computer Systems*, 16: 889-914, 2000.
- [47] Tsui, K. C. and Liu, J. Multiagent diffusion and distributed optimization. *International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia, 169-176, 2003.
- [48] Walters, T. Repair and brood selection in the traveling salesman problem. *Int. Conf. on Parallel Problem Solving from Nature*, Amsterdam, the Netherlands, 813-822, 1998.
- [49] Watson, J., Ross, C., and Eisele, V., et al. The traveling salesrep problem, edge assembly crossover, and 2-opt. *International Conference on Parallel Problem Solving from Nature*, Amsterdam, the Netherlands, 823-832, 1998.
- [50] Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1: 67-82, 1997.
- [51] Xie, X. F. and Liu, J. A compact multiagent system based on autonomy oriented computing. *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Compiègne, France, 38-44, 2005.
- [52] Yakoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. Distributed constraint satisfaction for formalizing distributed problem solving. *International Conference on Distributed Computing Systems*, Yokohama, Japan, 614-621, 1992.
- [53] Zhang, W. Phase transitions of the asymmetric traveling salesman. *International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 1202-1207, 2003.
- [54] Zhang, W. and Looks, M. A novel local search algorithm for the traveling salesman problem that exploits backbones. *International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 343-348, 2005.

Table 3. The average results on TSPLIB instances ($1000 \leq V < 3000$) by ILS-FDD [45], IBGLK [54] and MAOS_E cases

<i>F</i>	ILS-FDD (sec)	IBGLK (sec)	#NA100 (\bar{T})	#N400 (\bar{T}/sec)	#Std (\bar{T}/sec)	#D1.0 (\bar{T})	#NN (\bar{T})	#2Opt (\bar{T})
dsj1000	0.0091(604.2)	0.01 (2066)	1E-04 (119.5)	0 (89.4/477)	0 (78.8/203.9)	3E-4 (58.9)	0 (100.6)	0 (103.8)
pr1002	0 (93.5)	0.00 (345)	0 (76.9)	0 (59.52/338.6)	0 (56.68/126.6)	0 (44.92)	0 (78.4)	0 (106.08)
u1060	0.009 (671.8)	0.02 (873)	0 (106.9)	0 (67.3/411.1)	0 (63.4/351.7)	0 (51.3)	0 (86.3)	0 (109.6)
si1032	-	0.00 (772)	0 (2.1)	0 (1.7/51.7)	0 (1.5/21.7)	0 (2.5)	1E-4 (9.2)	8E-4 (9.8)
vm1084	0.011 (340.8)	0.02 (403)	0 (67.0)	0 (55.8/388.9)	0 (52.5/327.9)	0 (42.9)	0 (69.9)	0 (69.9)
pcb1173	0.0011 (493.8)	0.00 (213)	0.004 (86.9)	7E-4 (68.04/466.4)	6E-4 (61.88/168.9)	0.003 (55.4)	4E-4 (71.7)	0 (105.7)
d1291	0 (320.5)	0.08 (1086)	0 (39.2)	0 (30.04/345.1)	0 (28.48/137.7)	0 (22.36)	5E-4 (39.2)	0.003 (47.2)
rl1304	0 (23.6)	0.00 (504)	0.014 (55.8)	0 (39.8/339.1)	0 (39.9/162.7)	0.007 (33.6)	0.002 (49.1)	0 (50.8)
rl1323	0 (148.3)	0.00 (582)	0 (55.3)	0 (40.7/353.2)	0 (35.2/279.4)	0 (30.4)	0.005 (49.3)	0 (55.5)
nrw1379	0.051 (867.3)	0.06 (440)	0.006 (147.5)	0.003 (102.8/417.2)	0.004 (88.4/385.1)	0.003 (72.8)	0.005 (116.8)	0.006 (176.4)
fl1400	0 (218.2)	0.00 (19317)	0.002 (147.7)	0 (111.1/460.5)	0 (102.3/448.0)	0 (94.9)	0 (115.8)	0 (116.8)
u1432	0.039 (763.4)	0.00 (617)	0 (97.4)	0 (67.5/334.5)	0 (59.7/570.3)	0 (53.5)	0.003 (75.5)	0.008 (132.4)
fl1577	0 (648.5)	0.00 (10430)	0.01 (102.1)	0.017 (72.3/690.6)	0.027 (47.5/239.9)	0.034 (42.8)	0.053 (84.7)	0.009 (98.8)
d1655	0 (803.1)	0.00 (1869)	8E-04 (94.8)	0 (67.3/777.3)	0 (60.7/325.1)	0 (46.8)	6E-4 (94)	0.002 (129)
vm1748	0.031 (1267.4)	0.01 (829)	0 (128.3)	0 (96.1/1133.1)	0 (86.7/612.5)	0 (69.3)	0 (108)	0 (124.8)
u1817	0.056 (1316.3)	0.16 (680)	0.001 (136.4)	0 (91.4/1125.2)	0.003 (74.3/439.7)	0.006 (62.9)	7E-4 (101.3)	0.008 (126.3)
rl1889	0.0064(1101.4)	0.22 (1110)	0 (95.7)	0 (72.8/567)	0 (65.8/525.1)	4E-4 (53.3)	0 (88.9)	0 (91.2)
d2103	0.001 (1747.7)	0.00 (8669)	0.036 (100.6)	0.003 (63.9/495.5)	0.019 (44.7/365.6)	0.004 (41.5)	0.076 (50.6)	0.005 (107.5)
u2152	0.061 (2401.9)	0.19 (808)	0.002 (114.8)	0 (79.4/714.7)	0.002 (71.9/662.7)	5E-4 (57.6)	0 (84.9)	8E-4 (149)
u2319	0.11 (1233.8)	0.04 (2091)	0.119 (143.5)	0.041 (163.1/1434.6)	0.038 (132.1/1251.8)	0.025 (177.2)	0.067 (158.5)	0.049 (256.2)
pr2392	0.004 (2079.2)	0.14 (715)	0 (175.5)	0 (129/1159.6)	0 (120.5/1123.0)	0 (89.9)	0 (152.7)	0 (225.1)
Average	0.019 (857.2)	0.047 (2682.3)	0.0098 (104.6)	0.0032 (78.4/621.5)	0.0047(68.6/435.4)	0.0041(60.1)	0.010 (85.0)	0.0045 (119.1)