International Journal of Applied Metaheuristic Computing, 2012, 3(4): 1-24.

Round-Table Group Optimization for Sequencing Problems

Xiao-Feng Xie

The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

In this paper, a round-table group optimization (RTGO) algorithm is presented. RTGO is a simple metaheuristic framework using the insights of research on group creativity. In a cooperative group, the agents work in iterative sessions to search innovative ideas in a common problem landscape. Each agent has one base idea stored in its individual memory, and one social idea fed by a round-table group support mechanism in each session. The idea combination and improvement processes are respectively realized by using a recombination search (XS) strategy and a local search (LS) strategy, to build on the base and social ideas. RTGO is then implemented for solving two difficult sequencing problems, i.e., the flowshop scheduling problem and the quadratic assignment problem. The domain-specific LS strategies are adopted from existing algorithms, whereas a general XS class, called socially biased combination (SBX), is realized in a modular form. The performance of RTGO is then evaluated on commonly-used benchmark datasets. Good performance on different problems can be achieved by RTGO using appropriate SBX operators. Furthermore, RTGO is able to outperform some existing methods, including methods using the same LS strategies.

Keywords: Meta-heuristic frameworks, Group creativity, Sequencing problems, Global optimization, Recombination search, Idea combination process, Social-biased learning

1. Introduction

Group creativity techniques, e.g., *brainstorming* [39], have been widely studied in social science [34, 40]. A cooperative human group contains multiple individuals who have some interactions on ideas of each others [40], and the group tries to find innovative solutions (or high-quality *ideas*) for a specific task by generating new (especially innovative) ideas spontaneously contributed by its members in iterative idea-generating sessions.

Group creativity has been studied in both group and individual levels. At the group level, a *support mechanism* is used to assist individuals by providing useful external stimulus in their idea-generation process. The essential function of these mechanisms is to serve as *group memory* [4, 53] for the diffusion of innovative patterns [47] as well as providing diverse ideas

Email address: xfxie@cs.cmu.edu (Xiao-Feng Xie)

[40], based on a developing repository of nonprivate knowledge shared by members. For individuals, group memory might be heterogeneously shaped in *network structures* [25, 51].

For each individual, the ideation process involves *idea selection* [41, 46] and *idea generation* [21, 36, 40], based on its *individual memory* [15, 35]. A pre-selection mechanism [41] might be used to pick out relevant ideas from both the individual and group memory. Then an idea-generation mechanism is used to build new idea(s) on selected ideas. Afterward, a post-selection mechanism [46] is applied to update the individual memory. Each individual also holds a *sharing mechanism* to contribute nonprivate knowledge [24] for the group.

The idea-generation process is a critical part of the creative process. Since the research of Osborn [39], *combine-and-improve ideas* has been a general brainstorming rule to form a single better idea by building on existing ideas. The total process of building on existing ideas might be divided into two parts, the idea combination process [21, 65] and the idea improvement process, roughly corresponding to the divergent and convergent processes, where the former is of paramount importance although the latter is also nontrivial [8].

Human problem solving can be seen as searching in a structural space of states (ideas) [35]. Thus, there is a natural similarity between creative idea-generating tasks and hard optimization problems [25], both of them require efficiently achieving high-quality solutions.

Many optimization problems can be formulated as sequencing (or permutation) problems [22, 54]. All sequencing problems with n nodes share the same problem space, in which each potential solution (or *state*) $\vec{x}$ is a permutation $\{x_1, \ldots, x_n\}$ of the integer values from 1 through n, only their objective functions $f(\vec{x})$ possess different structural properties.

In this paper, we consider two typical sequencing problem examples, i.e., the quadratic assignment problem (QAP) [5, 57] and the flowshop scheduling problem (FSP) [3, 45, 59], among some other examples. Both QAP and FSP are *NP*-hard in the strong sense. FSP is a well-known problem in intelligent manufacturing systems. QAP arises in many practical applications, e.g., design of grey patterns and website structure improvement [52]. QAP also serves as a generalization of some other important optimization problems [27].

Exact algorithms, e.g., branch-and-bound [6], can only be tractable for solving small-scale instances, whereas fast constructive heuristics [31, 44] often obtain results that are far away from optimal. Thus, low-level search components, especially *local search* (LS) and *recombination search* (XS), as well as upper-level *meta-heuristic frameworks*, have been integrated together for finding near optimal solutions within practical computational costs.

Each LS strategy improves an incumbent solution by intensively *moves* based on some *neighborhood search* operations [14], e.g., 2-opt. A LS strategy is defined as *stable* if it only allows non-worse moves. Any stable LS, such as hill-climbing and the fast LS [14], cannot escape from the local minimum it first encounters. Some advanced LS strategies, such as simulated annealing [33], tabu search [37, 58], iterated local search [50, 55], etc., incorporate some unstable moves so as to explore in a rugged problem landscape.

Each XS strategy generates a new solution by preserving positive clues in two parent solutions, which has an implicit advantage of adaptive leaping by utilizing the difference between two parents [27]. Typical examples of XS operators for sequencing problems include order crossovers [30, 54], LCS crossover [18], similar block crossovers [49], partially mapped crossovers [54], distance preserving crossover [27], and path crossovers [2], etc.

Meta-heuristic frameworks, which use LS and XS strategies as their search components, have been applied for solving sequencing problems. Typical examples include ant colony optimization [14, 20, 26, 42], genetic algorithms and memetic algorithms [18, 27, 30, 45, 49, 63], particle swarm optimization [11, 60], and some other systems [62], etc.

In this paper, a round-table group optimizer (RTGO) is proposed for solving sequencing problems. We use the insights of previous research on group creativity to provide a theoretical context. RTGO is an extremely simple form of a cooperative group, in which each individual is an idea-generating agent. The agents work in iterative sessions to search innovative ideas in a common problem landscape. All agents are all intrinsically motivated to generate new ideas, and thus any negative group effects [40] are precluded.

The group support mechanism is a simple round-table mechanism. In each session, the agents are randomly (re-)allocated around a round table, where each agent only accesses one *social idea* (in the group memory) from the adjacent neighbor. This simple mechanism captures a limited sense on communicative and cognitive interference [40].

Each agent possesses only one *base idea* in its individual memory, and holds a XS strategy and a LS strategy respectively for the idea combination and improvement processes. The pre-selection mechanism only returns the base and social ideas for supporting socially biased learning [13, 38, 62]. The XS strategy then generates an intermediate idea modified from the basic idea by combining stimulation cues in the social idea. Afterward, the LS strategy locally refines the intermediate one into a polished new idea. The post-selection mechanism then stores the new idea if it is not worse than the old base idea in its individual memory. For each agent, its individual memory always stores the best-so-far idea during its own search.

Our focus is then on examining the meta-heuristic framework, RTGO, with different idea combination processes. Two stable LS strategies respectively for FSP [49, 50] and QAP [14] are chosen from prior work. In the "big-valley" structure [27, 45] of sequencing problems, any stable LS strategies can only find near local optimal solutions, thus the capability of efficiently escaping from local minima can only be achieved by the idea combination process. In addition, it should be quite fair to compare the overall performance of RTGO to that of some existing optimization frameworks using commonly-used LS strategies.

The remainder of this paper is organized as follows. In Section 2, RTGO is presented. In Section 3 , knowledge components for both sequencing problems are implemented into RTGO. In Section 4, the characteristics of RTGO are investigated by performing computational experiments on some benchmark datasets. This paper is concluded in Section 5.

2. Round-Table Group Optimizer (RTGO)

RTGO is a very simple meta-heuristic framework based on the insights of previous research on group creativity. As shown in Figure 1, RTGO contains a small group of idea-generating agents that their interactions are supported by using a round-table mechanism. Each specific task is formulated into a problem landscape, where each state in the problem space is an *idea*. For the agents, the goal is to obtain a high-quality idea with a near optimal objective value in the problem space, through generating new ideas in iterative sessions. The actual algorithm performance will demonstrate the effectiveness of group creativity.

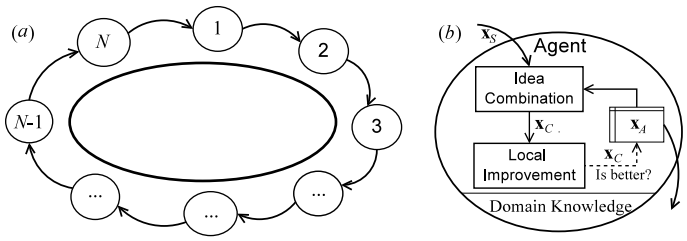

Figure 1: (a) A round table seated with N agents in a brainstorming group; and (b) the details of an agent.

At the group level, a simple round-table mechanism is considered to provides background supports for the group memory among the agents, as shown in Figure 1(a). In each session, the agents are randomly assigned to the seats around a round table, and each agent shares a nonprivate idea in its individual memory as the only social idea for its clockwise neighbor. Afterward, agents can spontaneously generate new ideas using the base and social ideas.

Each agent has a limited capability of generating innovative ideas based on available information, including available ideas and problem domain knowledge. Figure 1(b) gives the details of an agent, which possesses basic characteristics of an individual in an idea-generating group, although it is realized in a rather simple form, which involves the individual memory as well as sharing, (pre- and post-) idea selection, and idea generation mechanisms.

First, each agent possesses an individual memory that can only be modified by the agent itself. For an agent, memory [15] is essential for supporting its individual learning capability in utilizing its past experience. Here the memory contains only one base idea $\vec{x}_A$.

Second, each agent shares (parts of) its memory as nonprivate information [9], as a basic sharing mechanism to affect the group [24]. Here the whole $\vec{x}_A$ is contributed.

Third, each agent has the capability to utilize ideas that are contributed by others [40]. For real-world animals, the social learning capability [13, 38] can enhance their adaptability in a changing environment, which may lead to a cumulative evolution of ideas that are more novel and useful than those contributed by individuals [21]. In each session, each agent only accepts one social idea, called $\vec{x}_S$, from the group memory in its environment.

Finally, the ideation capability of each agent is accomplished by a socially-biased combine-and-improve procedure, as shown in Figure 1(b). The pre-selection mechanism simply returns $\vec{x}_A$ and $\vec{x}_S$. The idea combination process forms a new idea $\vec{x}_C$ by combining $\vec{x}_A$ and $\vec{x}_S$. Afterward, the local improvement process is applied for further improving $\vec{x}_C$. The post-selection mechanism is then used to update the individual memory using $\vec{x}_C$.

Algorithm 1 gives the working process of RTGO. Initially, each agent owns an idea that is randomly generated in the problem space (Line 1). Then RTGO runs in iterative sessions. Lines 3-6 describe the simple round-table mechanism that provides $\vec{x}_S$ for each agent. In Line 8, each agent runs the combine-and-improve procedure, including the idea combination process and the local improvement process, which are respectively realized using a XS strategy an a LS strategy, to generate a promising idea $\vec{x}_C$. Under the RTGO framework, the combine-and-improve procedure can be realized in the form of socially-biased learning [13] that using both the base and social ideas, where the base idea serves as an incumbent solution, and the social idea provides stimulation clues in different sessions. For

Algorithm 1 Round-table group optimizer (RTGO)

Initially, each agent owns an idea $\vec{x}_A$ that is randomly generated in the problem space
for Session $t = 1$ to T **do**
  The agents randomly take the seats (labelled from 1 to N) around a round table
  for Agent $h = 1$ to N **do**
    $\vec{x}_{S,((h+1) \bmod N)} = \vec{x}_{A,h}$ {Agent h shares its base idea to its clockwise neighbor}
  end for
  for Agent $h = 1$ to N **do**
    $\vec{x}_{C,h} = \mathrm{XS}(\vec{x}_{A,h}, \vec{x}_{S,h})$; $\vec{x}_{C,h} = \mathrm{LS}(\vec{x}_{C,h})$ {socially-biased combine-and-improve}
    if $f(\vec{x}_{A,h}) \geq f(\vec{x}_{C,h})$ **then** $\vec{x}_{A,h} = \vec{x}_{C,h}$ {steady-state memory}
  end for
end for
return $\vec{x}^* = (\vec{x}_{A,h}$ with the minimum objective value for $h \in [1, N])$

each agent, $\vec{x}_A$ is a steady-state idea over sessions since it is replaced by $\vec{x}_C$ only if $\vec{x}_C$ has a better quality. Finally, the best idea $\vec{x}^*$ of all agents is the solution of group brainstorming.

RTGO has two overall setting parameters, i.e., the number of agents N and the number of sessions T, and XS and LS strategies are to be implemented for specific problems.

3. Implementation on Sequencing Problems

Sequencing problems have the same problem space. For a problem with n nodes, each state $\vec{x}$ is an array containing a permutation $\{x_1, \ldots, x_n\}$ of the integer value from 1 through n. However, different sequencing landscapes might differ in domain-specific structures.

3.1. Problem Description

In this paper, we consider two important examples of sequencing problems, i.e., the flowshop scheduling problem (FSP) and the quadratic assignment problem (QAP).

3.1.1. Flowshop Scheduling Problem (FSP)

The flowshop scheduling problem (FSP) consists in scheduling n independent jobs to be processed on m independent machines in the same order. At any time, each job has one operation on one machine and each machine can process only one job. There is an $n \times m$ matrix of processing times $P{=}(p_{ij})$, where each p_{ij} is the processing time of the operation of the ith job on the jth machine. Each state $\vec{x}$, i.e., a schedule, is a job processing permutation, where $x_i (i \in [1, n])$ denotes the processing order of the ith job on every machine.

For each schedule $\vec{x}$, the completion time $c(\vec{x}, i, j)$ of the ith job on the jth machine is

$$c(\vec{x}, i, j) = \max(c(\vec{x}, i-1, j), c(\vec{x}, i, j-1)) + p_{x_i j}, \tag{1}$$

where $c(\vec{x}, i, 0) = 0$ for $\forall i \in [1, n]$ and $c(\vec{x}, 0, j) = 0$ for $\forall j \in [1, m]$.

The objective function is the total completion time (or makespan) of each $\vec{x}$, i.e.,

$$f(\vec{x}) = c(\vec{x}, n, m). \tag{2}$$

3.1.2. Quadratic Assignment Problem (QAP)

The quadratic assignment problem (QAP) consists of assigning n facilities to n locations, one facility at a location. There are two $n \times n$ matrices, i.e., the flow matrix $W = (w_{ij})$ and the distance matrix D=(d_{ij}), in which w_{ij} is the flow between facilities i and j, and d_{ij} is the distance between locations i and j. The objective function is

$$f(\vec{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} w_{x_i x_j}. \tag{3}$$

In addition, an asymmetric instance is preprocessed into a symmetric one without changing the resulting cost, if one of the two matrices D and W is symmetric [27].

3.2. Local Improvement

For the local improvement process, two stable LS strategies are respectively used for the two sequencing problems. Each local search strategy starts from an incumbent state $\vec{x}$ and then tries to improve it by executing basic neighborhood moves in a systematic way.

For FSP, we consider an insertion-based local search operator [49, 50, 55], as shown in Algorithm 2. The basic operation $BestInsert(\vec{x}, k_i)$ returns the best permutation obtained by inserting a job k_i to any possible position of $\vec{x}$. Although each objective function alone needs to be calculated in the time complexity $O(n \cdot m)$, the total $(n-1)$ positions can be checked in $O(n \cdot m)$ as well, based on the speed-up technique proposed by Taillard [57].

Algorithm 2 Iterative insertion-based local search for each incumbent state $\vec{x}$ of FSP

```
1: isImproved = true; f_O = f(x)
2: while isImproved do
3:    isImproved = false; Generate a random sequence (k_1, ..., k_n) for k_i ∈ [1, n]
4:    for i = 1 to n do
5:       x = BestInsert(x, k_i)
6:       if f(x) < f_O then
7:          f_O = f(x); isImproved = true
8:       end if
9:    end for
10: end while
```

For QAP, we consider an exchange-based LS operator [14, 43], which runs twice of the complete neighborhood search in Algorithm 3. It differs from *fast-2-opt* [27] only in that the latter one iteratively runs Algorithm 3 until no improvement can be found.

The delta quality of each exchange between locations i and j can be calculated as [58]:

$$\begin{aligned} \Delta Exchange(\vec{x}, i, j) = (d_{ii} - d_{jj}) \cdot (w_{x_j x_j} - w_{x_i x_i}) + (d_{ij} - d_{ji}) \cdot (w_{x_j x_i} - w_{x_i x_j}) \\ + \textstyle\sum_{k=1, k \neq i,j}^{n} ((d_{ki} - d_{kj}) \cdot (w_{x_k x_j} - w_{x_k x_i}) + (d_{ik} - d_{jk}) \cdot (w_{x_j x_k} - w_{x_i x_k})) \end{aligned} \tag{4}$$

If both matrices W and D are symmetric, and that all diagonal elements of either matrix are zeros, the delta evaluation can be simplified as [58]:

$$\Delta Exchange(\vec{x}, i, j) = 2 \cdot \textstyle\sum_{k=1, k \neq i,j}^{n} (d_{ik} - d_{jk})(w_{x_j x_k} - w_{x_i x_k}) \tag{5}$$

Algorithm 3 Exchange-based neighborhood search for each incumbent state $\vec{x}$ of QAP

1: Generate two random sequences $(r_1, \cdots, r_n)$ and $(s_1, \cdots, s_n)$ for $r_i \in [1, n]$ and $s_j \in [1, n]$
2: **for** $i = 1$ to n, $j = 1$ to n **do**
3: **if** $\Delta Exchange(\vec{x}, r_i, s_j) < 0$ **then** Exchange r_i and s_j in $\vec{x}$
4: **end for**

Both delta evaluation methods are computable in $O(n)$, and the latter one is a bit faster than the former one. In addition, an asymmetric instance can be converted into a symmetric one if one of the two matrices is symmetric [27].

3.3. Idea Combination

The idea combination is realized by a XS strategy, which produces one output idea, i.e., $\vec{x}_C$, by using information from two input ideas, i.e., the base idea $\vec{x}_A$ and the social idea $\vec{x}_S$.

We define a class of XS strategies, called *socially biased combination* (SBX), where $\vec{x}_A$ and $\vec{x}_S$ are viewed as the incumbent idea and social guidance information, respectively.

SBX operators can be realized in basic or macro forms. Each basic SBX operator contains three policies, i.e., a *base policy*, a *social policy*, and a *repair policy*. Initially, all positions in $\vec{x}_C$ are unoccupied. Then the three policies are sequentially executed to occupy positions in $\vec{x}_C$ with unused values, so that all positions are eventually occupied. Each macro SBX operator can be realized by applying a *macro policy* on basic and macro SBX operators.

3.3.1. Base Policy

The base policy marks the nodes from certain positions of the base state $\vec{x}_A$, and then copies the values at the marked nodes into the corresponding positions of $\vec{x}_C$.

There are three commonly-used modes. In the *one-point* mode (1P) [30], one cut-point is randomly selected, and then the set of nodes on a randomly chosen side of the cut-point are marked. In the *two-point* mode (2P) [30], a pair of cut-points is randomly selected, then the nodes located on either inside or outside of the selected two cut-points are marked. In the *uniform* mode (U) [56], each position is marked with the probability of 0.5. All these modes preserve position-based information. The difference is in that 1P, 2P and U provide $O(n)$, $O(n^2)$, and $O(2^n)$ numbers of choices, respectively. Furthermore, the 1P mode preserves whereas the U mode loses most precedence and adjacency information in the base idea.

Furthermore, a *common-avoiding* one-point mode (CA1P) is proposed. CA1P differs from 1P only in that the common nodes from the left and right sides are not considered when randomly selecting the cut-point. For example, for two states $\{2\ 6\ 1\ 3\ 4\ 9\ 5\ 8\ 7\}$ and $\{2\ 6\ 1\ 4\ 8\ 5\ 3\ 9\ 7\}$, the first three nodes and the last one node are the common nodes at both sides, and thus the cut-point is randomly chosen between the 4th location and the 7th location. CA1P ensures the marked nodes do not contain fully common values.

3.3.2. Social Policy

The social policy fills some remaining unoccupied positions of $\vec{x}_C$ by using unused nodes at certain selected positions of the social state $\vec{x}_S$. In the *position-based* mode (P), the

unused values of the state $\vec{x}_S$ are copied to corresponding positions of $\vec{x}_C$ if such positions are unoccupied. Hence, this mode preserves position and precedence information of the selected nodes of $\vec{x}_S$. In the *order-based* mode (O), all unused nodes of $\vec{x}_S$ are respectively copied to the unoccupied positions from left to right. Hence, this mode preserves precedence information of the selected nodes of $\vec{x}_S$. Furthermore, the order-based mode can always achieve a valid state, wheres the position-based mode may leave some positions unoccupied.

3.3.3. Repair Policy

The repair policy turns $\vec{x}_C$ into a valid state if there are any unoccupied positions. Here two modes are realized. The *random* mode (R) fills each unoccupied position by a randomly selected unused value. The *partially mapped* mode (PM), i.e., the final step in the partially mapped crossover [54], uses the position-based mappings between the nodes of both parent states to fill remaining positions for further preserving the order and position.

3.3.4. Macro Policy

In this paper, we only considered a simple macro policy, called the *parallel* mode (MP), which outputs the state with the best quality among the valid candidate states generated by totally N_{ICG} independent trials of a component XS operator.

The parallel mode can be viewed as an inner portfolio of algorithms [16], since the component XS operator can be seemed as a stochastic algorithm. An algorithm portfolio may avoid the heavy tails of individual trials, thus the output state can be more promising.

3.3.5. Summary

Each SBX case can be formally described by specifying its components, so that it is easy to know the subtle similarity and difference between various strategies.

A basic SBX case is denoted as a combination of three tags, i.e., A/B/C, where the tags A, B, and C designates the modes of base, social, and repair policies, respectively. Moreover, a tag is denoted as "-" if the corresponding policy is not required, or as "*" if the corresponding policy has not been assigned. If the parallel macro policy is applied on a basic SBX case A/B/C, then the macro SBX case is defined as MP(A/B/C).

Some existing XS operators can be approximately represented as SBX cases. One-point [30], two-point, and uniform (or position-based crossover [54]) order crossovers, are respectively similar to 1P/O/-, 2P/O/-, and U/O/-. LCS crossover [18] and similar block crossovers [49] differ from these simple order crossovers only in that they used more delicate base policies. The partially mapped crossover [54] and its uniform variant [7] are similar to 2P/P/PM and U/P/PM, respectively. The uniform like crossover (ULX) [61] and its optimized variant [29] are similar to U/P/R and MP(U/P/R), respectively.

4. Results and Discussion

All RTGO versions are coded in JAVA, and were run on a 1.4-GHz Opteron processor. By default, the maximum numbers of cycles (T) are fixed as 100 and 500 for FSP and QAP, respectively. For the parallel macro mode, there is $N_{ICG} = \mathrm{MAX}(\mathrm{INT}(0.05 \cdot n),\ 1)$, in which INT returns the closest integer value, MAX returns the larger value.

Table 1: The best-known upper bounds of the Taillard's dataset on 13-APR-2005 (UB05).

No.	01∼10	11∼20	21∼30	31∼40	41∼50	51∼60	61∼70	71∼80	81∼90
1	1278	1582	2297	2724	2991	3847	5493	5770	6202
2	1359	1659	2099	2834	2867	3704	5268	5349	6183
3	1081	1496	2326	2621	2839	3640	5175	5676	6271
4	1293	1377	2223	2751	3063	3719	5014	5781	6269
5	1235	1419	2291	2863	2976	3610	5250	5467	6314
6	1195	1397	2226	2829	3006	3679	5135	5303	6364
7	1234	1484	2273	2725	3093	3704	5246	5595	6268
8	1206	1538	2200	2683	3037	3691	5094	5617	6401
9	1230	1593	2237	2552	2897	3741	5448	5871	6275
10	1108	1591	2178	2782	3065	3756	5322	5845	6434

For FSP, experiments were performed on four benchmark datasets in OR-Library[1] [3], namely, Carlier's, Heller's , Reeves's, and Taillard's datasets. The Carlier's dataset contains 8 instances named Car1, Car2, through Car8. The Heller's dataset only possesses 2 instances, i.e., Hel1 and Hel2. The Reeves's dataset has 21 instances called Rec01, Rec03, through Rec41. The Taillard's dataset contains 90 instances from Tai01 through Tai90.

Table 1 lists the best-known upper bounds (f^*) of the Taillard's dataset lasted updated on 13-APR-2005 (or called as UB05)[2]. Besides, some researchers used the upper bounds reported by Taillard in 1993 (UB93) [59], or the upper bounds listed in 2004 (UB04) [49], etc. Basically, UB05 is much better than UB93, and slightly better than UB04.

For QAP, fifty commonly-used instances from QAPLIB [5] are used. The names and best-known upper bounds of the instances will be listed in the tables in Section 4.2.2.

There are two significant indices for measuring the performance of an algorithm. The first is the solution quality, which can be represented by relative percentage deviation (RPD) over the best-known upper bound (f^*). The second is running time (t_r) in seconds, which is counted at the cycle taken to reach the last improvement. Only t_r might be influenced by different machine configurations for running an algorithm. For the FSP and QAP instances, 10 and 50 independent runs were executed to obtain the mean results, respectively.

4.1. Effects of Idea Combination

During the search process, the idea combination process plays a navigating role for exploring the rugged permutation landscape by guiding the greedy local improvement process.

For comparison the differences between RTGO with different idea combination operators, RPD-t_r relations are used for examining the Pareto efficiency of both performance indices.

4.1.1. For FSP

For FSP, the effects of XS operators are evaluated on the Taillard's dataset. Moreover, 1P/O/-, i.e., the one-point order crossover [30], is considered as a standard XS operator for

[1] http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/flowshopinfo.html

[2] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt

the comparisons. For each case, RTGO with N=10, 20, 30, 40, and 50 were tested.

Three order-based operators (*/O/-), i.e., 1P/O/-, 2P/O/-, and U/O/-, have been frequently studied, where the difference is that they using different base policies from the viewpoint of SBX. Some researchers [30, 32] have claimed that 1P/O/- performs better than both 2P/O/- and U/O/-, However, some other researchers have declared [18] that 1P/O/- is the best among the above versions, and some advanced versions, such as the similar block order crossover (SBOX) [49], are modified from 1P/O/-.

Figure 2 gives the results by the RTGO versions using 1P/O/-, 2P/O/-, and U/O/-, respectively. If only RPD is concerned, contradicting conclusions may be achieved as for RTGO using different N values, where 2P/O/- has a lower RPD value than 1P/O/- as N=10, but has higher RPD values than 1P/O/- as N=20, 30, 40, and 50. However, it can be found that 1P/O/- has an overall better RPD-t_r performance than 2P/O/-. Moreover, U/O/- has the worst performance among all the three SBX operators, which may due to that the uniform mode preserves much less precedence information.

Figure 3 provides the results by the RTGO versions using 1P/O/-, 1P/P/O, and 1P/P/R, respectively. Here 1P/O/- performs better than the other two operators. Compared with 1P/P/*, 1P/O/- preserves more precedence information. For the repair operation, using the order-based mode is slightly better than using the random node. Thus for FSP, precedence information may be more significant than position information.

Figure 4 shows the results by the RTGO versions using 1P/O/-, CA1P/O/-, MP(1P/O/-) and MP(CA1P/O/-), respectively. Both CA1P/O/- and MP(1P/O/-) obtain better RPD-t_r performances than 1P/O/-. Specifically, CA1P/O/- achieves a much better RPD although it needs a longer running time, since the common-avoiding mode maintains larger information diversity among the agents, especially as N is smaller; while MP(1P/O/-) achieves a much shorter running time although it obtains a worse RPD, since the macro mode in the parallel mode leads to more promising states for the LS operator. MP(CA1P/O/-), which integrates of both improving strategies, produces better RPD-t_r performance than all three others.

4.1.2. For QAP

For QAP, the effects of XS operators are evaluated on the 50 instances in QAPLIB. Moreover, U/P/PM, i.e., the uniform partially mapped crossover [7], which is previously used for solving the largest common subgraph problem, is used as a standard XS operator for the comparisons. For each case, RTGO with N=10, 20, 30, 40, and 50 were tested.

Figure 5 gives the results by the RTGO versions respectively using U/P/PM, 1P/P/PM, and 2P/P/PM. As N is same, U/P/PM can achieve a lower RPD yet require a larger t_r than both 1P/P/PM and 2P/P/PM, which may due to that the U mode provides more possible choices than the 1P or 2P mode. Moreover, the base policy in 1P or 2P mode did not show an advantage in terms of the RPD-t_r relation.

Figure 6 shows the results by the RTGO versions using U/P/PM, U/O/-, and U/P/R, respectively. Both UP/P/R and UP/P/PM perform better than U/O/-. Here U/P/R represents the uniform like crossover (ULX) [61]. Compared with U/O/-, U/P/* preserves more position-based information. For the repair operation, It showed that using the PM mode is better than using the R mode, since the PM mode also preserves more position

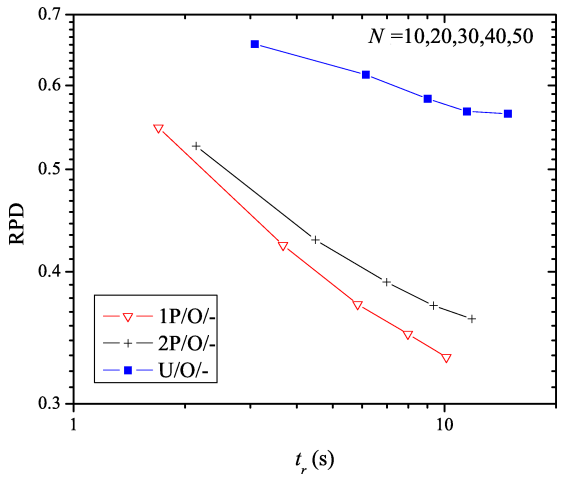

Figure 2: Results of the RTGO versions with SBX in different base policies.

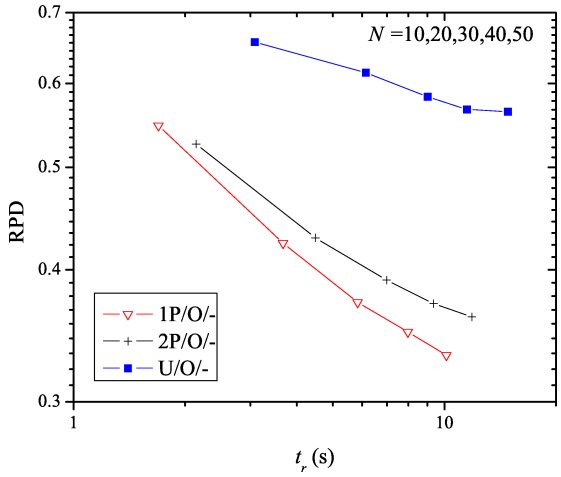

Figure 3: Results of the RTGO versions with SBX in different social and repair policies.

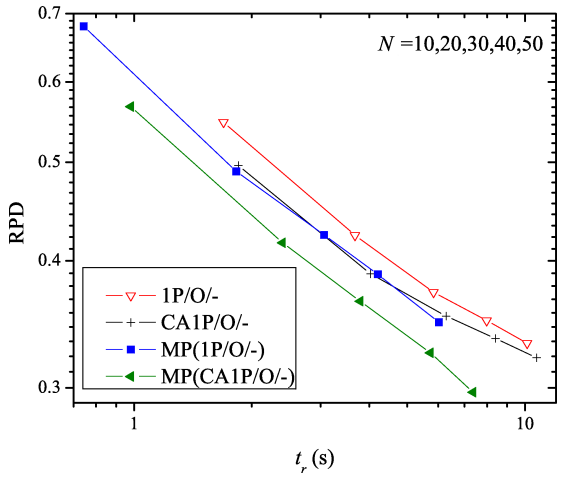

Figure 4: Results of the RTGO versions with some improving heuristics.

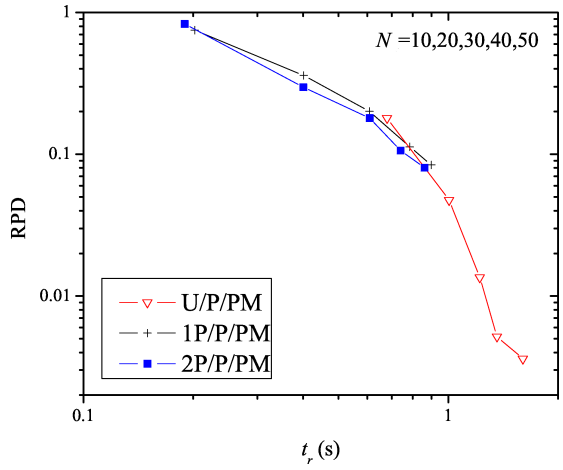

Figure 5: Results of the RTGO versions with SBX in different base policies.

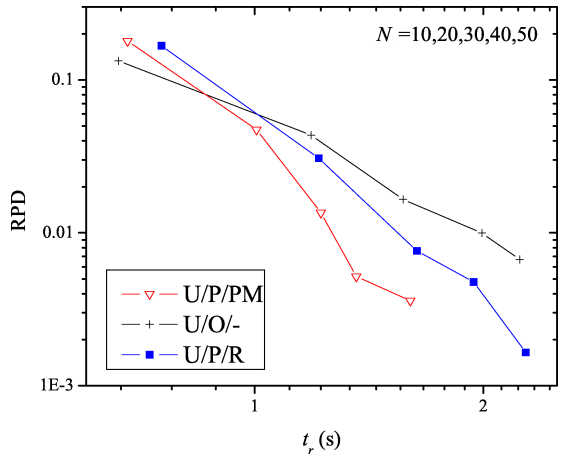

Figure 6: Results of the RTGO versions with SBX in different social and repair policies.

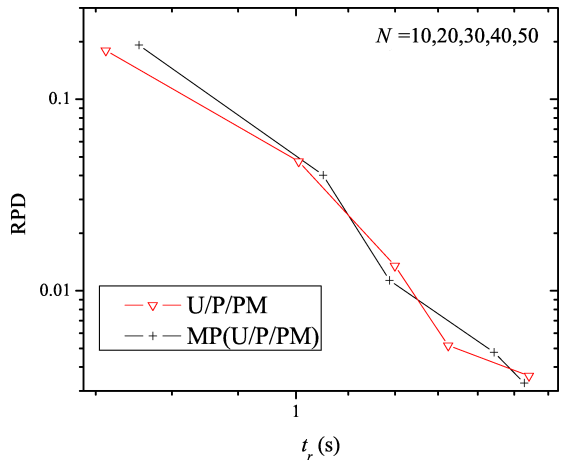

Figure 7: Results of the RTGO versions with U/P/PM and MP(UP/P/PM).

information. Thus for QAP, the position information might be more significant.

Figure 7 shows the results by the RTGO versions respectively using UP/P/PM and MP(UP/P/PM). Their performance are similar from the RPD-t_r relation.

4.2. Comparisons with Existing Algorithms

RTGO is coded in JAVA and most algorithms are coded in C/C++. Since the performance gap between JAVA and C++ is close, it only requires comparing the running times of different algorithms on different hardware configurations.

For simplicity, we calculate $r_{TR} = (t_{r,\mathrm{A}}/t_{r,\mathrm{B}}) \cdot (CR_{\mathrm{A}}/CR_{\mathrm{B}})$, in which $t_{r,\mathrm{A}}$ and $t_{r,\mathrm{B}}$ are the running times of the algorithms A and B, CR_A and CR_B are the clock rates of the processors for running the algorithms A and B, respectively. Moreover, the ratio of CPU clock rates is eliminated if no clock rate is provided for any of the algorithms. Here A and B are RTGO and the algorithm to be compared, respectively. In addition, a threshold value r_{MAX} is used for taking the factors of other configurations into account. For $r_{TR} \leq 1/r_{\mathrm{MAX}}$, $1/r_{\mathrm{MAX}} < r_{TR} < r_{\mathrm{MAX}}$, and $r_{TR} \geq r_{\mathrm{MAX}}$, RTGO are slower than, comparable to, and faster than the compared algorithm, respectively. In this paper, r_{MAX}=5 is used.

4.2.1. For FSP

In the following experiments, only the RTGO using MP(CA1P/O/-) is considered for comparing with some existing algorithms on the FSP benchmark datasets.

Tables 2 and 3 give the results by the RTGO with N=30, PSOMA [23], and HW-LS [17] on the Carlier's and Reeves's datasets, respectively. PSOMA [23] is a particle swarm optimization (PSO)-based memetic algorithm, which is coded in MATLAB 7.0, and was executed on a 2.2-GHz Mobile Pentium IV processor. HW-LS [17] is a LS method combined with two escape-from-trap procedures, which was run on a 500-MHz Pentium III processor.

Table 2 indicates that RTGO performed better than PSOMA for both RPD and running time on the Carlier's dataset. Both RTGO and HW-LS solved all instances in a 100% success rate, i.e., they are efficient in moving away from local minima.

Table 2: Results by RTGO, PSOMA [23], and HW-LS [17] on the Carlier's dataset.

Instance	n, m	f^*	RTGO(N=30)		PSOMA		HW-LS	
			RPD	t_r(s)	RPD	t_r(s)	RPD	t_r(s)
Car1	11,5	7038	0.000	0.008	0.000	0.68	0.00	0.010
Car2	13,4	7166	0.000	0.009	0.000	0.95	0.00	0.020
Car3	12,5	7312	0.000	0.015	0.000	1.06	0.00	0.020
Car4	14,4	8003	0.000	0.009	0.000	1.22	0.00	0.010
Car5	10,6	7720	0.000	0.011	0.018	0.70	0.00	0.020
Car6	8,9	8505	0.000	0.007	0.114	0.49	0.00	0.010
Car7	7,7	6590	0.000	0.006	0.000	0.30	0.00	0.010
Car8	8,8	8366	0.000	0.007	0.000	0.42	0.00	0.010
Average	-	-	0.000	0.009	0.017	0.73	0.00	0.014

Table 3: Results by RTGO, PSOMA [23], and HW-LS [17] on the Reeves's dataset.

Instance	n, m	f^*	RTGO(N=30)		PSOMA		HW-LS	
			RPD	t_r(s)	RPD	t_r(s)	RPD	t_r(s)
Rec01	20,5	1247	0.096	0.040	0.144	2.60	0.02	5.57
Rec03	20,5	1109	0.000	0.048	0.189	2.50	0.00	2.57
Rec05	20,5	1242	0.242	0.016	0.249	2.39	0.24	10.4
Rec07	20,10	1566	0.000	0.104	0.986	2.81	0.00	1.86
Rec09	20,10	1537	0.000	0.093	0.621	4.23	0.00	2.64
Rec11	20,10	1431	0.000	0.065	0.129	3.79	0.00	0.77
Rec13	20,15	1930	0.109	0.191	0.893	4.64	0.09	46.4
Rec15	20,15	1950	0.021	0.274	0.628	5.23	0.44	35.3
Rec17	20,15	1902	0.037	0.225	1.330	4.67	0.11	24.2
Rec19	30,10	2093	0.287	0.545	1.313	10.49	0.63	82.8
Rec21	30,10	2017	1.120	0.435	1.596	8.41	1.41	74.2
Rec23	30,10	2011	0.373	0.586	1.310	9.36	0.54	81.0
Rec25	30,15	2513	0.294	0.765	2.085	12.64	1.11	127
Rec27	30,15	2373	0.324	0.794	1.605	12.15	0.94	137
Rec29	30,15	2287	0.249	0.815	1.888	11.31	0.80	141
Rec31	50,10	3045	0.263	2.005	2.254	37.15	1.91	292
Rec33	50,10	3114	0.167	0.979	0.645	36.07	0.47	228
Rec35	50,10	3277	0.000	0.182	0.000	29.92	0.00	6.40
Rec37	75,20	4951	0.913	14.095	3.547	170.2	4.19	1700
Rec39	75,20	5087	0.698	11.966	2.426	155.7	2.90	1680
Rec41	75,20	4960	1.183	12.859	3.684	164.3	3.93	1910
Average	-	-	0.304	2.242	1.311	32.88	0.94	313.8

From Table 3, it can be found that RTGO produced much better than both PSOMA and HW-LS for both RPD and running time on the Reeves's dataset. The r_{TR} values of RTGO versus PSOMA and HW-LS are 127.5 and 0.6 for the Carlier's dataset, 23.0 and 50.0 for the Reeves's dataset, respectively. If considering the machine configurations, RTGO was not faster than HW-LS on the Carlier's dataset. However, RTGO was much faster than HW-LS on the Reeves's dataset. It may due to the Carlier's instances are easier than the Reeves's.

Table 4: Results by RTGO, DPSO2 [11], and NEH-ALA [1] on the Heller's dataset.

Instance	n, m	f^*	RTGO(N=30)		PSOMA		HW-LS	
			RPD	t_r(s)	RPD	t_r(s)	RPD	t_r(s)
Hel1	100,10	514	0.156	1.412	1.15	287	3.55	94
Hel2	20,10	135	0.000	0.153	1.08	67	0.39	2565

Table 4 compares the results by the RTGO with N=30, DPSO2 [11], and NEH-ALA [1] on the Heller's dataset. DPSO2 [11] is a discrete PSO hybridized with variable neighborhood

Table 5: Results by some existing algorithms on the Taillard's dataset.

Instance	HSA	PACO	ILS		HGA_RMA		PSO_{VNS}	
	RPD	RPD	RPD	t_r(s)	RPD	t_r(s)	RPD	t_r(s)
Tai01∼10	0.14	0.704	0.24	4.01	0.04	4.50	0.03	13.5
Tai11∼20	0.18	0.843	0.77	4.09	0.02	9.00	0.02	26.3
Tai21∼30	0.14	0.720	0.85	4.63	0.05	18.00	0.05	69.3
Tai31∼40	0.06	0.090	0.12	6.38	0.00	11.25	0.00	2.8
Tai41∼50	0.33	0.746	2.01	9.94	0.72	22.50	0.57	79.8
Tai51∼60	1.78	1.855	3.29	11.82	0.99	45.00	1.36	168.1
Tai61∼70	0.17	0.072	0.11	15.31	0.01	22.50	0.00	52.6
Tai71∼80	0.33	0.404	0.66	18.79	0.16	45.00	0.18	211.0
Tai81∼90	2.11	0.985	3.17	24.04	1.30	90.00	1.45	310.8
Average	0.58	0.713	1.25	11.00	0.37	29.75	0.41	103.8

Table 6: Results by RTGO with N=10, 30 and 50 on the Taillard's dataset.

Instance	n, m	RTGO(N=10)		RTGO(N=30)		RTGO(N=50)	
		RPD	t_r(s)	RPD	t_r(s)	RPD	t_r(s)
Tai01∼10	20,5	0.174	0.025	0.093	0.055	0.041	0.083
Tai11∼20	20,10	0.285	0.088	0.102	0.243	0.033	0.291
Tai21∼30	20,20	0.192	0.140	0.062	0.328	0.052	0.467
Tai31∼40	50,5	0.118	0.068	0.041	0.175	0.029	0.276
Tai41∼50	50,10	0.894	0.616	0.582	2.340	0.540	3.804
Tai51∼60	50,20	1.296	1.702	0.903	6.435	0.747	11.29
Tai61∼70	100,5	0.067	0.218	0.021	0.483	0.020	0.713
Tai71∼80	100,10	0.466	0.995	0.285	3.597	0.197	7.087
Tai81∼90	100,20	1.614	4.974	1.197	20.48	1.010	42.29
Average	-	0.567	0.981	0.365	3.793	0.297	7.367

search (VNS), and was executed on a 2.4-GHz Pentium IV processor. NEH-ALA [1] is an adaptive-learning approach in conjunction with the NEH heuristic [31], which is coded in Visual Basic 6.0 and was run on a 933-MHz processor. The r_{TR} values of RTGO versus DPSO2 and NEH-ALA are 387.8 and 1132.3, respectively. The results indicated that RTGO performs much better than DPSO2 and NEH-ALA on Hel1 and Hel2.

Table 5 summarizes the results by five existing algorithms, i.e., HSA [33], PACO [42], ILS [48, 55], HGA_RMA [49], and PSO_{VNS} [60] on the Taillard's dataset [59]. HSA is a population-based hybrid simulated annealing (SA). PACO is a newly-developed ant-colony optimization (ACO). ILS is an iterated local search [55], which is implemented by Ruiz and Maroto [48] in Delphi 6.0 and was run on a 1.4-GHz Athlon XP processor. HGA_RMA is a hybrid GA with the similar block order crossover (SBOX), which is coded in Delphi 7.0 and was run on a 2.8-GHz Pentium IV processor. For HGA_RMA, the running time is defined by the expression $n \cdot (m/2) \cdot 90$. PSO_{VNS} is coded in C and was run on a 2.6-GHz Pentium

Table 7: Results by RTGO with N=50 and five existing algorithms on the bur26* dataset.

Instance	f^*	RTGO	GRASP	ANT	HAS	IFLS	GA-1
bur26a	5426670	0.240	11.38	21.07	10	61.27	117.3
bur26b	3817852	0.313	59.45	35.03	17	60.27	112.7
bur26c	5426795	0.197	5.16	19.09	3.7	57.78	113.5
bur26d	3821225	0.285	15.12	19.4	7.9	61.27	106.7
bur26e	5386879	0.239	17.63	20.53	9.1	57.83	109.1
bur26f	3782044	0.151	5.05	11.23	3.4	59.19	102.2
bur26g	10117172	0.266	222.58	18.67	7.7	57.72	97.1
bur26h	7098658	0.152	37.58	5.67	4.1	57.47	101.9
Average	-	0.230	46.74	18.84	7.86	59.1	107.56

Table 8: Results by RTGO with N=50 and four existing algorithms on the esc* dataset.

Instance	f^*	RTGO	GRASP		IFLS	GA_{CT}	GA-1	
		t_r	RPD	t_r	t_r	t_r	RPD	t_r
esc32a	130	0.789	1.54	7.03	136.8	21.0	3.08	190.9
esc32b	168	0.094	0.00	2.80	110.4	18.0	0.00	200.1
esc32c	642	0.014	0.00	0.00	54.7	16.2	0.00	194.6
esc32d	200	0.031	0.00	1.92	74.3	16.8	0.00	176.3
esc32e	2	0.012	0.00	0.00	46.1	-	0.00	184.9
esc32f	2	0.012	0.00	0.00	44.5	-	0.00	184.4
esc32g	6	0.013	0.00	0.00	28.4	-	0.00	185.5
esc32h	438	0.047	0.00	3.41	85.8	17.4	0.00	174.5
esc64a	116	0.058	-	-	1521.7	183.0	0.00	1315.4

IV processor. Moreover, with regard to the upper bounds for calculating RPD values, HSA, PACO, and PSO_{VNS} use UB93 [59], while ILS and HGA_RMA use UB04 [49]. Hence, all the five algorithms in Table 5 should have larger RPD values if they use UB05.

Table 6 gives the results by the RTGO with N=10, 30, and 50, respectively. The r_{TR} values of RTGO with N=10 versus ILS, and RTGO with N=30 versus HGA_RMA and PSO_{VNS} are 11.2, 15.7, and 50.8, respectively. RTGO with N=10 performed better than HAS and PACO on RPD, and better than ILS on both RPD and running time. RTGO with N=30 produced better RPD and running time than both HGA_RMA and PSO_{VNS}. In addition, RTGO with N=50 obtained better RPD than RTGO with N=30, although the more agents is used, the more running time is required.

4.2.2. For QAP

In the following experiments, only the RTGO using MP(U/P/MP) is considered for comparing with some existing algorithms on the QAP benchmark datasets.

Different existing algorithms were applied on different instances. Thus the 50 QAPLIB instances are divided into five small datasets, i.e., bur26*, esc*, lipa*, tai*b, and misc*.

Table 9: Results by RTGO with N=50 and three existing algorithms on the lipa* dataset.

Instance	f^*	RTGO	ANT		IFLS		GA-1	
		t_r	RPD	t_r	RPD	t_r	RPD	t_r
lipa20a	3683	0.055	0.00	107.32	0.00	16.11	0.00	37.4
lipa30a	13178	0.313	0.00	54.85	0.00	119.72	0.00	172.3
lipa40a	31538	3.053	1.02	281.00	0.00	489.91	0.00	510.9
lipa50a	62093	10.225	-	-	1.02	1556.28	0.95	743.1
lipa20b	27076	0.027	0.00	0.00	0.00	16.78	0.00	37.2
lipa30b	151426	0.077	0.00	0.00	0.00	121.81	0.00	168.6
lipa40b	476581	0.203	0.00	0.00	0.00	485.98	0.00	513.2
lipa50b	1210244	0.469	-	-	0.00	1461.81	0.00	754.7

Table 10: Results by RTGO with N=50 and four existing algorithms on the tai*b dataset.

Instance	f^*	RTGO	CPTS	ANT_{SA}		HAS		GA_{HRR}
		t_r	t_r	RPD	t_r	RPD	t_r	t_r
tai20b	122455319	0.058	0.1	0.0000	27	0.0905	27	3.1
tai25b	344355646	0.204	0.4	0.0000	50	0.0000	12	5.6
tai30b	637117113	0.778	1.2	0.0000	90	0.0000	25	9.7
tai35b	283315445	1.015	2.4	0.0376	147	0.0256	147	15
tai40b	637250948	1.614	4.5	0.4872	240	0.0000	51	27
tai50b	458821517	7.581	13.8	0.2475	480	0.1916	480	49
tai60b	608215054	14.492	30.4	0.2258	855	0.0483	855	82

Some existing algorithms, such as GRASP [26], CPTS [19], ANT [26], ANT_{SA} [10], HAS [14], IFLS [43], GA_{CT} [12], GA-1 [2], and GA_{HRR} [28], have been applied on some QAPLIB instances. Here GRASP and CPTS are local-based search methods; ANT, ANT_{ST} and HAS are ant-based systems; IFLS, GA_{CT}, GA-1, and GA_{HRR} are genetic algorithms. Both GRASP and ANT are coded in Fortran 77 and were run on a 166-Mhz Pentium processor. CPTS, i.e., a cooperative parallel tabu search algorithm, is written in C and was run on ten 1.3-GHz Itanium processors. ANT_{SA}, or called AntSimulated [10], which uses simulated annealing as its LS, is coded in C. HAS is a hybrid ant colony system. IFLS is coded in JAVA, and was run on a 2.4GHz Athlon XP processor. GA_{CT} is a hybrid GA with a concentric tabu search [12], which is coded in Fortran, and was tested on a 600-MHz Pentium III processor. GA-1 is a greedy GA. GA_{HRR} is a GA hybridized with ruin and recreate procedure.

For an algorithm, the RPD column is not listed if the algorithm can achieved 100% success rate for all the tested instances. The calculation of an r_{TR} value for two algorithms is applied only on the instances tested by both the related algorithms.

Table 7 compares the results by RTGO with N=50, GRASP [26], ANT [26], HAS [14], IFLS [43], and GA-1 [2] on the bur26* dataset. Only the running times are listed, since all these algorithms achieved 100% success rate. The r_{TR} values of RTGO versus the five algorithms are 24.1, 9.7, 34.2, 440.5, and 467.7, respectively.

Table 11: Results by RTGO with N=100 and three existing algorithms on the misc* dataset.

Instance	f^*	RTGO		ANT		IFLS		GA-1	
		RPD	t_r	RPD	t_r	RPD	t_r	RPD	t_r
chr20a	2192	0.000	1.544	0.00	331.20	4.38	10.95	0.18	47.3
chr20c	14142	0.000	0.186	0.00	29.49	0.00	13.55	4.72	48.9
chr22a	6156	0.000	1.077	0.00	314.68	0.88	19.11	0.62	72.9
chr22b	6194	0.000	1.770	0.97	161.75	1.68	17.00	1.19	76.1
chr25a	3796	0.000	1.264	0.00	236.29	11.17	33.59	10.54	96.8
had20	6922	0.000	0.039	0.00	158.71	0.00	10.58	-	-
kra30a	88900	0.000	0.807	0.00	199.06	1.34	105.55	1.34	150.7
kra30b	91420	0.000	1.731	0.00	140.02	0.13	101.83	0.18	165.3
mc33	339416	0.000	1.019	0.00	379.65	-	-	-	-
nug20	2570	0.000	0.109	0.00	119.28	0.00	16.06	0.00	48.9
nug30	6124	0.000	1.610	0.00	180.75	2.12	116.66	0.07	177.1
rou20	725522	0.002	2.171	0.00	244.54	0.02	11.73	0.08	37.6
scr20	110030	0.000	0.190	0.00	46.09	0.00	12.69	0.03	39.8
sko42	15812	0.000	8.458	-	-	0.30	613.92	0.23	503.1
ste36a	9526	0.000	7.249	0.76	295.23	0.00	204.36	1.47	354.8
ste36b	15852	0.000	1.319	0.25	212.81	3.43	222.45	-	-
tho30	149936	0.000	1.400	0.00	287.50	0.29	118.94	0.31	197.8
tho40	240516	0.008	26.372	0.66	312.46	0.53	501.77	0.33	479.1

Table 8 shows the results by RTGO with N=50, GRASP [26], IFLS [43], GA$_{CT}$ [12], and GA-1 [2] on the esc* dataset. Here RTGO, IFLS, and GA$_{CT}$ obtained a 100% success rate, whereas GRASP and GA-1 could not fully solve esc32a. The r_{TR} values of RTGO versus the four algorithms are 1.8, 3368.8, 113.0, and 2623.0, respectively. For r_t, RTGO was comparable to GRASP, but much faster than IFLS, GA$_{CT}$, and GA-l, on the esc* dataset.

Table 9 compares the results by the RTGO with N=50, ANT [26], IFLS [43], and GA-1 [2] on the lipa* dataset. Here only RTGO obtained a 100% success rate. The r_{TR} values of RTGO versus the four algorithms are 14.1, 507.4, and 203.7, respectively.

Table 10 shows the results by RTGO with N=50, CPTS [19], ANT$_{SA}$ [10], HAS [14], and GA$_{HRR}$ [28] on the tai*b dataset. Here RTGO, CPTS, and GA$_{HRR}$ obtained a 100% success rate. The r_{TR} values of RTGO versus the four algorithms are 1.9, 73.4, 62.0, and 7.4, respectively. RTGO was comparable with CPTS, but much faster than ANT$_{SA}$ and HAS. Moreover, it should be noticed that CPTS is a parallel algorithm with ten processors. Naturally, the performance of RTGO can also be improved significantly by assigning agents to different processors due to the inherent parallelism in a cooperative group.

Table 11 gives the results by RTGO with N=100, ANT [26], IFLS [43], and GA-1 [2] on the misc* dataset. The r_{TR} values of RTGO versus the three algorithms are 8.7, 63.8, and 44.6, respectively. Furthermore, RTGO achieved a 100% success rate for all 18 instances except for rou20 and tho40. In terms of RPD, RTGO also performed much better.

4.3. Discussion

Due to its simplicity, RTGO might be interpreted straightforwardly from the viewpoint of an agent. Over sessions, each agent can be seen as walking in the rugged problem landscape, and the current location is its base idea (its best-so-far solution). Each stable LS strategy can reach to a (near) local optimum but cannot escape from the current one, whereas the round-table group support mechanism can only provides diverse social ideas. Thus only the XS strategy provides the agent a capability of escaping from local minima. With an SBX operator, the base idea is adaptively modified by the social and repair policies using the guidance clues in the social idea. In contrast, iterated local search (ILS) [50, 55] can only use blind perturbations to have a randomized walk in the space of local optima. RTGO might also benefit from the cooperative portfolio effect [16] on the agents. Compared to genetic local search [63] and memetic algorithms (MA) [27], each agent in RTGO has its private memory, which allow a natural way to perserve diverse ideas for possible improvements.

As shown in Section 4.1, good performance can be achieved by using appropriate SBX operators, for different sequencing problems. The modular design of SBX might help us to roughly identify some structure features of different problems. FSP and QAP respectively prefer more precedence and position information. The common-avoiding style does help on the 1P base policy, and the parallel macro policy has more usage on FSP that on QAP.

From the results shown in Section 4.2, RTGO can achieve better performance than many existing methods, including some metaheuristic methods that using the same LS strategies, e.g., HGA_RMA [49] and ILS [55] for FSP, and HAS [14] and IFLS [43] for QAP.

RTGO is a preliminary step in studying and utilizing the group creativity from the viewpoint of a metaheuristic framework. We have shown that group creativity can emerge from a group of agents with very limited memory and thinking capability. The research in this direction might help for unravelling the real-world complexity, given that large-scale experiments in human groups might be too expensive and might have too much uncertainty.

5. Conclusion

The round-table group optimizer (RTGO) is a very simple realization of a cooperative idea-generating group. The group-level support is a round-table mechanism for providing one social idea for each agent, and each agent only stores the best-so-far idea as the base idea in its individual memory. Given the base and social ideas, the idea combination and local improvement processes of each agent are respectively realized by using a XS strategy and a LS strategy, in the form of socially biased learning. The agents are able to search in the problem landscape in parallel based on their individual memory, as well as utilize the stimulation from social ideas for achieving a collective performance, over iterative sessions.

The implementation of RTGO was performed on two sequencing problems, i.e., FSP and QAP. Two stable, domain-specific LS strategies were adopted from existing algorithms. Then a general XS class, called socially biased combination (SBX), was realized in a modular form: A basic SBX operator contains three policies, i.e., base, social and repair policies, and a macro SBX operator can be realized by applying a macro policy on any SBX operator(s).

We then evaluated the performance of the RTGO metaheuristic framework on some commonly-used FSP and QAP benchmark datasets. The effects of idea combination processes were evaluated on different SBX realizations. RTGO outperformed many existing methods, including some methods that using the same LS strategies, in terms of the solution quality and the running time. The results might also indicate that appropriate SBX operators have the capability of leaping adaptively in the problem landscape, while the diversity of promising ideas can be well-preserved by the simple RTGO framework.

There are several aspects of the proposed method that warrant further study. One issue concerns the development of more effective search strategies. For example, adaptive memory strategies [64] rather than a stable LS strategy might be used to overcome local minima. The group metaphor is very open for efficient computation using any sophisticated operators.

Moreover, RTGO variants might be designed for promoting group creativity. For example, each agent might possess a pool of ideas in its memory [21], so that the agent has a strong individual learning capability and can adaptively using the base and social ideas in different sessions. Each agent might also hold multiple search strategies that can be automatic configured by a hands-off learning procedure for handling complex problem domains.

References

[1] A. Agarwal, S. Colak, E. Eryarsoy, Improvement heuristic for the flow-shop scheduling problem: an adaptive-learning approach, European Journal of Operational Research 169 (2006) 801–815.

[2] R. Ahuja, J. Orlin, A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, Computers and Operations Research 27 (2000) 917–934.

[3] J. Beasley, OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41 (1990) 1069–1072.

[4] K.R. Betts, V.B. Hinsz, Collaborative group memory: Processes, performance, and techniques for improvement, Social and Personality Psychology Compass 4 (2010) 119–130.

[5] R. Burkard, S. Karisch, F. Rendl, QAPLIB - a quadratic assignment problem library, Journal of Global Optimization 10 (1997) 391–403.

[6] J. Carlier, I. Rebai, Two branch and bound algorithms for the permutation flow shop problem, European Journal of Operational Research 90 (1996) 238–251.

[7] V.A. Cicirello, S.F. Smith, Modeling ga performance for control parameter optimization, Genetic and Evolutionary Computation Conference, Morgan Kaufmann, Las Vegas, NV, USA, 2000, pp. 235–242.

[8] A. Cropley, In praise of convergent thinking, Creativity Research Journal 18 (2006) 391–404.

[9] É. Danchin, L.A. Giraldeau, T. Valone, R. Wagner, Public information: From nosy neighbors to cultural evolution, Science 305 (2004) 487–491.

[10] N. Demirel, M. Toksari, Optimization of the quadratic assignment problem using an ant colony algorithm, Applied Mathematics and Computation 183 (2006) 427–435.

[11] L. Deroussi, M. Gourgand, S. Kemmoe, A. Quilliot, Discrete particle swarm optimization for the permutation flow shop problem, IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, IMACS, 2005, pp. T2–I–91–0824.

[12] Z. Drezner, Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem, Computers & Operations Research 35 (2008) 717–736.

[13] B.G. Galef, Why behaviour patterns that animals learn socially are locally adaptive, Animal Behaviour 49 (1995) 1325–1334.

[14] L. Gambardella, E. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problem, Journal of the Operational Research Society 50 (1999) 167–176.

[15] A.M. Glenberg, What memory is for, Behavioral and Brain Sciences 20 (1997) 1–55.

[16] C. Gomes, B. Selman, Algorithm portfolios, Artificial Intelligence 126 (2001) 43–62.
[17] W. Huang, L. Wang, A local search method for permutation flow shop scheduling, Journal of the Operational Research Society 57 (2006) 1248–1251.
[18] S. Iyer, B. Saxena, Improved genetic algorithm for the permutation flowshop scheduling problem, Computers & Operations Research 31 (2004) 593–606.
[19] T. James, C. Rego, F. Glover, A cooperative parallel tabu search algorithm for the quadratic assignment problem, European Journal of Operational Research 195 (2009) 810–826.
[20] S. Khalouli, F. Ghedjati, A. Hamzaoui, An ant colony system algorithm for the hybrid flow-shop scheduling problem, International Journal of Applied Metaheuristic Computing 2 (2011) 29–43.
[21] N.W. Kohn, P.B. Paulus, Y.H. Choi, Building on the ideas of others: An examination of the idea combination process, Journal of Experimental Social Psychology 47 (2011) 554–561.
[22] M. Koivisto, P. Parviainen, A space-time tradeoff for permutation problems, in: Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, Austin, TX, USA, 2010, pp. 484–492.
[23] B. Liu, L. Wang, Y. Jin, An effective PSO-based memetic algorithm for flow shop scheduling, IEEE Transactions on Systems Man and Cybernetics Part B - Cybernetics 37 (2007) 18–27.
[24] J. Liu, K.C. Tsui, Toward nature-inspired computing, Communications of the ACM 49 (2006) 59–64.
[25] W.S. Lovejoy, A. Sinha, Efficient structures for innovative social networks, Management Science 56 (2010) 1127–1145.
[26] V. Maniezzo, A. Colorni, The ant system applied to the quadratic assignment problem, IEEE Transactions on Knowledge and Data Engineering 11 (1999) 769–778.
[27] P. Merz, B. Freisleben, Fitness landscape analysis and memetic algorithms for the quadratic assignment problem, IEEE Transactions on Evolutionary Computation 4 (2000) 337–352.
[28] A. Misevicius, A modified simulated annealing algorithm for the quadratic assignment problem, Informatica 14 (2003) 497–514.
[29] A. Misevicius, An improved hybrid genetic algorithm: new results for the quadratic assignment problem, Knowledge-Based Systems 17 (2004) 65–73.
[30] T. Murata, H. Ishibuchi, H. Tanaka, Genetic algorithms for flowshop scheduling problems, Computers & Industrial Engineering 30 (1996) 1061–1071.
[31] M. Nawaz, E. Enscore, I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA 11 (1983) 91–95.
[32] A. Nearchou, The effect of various operators on the genetic search for large scheduling problems, International Journal of Production Economics 88 (2004) 191–203.
[33] A. Nearchou, A novel metaheuristic approach for the flow shop scheduling problem, Engineering Applications of Artificial Intelligence 17 (2004) 289–300.
[34] C.J. Nemeth, Differential contributions of majority and minority influence, Psychological Review 93 (1986) 23–32.
[35] A. Newell, H. Simon, Human Problem Solving, Prentice-Hall, NJ, 1972.
[36] B.A. Nijstad, W. Stroebe, How the group affects the mind: A cognitive model of idea generation in groups, Personality and Social Psychology Review 10 (2006) 186–213.
[37] E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the permutation flow shop problem, European Journal of Operational Research 91 (1996) 160–175.
[38] M.A. Montes de Oca, T. Stützle, K. Van den Enden, M. Dorigo, Incremental social learning in particle swarms, IEEE Transactions on Systems, Man, and Cybernetics, Part B 41 (2011) 368–384.
[39] A.F. Osborn, Applied Imagination: Principles and Procedures of Creative Problem Solving, Charles Scribner's Sons, New York, 1953.
[40] P.B. Paulus, Groups, teams, and creativity: the creative potential of idea-generating groups, Applied Psychology: an International Review 49 (2000) 237–262.
[41] V.L. Putman, P.B. Paulus, Brainstorming, brainstorming rules and decision making, The Journal of Creative Behavior 43 (2009) 23–39.
[42] C. Rajendran, H. Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, European Journal of Operational Research 155 (2004) 426–438.

[43] A. Ramkumar, S. Ponnambalam, N. Jawahar, R. Suresh, Iterated fast local search algorithm for solving quadratic assignment problems, Robotics and Computer-Integrated Manufacturing 24 (2008) 392–401.
[44] C. Reeves, An improved heuristic for the quadratic assignment problem, Journal of the Operational Research Society 36 (1985) 163–167.
[45] C. Reeves, A genetic algorithm for flowshop sequencing, Computers & Operations Research 22 (1995) 5–13.
[46] E.F. Rietzschel, B.A. Nijstad, W. Stroebe, The selection of creative ideas after individual idea generation: Choosing between creativity and impact, British Journal of Psychology 101 (2010) 47–68.
[47] E.M. Rogers, Diffusion of Innovations, Free Press, New York, NY, 2003.
[48] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2005) 479–494.
[49] R. Ruiz, C. Maroto, J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problem, OMEGA 34 (2006) 461–476.
[50] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (2007) 2033–2049.
[51] D.L. Rulke, J. Galaskiewicz, Distribution of knowledge, group network structure, and group performance, Management Science 46 (2000) 612–625.
[52] H. Saremi, B. Abedin, A. Kermani, Website structure improvement: quadratic assignment problem approach and ant colony meta-heuristic technique, Applied Mathematics and Computation 195 (2008) 285–298.
[53] J.W. Satzinger, M.J. Garfield, M. Nagasundaram, The creative process: the effects of group memory on individual idea generation, Journal of Management Information Systems 15 (1999) 143–160.
[54] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley, A comparison of genetic sequencing operators, in: International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, pp. 69–76.
[55] T. Stützle, Applying iterated local search to the permutation flow shop problem, Technical Report AIDA-98-04, FG Intellektik, TU Darmstadt, 1998.
[56] G. Syswerda, L. Davis, Schedule optimization using genetic algorithms, Schedule optimization using genetic algorithms, 1991, pp. 332–349.
[57] É. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research 47 (1990) 67–74.
[58] É. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17 (1991) 443–455.
[59] É. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278–285.
[60] M. Tasgetiren, Y. Liang, M. Sevkli, G. Gencyilmaz, A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, European Journal of Operational Research 177 (2007) 1930–1947.
[61] D. Tate, A. Smith, A genetic approach to the quadratic assignment problem, Computers and Operations Research 22 (1995) 73–83.
[62] X.F. Xie, J. Liu, Multiagent optimization system for solving the traveling salesman problem (TSP), IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 39 (2009) 489–502.
[63] X. Xu, Z. Xu, X. Gu, An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization, Expert Systems with Applications 38 (2011) 7970–7979.
[64] P.Y. Yin, F. Glover, M. Laguna, J.X. Zhu, Cyber swarm algorithms - improving particle swarm optimization using adaptive memory strategies, European Journal of Operational Research 201 (2010) 377–389.
[65] L. Yu, J. Nickerson, Cooks or cobblers?: crowd creativity through combination, in: Annual Conference on Human Factors in Computing Systems, ACM, Vancouver, BC, Canada, 2011, pp. 1393–1402.