A Multi-Agent Based Optimization Method Applied to the Quadratic Assignment Problem

Ines Sghir[a,b], Jin-Kao Hao[a,*], Ines Ben Jaafar[b], Khaled Ghédira[b]

[a]*LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, Cedex 01, France*
[b]*SOIE, ISG, Université de Tunis, Cité Bouchoucha 2000 Le Bardo, Tunis, Tunisie*

Abstract

Inspired by the idea of interacting intelligent agents of a multi-agent system, we introduce a multi-agent based optimization method applied to the quadratic assignment problem (MAOM-QAP). MAOM-QAP is composed of several agents (decision-maker agent, local search agents, crossover agents and perturbation agent) which are designed for the purpose of intensified and diversified search activities. With the help of a reinforcement learning mechanism, MAOM-QAP dynamically decides the most suitable agent to activate according to the state of search process. Under the coordination of the decision-maker agent, the other agents fulfill dedicated search tasks. The performance of the proposed approach is assessed on the set of well-known QAP benchmark instances, and compared with the most advanced QAP methods of the literature. The ideas proposed in this work are rather general and could be adapted to other optimization tasks. This work opens the way for designing new distributed intelligent systems for tackling other complex search problems.

Keywords: Multi-agent based optimization; cooperative search; heuristics; quadratic assignment; combinatorial optimization.

---

1. Introduction

The quadratic assignment problem (QAP) is one of the most popular combinatorial optimization problems with a number of practical applications like planning, backboard wiring in electronics, analysis of chemical reactions for organic compounds, design of typewriter keyboards balancing turbine runners (Burkard, 1991; Duman & Or, 2007). The QAP is known to be computationally difficult since it belongs to the class of NP-hard problems (Sahni & Gonzalez, 1976).

QAP was initially introduced to formulate the location of a set of indivisible economical activities. Given a flow f_{ij} from facility i to facility j for all i, j in

*Corresponding author: jin-kao.hao@univ-angers.fr
Email addresses: `inessghir@gmail.com` (Ines Sghir), `Ines.BenJaafar@gmail.com` (Ines Ben Jaafar), `khaled.ghedira@isg.rnu.tn` (Khaled Ghédira)

$\{1, 2, ...n\}$ and a distance d_{ab} between locations a and b for all a, b in $\{1, 2, ...n\}$, the QAP is to assign the set of n facilities to the set of n locations while minimizing the sum of the products of the flow and distance matrices. Let Π be the set of the permutation functions π: $\{1, 2, ...n\} \rightarrow \{1, 2, ...n\}$. The QAP is mathematically formulated as follows:

$$Minimize_{\pi \in \Pi} F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\pi_i \pi_j} \tag{1}$$

The computational challenge of the QAP has motivated many solution approaches including exact methods like (Erdogan & Tansel, 2007; Hahn, Grant, & Hall , 1998) and numerous heuristic methods. Among the most representative heuristic methods, we can mention the popular robust tabu search algorithm (Ro-TS) (Taillard, 1991), the memetic algorithm (Merz & Freisleben, 2000), the improved hybrid genetic algorithm (IHGA) (Misevicius, 2004), the iterated tabu search algorithm (ITS) (Misevicius, Lenkevicius, & Rubliauskas, 2006), the population-based iterated local search (PILS) (Stützle, 2006), the hybrid genetic algorithm MRT (Drezner, 2008), the cooperative parallel tabu search algorithm (CPTS) (James, Rego, & Glover, 2009), the breakout local search (BLS) (Benlic & Hao, 2013) and the memetic search algorithm (BMA) (Benlic & Hao, 2015). These methods generally perform well on a number of benchmark instances. Yet, no single method clearly dominates all other methods.

In this work, we investigate a new solution approach for the QAP based on the principles of multi-agent systems (MAS). Our work is motivated by appealing features of a MAS which could be advantageously used to elaborate intelligent computing systems (Martin, Ouelhadj, Smetb, Beullens, & Özcan, 2013; Guo, Goncalves, & Hsu, 2013; Gonçalves, Guimarães, & Souza, 2014; Satunin & Babkin, 2014; Couellan, Jan, Jorquera, & Georgé, 2015; Baykasoğlu & Kaplanoğlu, 2015; Wang & Wang, 2015; Zheng & Wang, 2015). Compared with the existing studies on the QAP, this work has the following main contributions:

- The proposed algorithm is the first distributed method for the QAP that adopts multi-agent systems as a source of inspiration for optimization.
- The proposed algorithm integrates a set of collaborative agents (local search agents, crossover agents, perturbation agent) which are managed dynamically by a distributed model to ensure a suitable balance of intensification and diversification of the given search space.
- Decision making is based on reinforcement learning which is used to adjust the probability of applying dedicated actions to trigger specific agents under specific conditions.
- We show the viability of the proposed approach by presenting computational results on the set of 135 well-known QAP benchmark instances.
- The proposed approach is general and could be adapted to design distributed intelligent systems for other complex search problems.

The rest of the paper is organized as follows. Section 2 is dedicated to literature review. Section 3 describes the proposed distributed algorithm. Section 4 shows computational results and comparisons with representative QAP algorithms of the literature. An analysis of the proposed algorithm is also provided. In the last section, we provide concluding comments and research perspectives.

2. Literature review

In this section, we first present a brief summary of some of the most representative heuristic algorithms for the QAP. These algorithms will be used as reference methods for our computational study. Note that none of these QAP approaches can be considered as the most effective method for all QAP benchmark instances, due to the differences in structures of the QAP benchmark instances. We also provide a literature review of some recent applications of multi-agent systems for solving search problems.

The robust tabu search (Ro-TS) algorithm proposed by Taillard (1991) is an early and influential heuristic. Ro-TS employs the swap move which exchanges two elements of a solution (a permutation). The tabu list forbids the reverse exchange of a swap move during the next h iterations. The tabu tenure h varies randomly within a given interval. The most important new feature introduced in Ro-TS is that a complete swap neighborhood is explored in $O(n^2)$ instead of $O(n^3)$ as in previous algorithms. We use this technique in our algorithm.

The improved hybrid genetic algorithm (IHGA) is presented by Misevicius (2004). IHGA integrates a robust local improvement procedure and a new optimized crossover. The optimized crossover uses M runs of an uniform crossover to produce a child that has the best fitness value. The offspring is then improved with a tabu search procedure and a solution reconstruction procedure. The reconstruction is achieved by performing a number of random swaps. IHGA uses also a shift mutation, which simply shifts all the items of the solution in a wrap-around fashion by a predefined number of positions. Later Misevicius et al. proposed an iterated tabu search (ITS) (Misevicius, Lenkevicius, & Rubliauskas, 2006) which iterates between a traditional tabu search and a perturbation phase in order to escape an attained local optimum.

The particular population-based iterated local search (PILS) proposed by Stützle (2006) is an extension of iterated local search (ILS). The algorithm applies the don't look bit strategy, inspired by local search algorithms for the TSP. When a local optimum is attained, ILS executes a perturbation move that exchanges k randomly chosen items. In PILS, the population contains p solutions and in each iteration q new solutions are generated. The new population of p solutions is created from the p former solutions and the q new solutions.

The cooperative parallel tabu search algorithm (CPTS) introduced by James, Rego, & Glover (2009) applies in parallel several tabu search (TS) runs on multiple processors. The TS procedure is the same as Ro-TS (Taillard, 1991), but uses different stopping conditions and tabu tenures for each processor. The cooperation and information exchanges between the TS processes are realized with the help of a global reference set.

The Breakout Local Search (BLS) described by Benlic & Hao (2013) is based on a local search phase and a dedicated perturbation phase. The local search phase aims to reach new local optima while the perturbation phase is used to discover new promising regions. The perturbation mechanism of BLS dynamically determines the number of perturbation moves and adaptively chooses between two types of moves of different intensities depending on the current search state. Perturbations are either guided by using a tabu list or simply based on random moves. BLS is later integrated into the memetic search framework in Benlic & Hao (2015). BMA combines BLS as local optimizer, a crossover operator, a pool updating strategy, and an adaptive mutation mechanism. BMA outperforms its local search component (BLS).

In this work, we introduce a new multi-agent optimization method for the QAP (MAOM-QAP) inspired by multi-agent systems. The proposed method is motivated by specific features offered by MAS like distributed computing, agent cooperation and dynamic decision making. Indeed, multi-agent systems have been successfully applied to solve many challenging and divers problems encountered in various settings. The review below, which is by no means exhaustive, aims to describe some recent MAS-related studies to illustrate the interest of MAS for building expert and intelligent systems for problem solving.

In (Gonçalves, Guimarães, & Souza, 2014), the authors presented an evolutionary multi-agent system to solve the join ordering optimization problem of queries in relational database management systems in a non-distributed environment. For this, they defined a working environment composed by a set of collaborative agents, where each agent is designed to find the best solution, i.e. the best join order for the relations in a query. Interesting results are reported with the proposed approach.

Satunin & Babkin (2014) tackled a challenging design problem raised in flexible public transportation systems, i.e., the design of demand responsive transport systems (DRT) which aims to provide a share transportation services with flexible routes and focus on optimizing economic and environmental values. The proposed approach uses a distributed multi-agent system to model DRT where various autonomous agents represent interests of systems stakeholders. The authors reported very interesting results with the proposed approach.

Baykasoğlu & Kaplanoğlu (2015) developed a multi-agent based approach for a load/truck planning problem in transportation logistics. The proposed approach is characterized by its cooperative structure which is motivated by real-world third party logistics company operations and uses negotiation mechanisms among the agents to handle the dynamic events. The solutions obtained by using the proposed approach demonstrate the usefulness of the approach in providing high-quality solutions while generating real-time schedules.

Couellan, Jan, Jorquera, & Georgé (2015) are interested in solving challenging optimization problems raised in training problems of Support Vector Machines (SVM). They observe that multi-agents systems are able to break down a complex optimization problem into elementary oracle tasks which are solved by performing a collaborative solution process. Based on this observation, they proposed a multi-agent system to solve the basic SVM training problem and provide several perspectives for binary classification, hyperparameters selection, multiclass learning as well as unsupervised learning.

Zheng & Wang (2015) proposed a multi-agent optimization algorithm for solving the resource-constrained project scheduling problem. The proposed algorithm uses multiple agents working in a grouped environment where each agent represents a feasible solution. The evolution of agents is achieved by using four main elements, i.e., social behavior, autonomous behavior, self-learning, and environment adjustment. The comparisons to the existing algorithms demonstrate the effectiveness of the proposed algorithm.

Wang & Wang (2015) considered the scheduling problem of the final testing process which ensures the quality of the products in the semiconductor manufacturing factory. They presented an effective knowledge-based multi-agent evolutionary algorithm where each agent represents a solution, which is a combination of the operation sequence vector and the machine assignment vector. Agents evolve by mutual-learning and competition based on a model of agent lattice. A knowledge base is employed to store the useful information during the search process for the purpose of generating new agents in the competition phase.

Our work shares similarities with these previous studies in the sense that it is based on the general framework of multi-agent systems. On the other hand, the proposed work, as described in the next section, distinguishes itself by some particular features including the distributed and collaborative architecture, the design of both intensification and diversification agents as well as the decision making method based on reinforcement learning.

3. A multi-agent optimization method for the QAP

A multi-agent system (MAS) is typically composed of a group of interacting agents where each agent has one or more basic skills (Weiss, 2013). The agents of a MAS can collectively find solutions to a difficult problem even if each agent alone cannot solve the problem. In this work, we are interested in using MAS as a source of inspiration to create a Multi-Agent Optimization Method applied to the QAP (MAOM-QAP).

3.1. MAOM-QAP architecture

The proposed MAOM-QAP model contains the following agents: decision maker agent, local search agents (and their perturbation agent) and crossover agents. Figure 1 illustrates the general MAOM-QAP architecture, whose components are presented in the following sections. Algorithm 1 describes the general MAOM-QAP procedure. In addition to the above agents, the MAOM-QAP model relies on learning-based weight matrices for decision making. By linking a set of conditions and a set of actions, such a matrix helps a agent to know the agents with which it will communicate, according to the state of the search process.

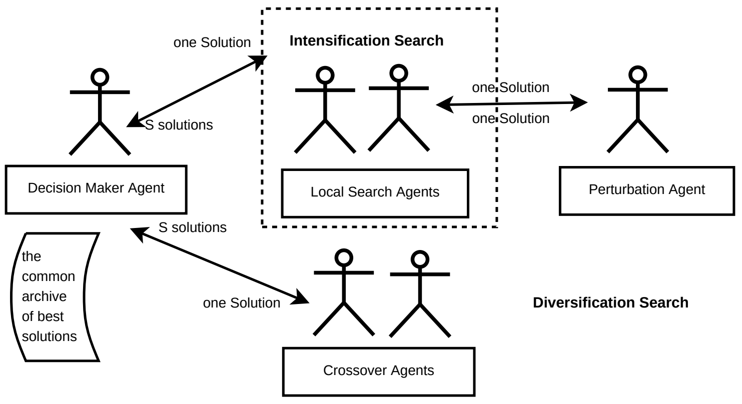

Figure 1: Agent communication in MAOM-QAP

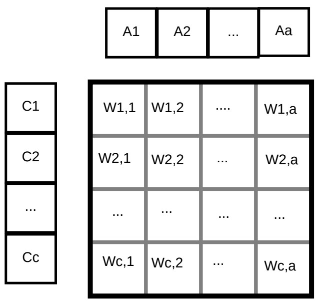

Figure 2: Structure of the weight matrix

3.2. Weight matrix with reinforcement learning

In the proposed MAOM model, some agents (decision maker agent and local search agents) need to decide when to activate other agents and which agents to activate. Such decisions are made based on a weight matrix (i.e., decision matrix) which is dynamically adjusted by a reinforcement learning process. This technique allows to adapt the search strategy according to the experiences acquired during the search process. For instance, after the application of an intensification action (e.g., ensured by a local search agent), if the search is observed to be stagnating (e.g., captured by the condition 'the best solution is not improved for a high number of iterations'), the applied action should be avoided for the next search step and an action ensuring more diversification should be favored. Inversely, if the applied action leads to a search progress (e.g., captured by the condition 'the best solution is just improved'), the same action should be given a high chance to be applied again (notion of reward).

Following Guo, Goncalves, & Hsu (2013), we use a pair $(condition, action)$ to represent the decision rules. The *condition* part corresponds to the necessary

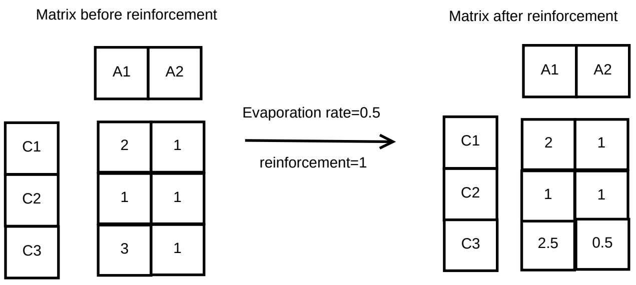

Figure 3: An example of reinforcement learning with the weight matrix. We suppose that the current condition is C3 (e.g., the local best solution has been improved during the last 20 generations). Under this condition, action A1 (e.g., to activate local search agents) is applied for the current generation (this action has the highest weight value in the matrix) and obtains a further improvement. Then, reinforcement is applied by adjusting the weight $W31$ to augment the chance of selecting again the applied action under this condition (e.g., $W31 = 3 \times 0.5 + 1 = 2.5$). Meanwhile, the weight $W32$ is decreased by μ (e.g., $W32 = 1 \times 0.5 = 0.5$)

prerequisite to trigger an associated action, the *action* part indicates which action is to be performed. Let C be the set of conditions and A the set of actions to perform. For a condition C_i, a weight W_{ij} (initialized to 0) is associated to each action A_j. The conditions are defined based on the improvement situation occurred at the end of each search generation (i.e., one *while* iteration in Algorithm 2). The weight matrix (i.e., decision matrix) W is used to dynamically influence the probability of applying each action under each condition.

Given the weight matrix W (see Figure 2 for an example), we use the following equation (Guo, Goncalves, & Hsu, 2013) to calculate the probability $P(C_i, A_j)$ of applying action $A_j \in A$ under condition $C_i \in C$.

$$P(C_i, A_j) = \frac{W_{ij}}{\sum_{j \in A} W_{ij}} \tag{2}$$

At the beginning of each generation, the improvement situation is associated with a default condition. Then, according to the weight matrix, the appropriate action for this condition is selected according to the probability given by Eq. (2). At the end of each generation, the performed action is evaluated with respect to its condition and the concerned weight value is increased if an improvement in solution quality is obtained in this generation. We use a credit assignment to perform reinforcement learning in order to identify the beneficial experiences and determine a reward for them. Here, an experience is represented as a triplet (*condition* C_i, *action* A_j, *improvement* V). When a new best local or global solution is found, the weight value W_{ij} which is related to the action of this generation is reinforced by adding a reward rate σ to W_{ij}. Before adding the reinforcement value, all the weight values W_{ij} in the decision matrix is decreased with an evaporation value μ, in order to enlarge the influence of the new reward obtained in the current generation. The reinforcement with reward σ is then performed using the following equations (Guo, Goncalves, & Hsu, 2013):

$$W'_{ij} = \mu \times W''_{ij} \tag{3}$$

$$W_{ij} = \mu \times W'_{ij} + \sigma \tag{4}$$

where W'_{ij} is the weight value before adding the reinforcement σ, W''_{ij} is the weight value before the evaporation μ, and σ is the learning factor.

Figure 3 shows an illustrative example of this reinforcement learning process (More information about the example are given in Section 3.4.1). In the proposed MAOM-QAP approach, such a matrix is used by the decision maker agent (Section 3.4) and the local search agents (Section 3.5). The respective conditions and actions used by these agents are provided in these sections.

3.3. MAOM-QAP procedure

The general MAOM-QAP procedure is summarized in Algorithm 1. The decision maker agent initiates the search with a random feasible solution. Based on the weight matrix explained in the previous subsection, MAOM-QAP decides to trigger either the local search agents or the crossover agents. As a result, one of the two following cases occurs.

Algorithm 1 MAOM-QAP General Procedure.

```
Require: Four types of agents: one Decision Maker agent, two Local Search agents,
    one Perturbation agent, two Crossover agents
Ensure: Best information (i.e., solution) found which is a vector $V_{best}$ of location-
    facility assignments
 1: The decision maker agent is active until it needs exchanging with local search
    agents or crossover agents (Algorithm 2)
 2: if The decision maker agent decides to trigger local search agents then
 3:   The local search agents are activated and the decision maker agent waits for
      new information from them (Algorithm 3)
 4:   if A local search agent requests help from the perturbation agent then
 5:     The perturbation agent is activated and the local search agent is blocked until
        it receives information from the perturbation agent (Section 3.6)
 6:     The perturbation agent is killed after sending information to the correspond-
        ing local search agent
 7:   end if
 8:   if A local search agent requests help from the other local search agent then
 9:     The requesting local search agent is blocked until it receives information from
        the other local search agent (Algorithm 3)
10:   end if
11:   The local search agents are killed after sending information to the decision maker
      agent
12: end if
13: if The decision maker agent decides to trigger crossover agents then
14:   The crossover agents are activated and decision maker agent waits for new in-
      formation from the crossover agents (Section 3.7)
15:   The crossover agents are killed after sending information to the decision maker
      agent
16: end if
17: Return best information found $V_{best}$
```

Case 1: Local search agents are triggered. This case corresponds to the situation where the decision maker agent decides that a more intensified search is needed considering the current search state. For this, it triggers the local search agents by sending them the current solution (lines 10-11 of Algorithm 2). Each local search agent looks for the best solution in the predefined neighborhood for q iterations by applying a tabu search procedure. Each agent uses a different neighborhood structure and starts with the solution received from the decision maker agent (lines 6-11 of Algorithm 3).

After an iteration (i.e., one *while* iteration in Algorithm 3) of the local search agent concerned, each local search agent decides whether it needs to communicate with the other local search agent or it can continue its process without information exchange. This decision depends on another weight matrix Q (see Section 3.5). Q has the same mechanism as the weight matrix of the decision maker agent, but here, the actions to be performed are either to trigger another local search agent (for intensification) or a perturbation agent (for diversification). When the search needs to be intensified, the agent concerned calls another intensification-oriented local search agent and remains blocked until it receives the best solution sent by the other local search agent. Thereafter, the requesting agent completes its search starting with the solution received. When the desired action is about diversification, the perturbation agent is triggered (lines 15-25 of Algorithm 3). This agent performs one of two types of perturbations (reduced and strong perturbations) with the purpose of helping the local search agents to move towards new search areas. According to whether the requesting local search agent needs a small or large diversification, either a reduced perturbation or strong perturbation is performed. The new solution from the perturbation is passed to the local search agent to continue its search. The local search agents executes for a number of iterations by exchanging information as we just explained (lines 27-28 of Algorithm 3). The best solution obtained is forwarded to the decision maker agent (line 42 of Algorithm 3). The decision maker agent records the solution received in an archive that represents a common memory shared by all agents in the algorithms (lines 17-30 of Algorithm 2).
Case 2: Crossover agents are triggered. If the crossover agents are activated (lines 13-14 of Algorithm 2), two crossover agents are applied to two parent solutions (which are randomly selected from the archive) to create two offspring solutions (Section 3.7). Thes offspring solutions are sent to the decision maker agent which stores them in the archive if they are of good quality.

The decision maker agent uses the current best solution found so far to start the next cycle (generation) of the algorithm. The values of the weight matrix are updated according to the new state reached during the last generation, in order for the decision maker agent to activate the appropriate agents for the next generation. This search process is repeated until a stop condition is satisfied (e.g., a maximum number of generations) and the best solution discovered is retained as the final result.

3.4. Decision maker agent

The decision maker agent is the coordinating agent. According to the decision making matrix W (Section 3.2), the decision maker agent decides whether it triggers the local search agents (for more intensification) or crossover agents (for more diversification). It records the high-quality solutions which are discovered during the search in the shared memory (archive) (Algorithm 2).

The decision maker agent thus exchanges information with the local search agents and the crossover agents. The decision maker agent stays alive until reaching a stop criterion (a cutoff time limit, a allowed number of generations). During its life, it is blocked when other agents are activated. So, it has only one life cycle.

Algorithm 2 Decision maker agent behavior

```
Require: $n$ by $n$ distance matrix $d$ and flow matrix $f$
Ensure: A vector $V_{best}$ of facility locations
 1: $V \leftarrow Random\_permutation(n)$  {Random initial facility location assignment}
 2: $V_{best} \leftarrow V$  {$V_{best}$ records the best solution found so far}
 3: $C_{best} \leftarrow F$  {$F_{best}$ records the best objective value reached so far}
 4: $opt \leftarrow 0$  {$opt$ is the counter for consecutive non-improving local optimum}
 5: $W \leftarrow 0$  {Initialization of the weight matrix of the decision maker agent}
 6: $pop \leftarrow \emptyset$  {$pop$ is the archive of elite solutions found during the search}
 7: while Stopping condition not reached do
 8:   Update $W$  {Sections 3.2 and 3.4.1}
 9:   $Action\_type \leftarrow$ Select an action (agents) to activate based on $W$ {Section 3.4.1}
10:   if $Action\_type$ = Local search agents then
11:     Activate the local search agents and send $V$ to the local search agents
12:   else
13:     Activate the crossover agents and send $V$ to the crossover agents
14:     $opt \leftarrow 0$
15:   end if
16:   $V_1 \leftarrow \emptyset$, $V_2 \leftarrow \emptyset$  {$V_1$ and $V_2$ are two solutions received from the activated
      agents, initialized to empty}
17:   if $V_1 \neq \emptyset$ AND $V_2 \neq \emptyset$ then
18:     if $F(V_1) \geq F(V_2)$ then
19:       $V \leftarrow V_1$
20:     else
21:       $V \leftarrow V_2$
22:     end if
23:     $tr \leftarrow Exist(V_1, V_2, pop)$  {Check if $V_1$ and/or $V_2$ are in the archive $pop$}
24:     if $tr = false$ then
25:       Add $V_1$ and/or $V_2$ to $pop$  {Add both solutions or one of them in $pop$}
26:     end if
27:     Let $V'$ be the best solution between $V_1$ and $V_2$
28:     if $F(V') \leq F_{best}$ then
29:       $V_{best} \leftarrow V'$, $F_{best} \leftarrow F(V')$
30:     else
31:       $opt$=$opt$+1
32:     end if
33:   else
34:     Block this agent  {Decision maker agent waits for solutions from other agents}
35:   end if
36: end while
37: Return $F_{best}$ and $V_{best}$
```

3.4.1. Conditions and actions

During the search process of our algorithm, three types of solutions are used to define conditions for agent activation: the *local current solution* obtained by each agent, the *local best solution* obtained by each agent and the *global best solution* obtained among all agents in the process. The weight matrix of the decision maker agent is composed of four different conditions which cover significant situations that may occur during the search process:

- C_1 = The algorithm does not reach g_0 generations (cycles);
- C_2 = The local or global best solution is improved in the recent g_1 generations and this improvement is a small improvement in the objective function value F;
- C_3 = The local or global best solution is improved in the recent g_1 generations and this improvement is a large improvement in the objective function value F;
- C_4 = The global best solution has not been improved in the recent g_2 generations. This solution is a deep local optimum or an optimum solution.

where g_0, g_1 and g_2 are parameters set by the user according to the total allowed generation number or total run time.

The set of actions are:

- A_1 = Activating the local search agents;
- A_2 = Activating the crossover agents;

At the beginning of the search or when there is a large improvement obtained by the application of an action between two successive generations (this corresponds to the situations of C_1 and C_3), the search progresses well and in this case, it is appropriate to make intensified search by triggering the local search agents. If the decision maker agent observes no improvement or an insignificant improvement (this corresponds to the situations of C_2 and C_4), the search is stagnating and needs to be diversified by activating the crossover agents.

After each generation (i.e., when the activated agents return their found solution), the decision maker agent updates its weight matrix as explained in Section 3.2. Figure 3 illustrates how the weight matrix is changed by the reinforcement learning procedure. We suppose that in iteration i of Algorithm 2, the current condition C_3 is verified (i.e., the local best solution is greatly improved in the recent $g_1 = 20$ generations). Under this condition, action A_1 which corresponds to activating local search agents is applied for the current generation. Reinforcement learning is applied by modifying the weight W_{31} to augment the chance of selecting the corresponding action under this condition. In Figure 3, we show for this example, the initial weight matrix (left), and the matrix (right) after the update with a reward value $\sigma = 1$ and an evaporation value $\mu = 0.5$.

3.4.2. Archive of elite solutions

The decision maker agent records the best solutions discovered during the search in an archive. These solutions are generated and submitted by the local search agents and crossover agents. Even if the archive is shared by all the agents of the model, only the decision maker agent is responsible to update the archive. Each time the decision maker agent receives a new solution, it adds the solution in the archive if the solution is of good quality and is not present already in the archive.

3.5. Local search agents

The local search agents are designed for intensification. During its life time, a local search agent employs a neighborhood to search for improved solutions. During the search, each local search agent can decide, with the help of its weight matrix, to exchange information with another alive local search agent or with the perturbation agent depending on its state of search. At the end of each local search agent run, the best solution found by the agent is sent to the decision maker agent. For the QAP, we use two local search agents which are both based on tabu search (Glover & Laguna, 1997) and apply two different strategies to explore the swap-based neighborhood (See Algorithm 3). Below, we define the used neighborhood, present the two neighborhood exploration strategies and explain the conditions and actions employed by the local search agents.

Algorithm 3 Local search agent behavior

```
Require: Solution $V_0$ received from the decision maker agent, parameters: maximum allowed iterations $iteration\_max$, improvement threshold $interval$, consecutive non-improving iterations $max\_opt\_LS$
Ensure: A vector $V_{best\_LS}$ of facility locations
1: $V \leftarrow V_0$ {$V$ is the current solution found by each local search agent}
2: $Q \leftarrow 0$ {$Q$ is the weight (decision) matrix of local search agents, Section 3.5.4}
3: $Tabu\_List \leftarrow \emptyset$ {$Tabu\_List$ is the tabu list, Section 3.5.3}
4: $opt = 0$ {$opt$ is the counter for consecutive non-improving local optima for each local search agent}
5: Compute $\delta$ {$\delta$ is the move gain matrix, Section 3.5.1}
6: $V_1 \leftarrow V_0$ {$V_1$ records the solution obtained in generation $iteration$-1}
7: while $iteration \leq iteration\_max$ do
8:   $V \leftarrow$ Generate the best neighboring solution based on $\delta$ {Sections 3.5.1 & 3.5.3}
9:   Update $\delta$ and $Tabu\_List$
10:  if $F(V) \leq F(V_{best\_LS})$ then
11:    $V_{best\_LS} \leftarrow V$
12:  else
13:    $opt = opt + 1$
14:  end if
15:  if $(F(V) - F(V_1)) < interval$ or $opt = max\_opt\_LS$ then
16:    {The local search agent is stagnating and needs helps from another local search agent or the perturbation agent}
17:    $V_{perturbed} \leftarrow \emptyset$ {$V_{perturbed}$ is the solution received from another agent (local search agent or perturbation agent), initialized to empty}
18:    Update $Q$ {Update the weight matrix based on the improvement of the current solution, Sections 3.5.4 & 3.2}
19:    $Action\_exchange \leftarrow$ Select the action (agent) to activate based on $Q$
20:    if $Action\_exchange$ = Triggering perturbation agent with weak behavior then
21:      Activate the perturbation agent with reduced behavior and send it solution $V$
22:    end if
23:    if $Action\_exchange$ = Triggering the perturbation agent with strong behavior then
24:      Activate the perturbation agent with strong behavior
25:      $opt \leftarrow 0$
26:    end if
27:    if $Action\_exchange$ = Triggering other local search agent then
28:      Request the current solution of other local search agent
29:    end if
30:    Let $V_{perturbed}$ be the new solution received from any of the above exchange
31:    if $V_{perturbed} \neq \emptyset$ then
32:      $V \leftarrow V_{perturbed}$
33:      Update $\delta$
34:    else
35:      Block this agent {This agent waits for a solution from other agents }
36:    end if
37:  else
38:    $V_1 \leftarrow V$ {Local search agent continues its exploration without exchanging information with other agents}
39:  end if
40:  $iteration = iteration + 1$
41: end while
42: Return $V_{best\_LS}$ to decision maker agent
```

3.5.1. Neighborhood

As explained in the introduction, a candidate QAP solution can be conveniently represented by a permutation π of $\{1, 2, ...n\}$ where π_i is the facility assigned to location i. Let $swap(i, j)$ be a move operator which exchanges the facilities located at i and j. Given a candidate solution π, let $\pi' = \pi \oplus swap(i, j)$ be the neighboring solution of π obtained by exchanging the facilities π_i and π_j of locations i and j. Then $N(\pi) = \{\pi' : \pi' = \pi \oplus swap(i, j), i, j \in \{1, 2, ...n\}, i \neq j\}$ is the set of neighboring solutions induced by the swap operator. To assess the relative quality of a neighboring solution π', i.e., the cost variation $\delta(\pi, i, j) = F(\pi') - F(\pi)$ between π and π' (also called the move gain of $swap(i, j)$), we use the incremental technique proposed in (Taillard, 1991) which can be achieved in $O(n)$ in the worst case.

3.5.2. Neighborhood exploration strategies

Given this neighborhood, our local search agents employ two different strategies to explore the neighboring solutions. Let π be the incumbent solution and $N(\pi)$ its neighborhood. Our first local search agent examines the whole neighborhood $N(\pi)$ (in $O(n^3)$) and retains the best neighboring solution which becomes the new incumbent solution. As such, this local search agent realizes a highly aggressive exploitation of the neighborhood, ensuring thus an intensified search. Our second local search agent operates slightly differently in two stages. First, picks at random a location i. Then it seeks the best location j which leads to the highest $swap(i, j)$ move gain. This neighborhood exploration strategy, which is achieved in $O(n^2)$, leads to a less aggressive search. Yet, given the random choice of one of the two locations to be exchanged, this strategy provides the local search agent with an intensified search while ensuring some degree of diversification at the same time.

3.5.3. Tabu list

Each local search agent uses a traditional tabu list to prevent the search from revisiting a previously encountered solution. Each time a facility x_i is displaced from location i to a new location by a $swap(i, j)$ move, x_i is forbidden to move back to location i during the next h iterations. The iterations h is dynamically determined by $h = \alpha \times F(S) + rand(10)$ where $rand(10)$ takes a random number in $\{1, ..., 10\}$ and α is set to 0.09.

3.5.4. Conditions and actions

As explained at the beginning of this section, each running local search agent can decide, with the help of a weight matrix, to exchange information with another alive local search agent or with the perturbation agent depending on its state of search. In this section, we present the conditions and actions employed by the local search agents. The underlying weight matrices are managed by the same technique of Section 3.2.

The set of the conditions are:

- C_1 = The local best solution is improved in recent q_3 generations and this improvement is a small improvement;
- C_2 = The local best solution is not improved in recent q_4 generations;
- C_3 = The local best solution is not improved in recent q_5 generations and $q_5 > q_4$.

where q_3, q_4 and q_5 are parameters set by the user according to the total generation number or total run time.

The set of actions are:

- A_1 = Activating other local search agent;
- A_2 = Activating the reduced perturbation behavior in the perturbation agent;
- A_3 = Activating the strong perturbation behavior in the perturbation agent.

Each condition promotes a certain action. Thus, when the condition C_1 is met, one pursues an intensified search by activating other local search (A_1). When C_2 (resp. C_3) is satisfied, the search needs to be diversified by triggering the perturbation agent with reduced (resp. strong) behavior (A_2 or A_3). The choice of the most suitable action is controlled by the corresponding weight matrix of each local search agent.

3.6. Perturbation agent

The perturbation agent is triggered by an alive local search agent under specific conditions (C_2 and C_3 of Section 3.5.4). Basically, this agent disrupts a solution received from the calling local search agent. The disruption is achieved by either a reduced perturbation behavior or a strong perturbation behavior. The resulting solution is then sent back to the local search agent which uses the perturbed solution as its new current solution. Since the perturbation agent can be called many times, it can have several life cycles.

3.6.1. Reduced perturbation behavior

The perturbation agent can be triggered when a local search agent observes a slight search stagnation (condition C_2 of Section 3.5.4). From the solution received from the local search agent, the perturbation agent performs a number of random swap moves to generate a new solution. This is achieved by exchanging the locations of two facilities chosen randomly. Also, the number of perturbation swap moves is chosen randomly between 1 and $n/3$ (n being the number of facilities).

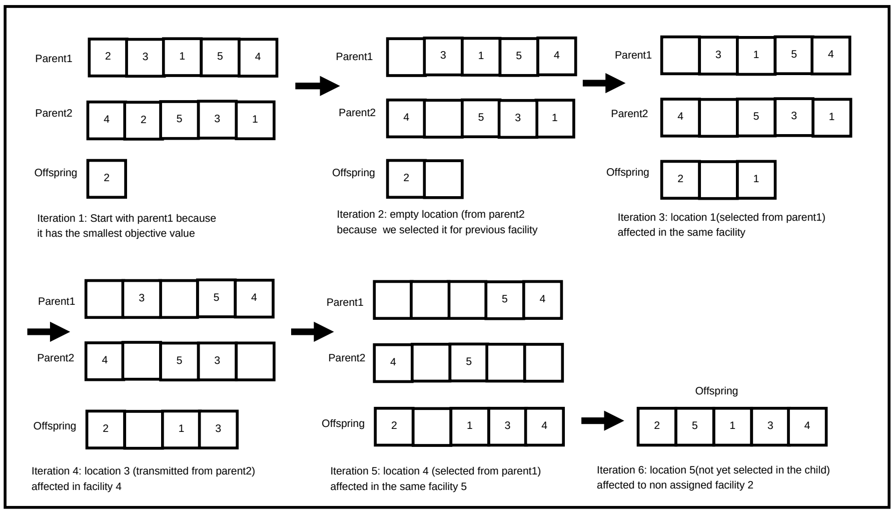
Parent1
Parent2
Offspring
Iteration 1: Start with parent1 because it has the smallest objective value
Iteration 2: empty location (from parent2 because we selected it for previous facility
Iteration 3: location 1(selected from parent1) affected in the same facility
Iteration 4: location 3 (transmitted from parent2) affected in facility 4
Iteration 5: location 4 (selected from parent1) affected in the same facility 5
Iteration 6: location 5(not yet selected in the child) affected to non assigned facility 2

Figure 4: An example for the first crossover operator used by the crossover agent

3.6.2. Strong perturbation behavior

The second case where the perturbation agent can be activated is when it receives a request for strong perturbation from a local search agent. The perturbation agent then employs the common archive of elite solutions to create a new solution. From this archive, the perturbation agent extracts the number of occurrence of each facility i assigned to location x_i. Then, each facility i is assigned to the location having the smallest occurrence number. Additional data structures are employed to avoid the creation of the same solution for future calls to the perturbation agent.

3.7. Crossover agents

These are agents for diversification. Each crossover agent performs a different crossover operation to generate one offspring solution. Offspring solutions are transmitted to the decision maker agent to serve as a new starting point for the search process. For the QAP, we employ two crossover agents. In both cases, two parents are selected randomly from the common archive. Each crossover agent applies one of the following crossover operators.

- The first crossover operator consists in blending uniformly information from the parents. Given two selected parents, the crossover operator builds one offspring solution by alternatively transmitting location-facility assignments from the parent. Specifically, starting with the parent having the smallest objective value, we transmit the facility of the first location (i.e., with index one) to the first location of the child and then remove the assigned facility from both parents. For the second location of the child, we switch to the other parent and transmit the facility (which may

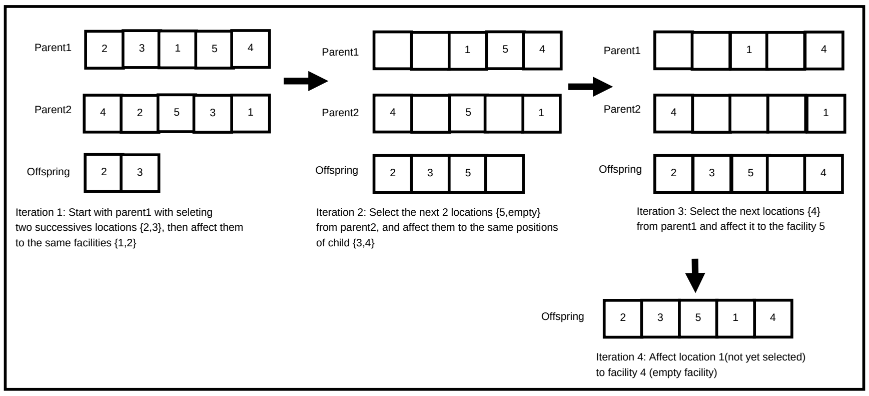
Parent1
2
3
1
5
4
Parent2
4
2
5
3
1
Offspring
2
3
Iteration 1: Start with parent1 with seleting two successives locations {2,3}, then affect them to the same facilities {1,2}
Parent1
1
5
4
Parent2
4
5
1
Offspring
2
3
5
Iteration 2: Select the next 2 locations {5,empty} from parent2, and affect them to the same positions of child {3,4}
Parent1
1
4
Parent2
4
1
Offspring
2
3
5
4
Iteration 3: Select the next locations {4} from parent1 and affect it to the facility 5
Offspring
2
3
5
1
4
Iteration 4: Affect location 1(not yet selected) to facility 4 (empty facility)

Figure 5: An example for the second crossover operator used by the crossover agent (z=2 in this example)

be empty) of the second location (i.e., with index two) to the child. Then we go back to parent one and repeat this process until reaching the last location. Finally, the unassigned facilities of the offspring are affected to a location randomly chosen among the set of the free locations. This operator can be realized in $O(n)$. See Figure 4 for an illustrating example.

- The second crossover operator has the same idea of the first crossover operator, only the first $z << n$ (a parameter) location-facility assignments of each parent are transmitted to the offspring solution. The crossover agent starts from the parent who has the smallest objection value to build the child. It copies the z first location-facility assignments of this parent into the child. Then, it extracts from the other parent, the next z location-facility assignments and copies them to the child from the $z + 1$ locations. Finally, each unassigned facility is affected to a random unassigned location. This operator can be realized in $O(n)$. See Figure 5 for an illustrating example.

4. Experimentation

4.1. Experimental results

This section presents experimental results of the MAOM-QAP algorithm which is implemented in Java using the multi-agent platform Jade. The program is run on a computer with a Core I5 2.5 GHz, 8GB of RAM. To assess the MAOM-QAP algorithm, tests were realized on various benchmark instances from the QAPLIB (http://www.seas.upenn.edu/qaplib/inst.html). The instance size n varies from 12 to 150 (indicated in the instance name). The QAPLIB archive contains 135 instances divided into four types:

1. Type I: Real-life instances obtained from practical applications;

2. Type II: Unstructured and random instances for which the distance and flow matrices are randomly generated based on a uniform distribution;
3. Type III: Randomly generated instances with structure that is similar to that of real-life instances;
4. Type IV: Instances in which distances are based on the Manhattan distance on a grid.

Following Benlic & Hao (2015), we focus on the set of 21 most challenging instances: 5 instances of type II, 5 instances of type III and 11 instances of type IV. Since the remaining 114 instances, including all the real-life instance of Type I, are easy for MAOM-QAP and other current QAP algorithms, they are not included in the paper.

We adjusted the parameters of the proposed algorithms by an experimental study. They depend on the type of the problem. The number of iterations for each local search agent (*iter_max*) is fixed to 1000. The parameter *interval* that evaluates the improvement of the solution is fixed to 10000 for the decision maker agent and the local search agents. The parameters g_0, g_1, g_2, q_3, q_4 and q_5, which are the numbers of generations responsible for controlling the improvement of the search process(presented in Section 3.4.1 and Section 3.5.4), are fixed respectively to 2, 10, 2, 20, 20 and 25. The parameter rate μ used in updating the weight matrices is fixed to 0.5. The stopping condition is the elapsed time which we set to 12 hours for all the instance of size $n < 100$, and to 24 hours for the large instances of size $n >= 100$. The best-known solutions can be attained before these time limits.

We compare MAOM-QAP to seven best-known algorithms of the literature reviewed in Section 2.

- Improved hybrid genetic algorithm (IHGA) (Misevicius, 2004);
- Iterated tabu search (ITS) (Misevicius, Lenkevicius, & Rubliauskas, 2006);
- Population-based iterated local search (PILS) (Stützle, 2006);
- A hybrid genetic tabu search algorithm (MRT60) (Drezner, 2008);
- Cooperative parallel tabu search (CPTS) (James, Rego, & Glover, 2009);
- The Breakout local search (BLS) (Benlic & Hao, 2013);
- The population-based Memetic Algorithm (BMA) (Benlic & Hao, 2015).

The main purpose of this assessment is to compare our results with the *best-known results* even reported by any existing algorithms of the literature. Note that these best best-known results, as well as those of the reference algorithms, have been achieved by different algorithms under various conditions (different stop conditions, computing platforms etc). As a result, the comparisons with the existing methods are included only for indicative information.

Table 1: Results of MAOM-QAP compared to five best performing QAP approaches on unstructured instances (type II) and on Real-life like instances (type III). The times are given in minutes.

Problem	BKS	MAOM-QAP		BMA		BLS		CPTS		ITS		IHGA	
		% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)
Random instances (Type II)													
tai40a	3139370	0.099(2)	83.5	0.059(2)	8.1	**0.022**(7)	38.9	0.148(1)	3.5	0.210(1)	0.8	0.209(1)	1.4
tai50a	4938796	0.320 (1)	135.2	**0.131**(2)	42.0	0.157(2)	45.1	0.440(0)	10.3	0.373(0)	3.0	0.262(0)	5.0
tai60a	7205962	0.385 (2)	178.1	**0.144**(2)	67.5	0.251(1)	47.9	0.476(0)	26.4	0.330(1)	9.7	0.583(0)	12
tai80a	13499184	**0.426**(0)	225	**0.426**(0)	65.8	0.517(0)	47.3	0.691(0)	94.8	0.494(0)	25.0	0.756(0)	53.3
tai100a	21052466	0.470 (0)	288	**0.405**(0)	44.1	0.430 (0)	39.0	0.589(0)	261.2	0.427(0)	60.0	0.606(0)	200.0
Average		0.341	107.6	0.233	45.5	0.275	43.6	0.469	79.2	0.367	19.2	0.483	54.3
Real-life like instances (Type III)													
tai50b	458821517	**0.000**(10)	14.3	**0.000**(10)	1.2	**0.000**(10)	2.8	**0.000**(10)	13.8	**0.000**(10)	0.9	**0.000**(10)	0.3
tai60b	608215054	**0.000**(10)	38.2	**0.000**(10)	5.2	**0.000**(10)	5.6	**0.000**(10)	30.4	**0.000**(10)	2.2	**0.000**(10)	0.7
tai80b	818415043	**0.000**(10)	62.7	**0.000**(10)	31.3	**0.000**(10)	11.4	**0.000**(10)	110.9	**0.000**(10)	5.8	**0.000**(10)	2.5
tai100b	1185996137	**0.000**(10)	91.2	**0.000**(10)	13.6	**0.000**(10)	16.0	0.001(8)	241.0	**0.000**(9)	23.3	**0.000**(10)	7.3
tai150b	498896643	0.077(0)	9982	**0.060**(1)	78.1	0.100(0)	80.5	0.076(0)	7377.8	0.100(1)	60.0	0.111(2)	38.3
Average		0.015	1030.8	0.012	25.9	0.020	23.3	0.015	1554.8	0.020	18.4	0.022	9.8

Table 2: Comparative results between MAOM-QAP and four best performing QAP approaches on grid-based (type IV) instances. The times are given in minutes

Problem	BKS	MAOM-QAP		BMA		BLS		CPTS		MRT60	
		% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)
sko72	66256	**0.000**(10)	63.3	**0.000**(10)	3.5	**0.000**(10)	4.1	**0.000**(10)	69.6	**0.000**(10)	19.9
sko81	90998	**0.000**(10)	208.5	**0.000**(10)	4.3	**0.000**(10)	13.9	**0.000**(10)	121.4	**0.000**(10)	31.9
sko90	115534	**0.000**(10)	256.4	**0.000**(10)	15.3	**0.000**(10)	16.6	**0.000**(10)	193.7	**0.000**(10)	48.5
sko100a	152002	**0.000**(10)	321	**0.000**(10)	22.3	0.001(9)	20.8	**0.000**(10)	304.8	**0.000**(10)	73.6
sko100b	153890	**0.000**(10)	322.2	**0.000**(10)	6.5	**0.000**(10)	10.8	**0.000**(10)	309.6	**0.000**(10)	73.6
sko100c	147862	**0.000**(10)	324.8	**0.000**(10)	12.0	**0.000**(10)	15.5	**0.000**(10)	316.1	**0.000**(10)	73.6
sko100d	149576	**0.000** (9)	330	0.006(9)	20.9	0.001(5)	38.9	**0.000**(10)	309.8	**0.000**(10)	73.6
sko100e	149150	**0.000**(10)	343.3	**0.000**(10)	11.9	**0.000**(10)	42.5	**0.000**(10)	309.1	**0.000**(10)	73.6
sko100f	149036	**0.000**(10)	320	**0.000**(10)	23.0	**0.000**(10)	17.3	0.003(4)	310.3	0.000(9)	43.5
wil100	273038	**0.000**(10)	355	**0.000**(10)	14.5	**0.000**(10)	18.9	**0.000**(10)	316.6	**0.000**(10)	73.6
tho150	8133398	0.011 (0)	523	0.008(3)	416.4	0.023(1)	268.8	0.013(0)	1991.7	**0.003**(3)	1223.6
Average		0.001	229	0.001	50.1	0.002	42.6	0.001	413.9	0.000	164.5

Table 1 reports our computational results along with those of five reference algorithms on the unstructured instances (type II) and real-life like instances (type III). The second column 'BKS' presents for each instance the best-known objective value ever reported in the literature. For each algorithm, column $\bar{\delta}$ shows the percentage deviation of the average solution, obtained with the considered algorithm over a certain number of trials, from the best-known solution. If known, the success rate for reaching the best-known solution over several trials is given in parentheses next to the value of the $\bar{\delta}$. The CPU time (in minutes) is only given for indicative purposes. The last row indicates the averaged information. Table 1 discloses that for the unstructured instances (type II), MAOM-QAP finds the best-known solution for 7 out of the 9 instances (including four omitted easy instances) like other algorithms. The average deviation $\bar{\delta}$ from the best-known solution is 0.341 over the 5 instances, which is better than CPTS, ITS and IHGA, but worse than the most recent BMA and BLS. As to the real-life like instances (type III), MAOM-QAP can attain the best-known solution for all the instances, except for the largest tai150b. The deviation $\bar{\delta}$ from the best-known solution is only 0.015 over the 5 instances, which matches the performance of BLS and CPTS.

Table 2 presents our computational results along with those of four reference algorithms on the instances with grid distances (type IV). We observe that MAOM-QAP is able to reach the best-known results for 14 out of the 15 instances (including four omitted easy instances) with a deviation $\bar{\delta}$ of 0.001 over the 5 instances, which is among the three best results with BMA and CPTS. As to the computing times, MAOM-QAP is more computationally expensive. This issue is further discussed in the last section.

4.2. Impact of perturbation agent on MAOM-QAP

The MAOM-QAP algorithm integrates a perturbation agent which is called by the local search agents for the purpose of diversification. In this section, we perform an experiment to assess the usefulness of the perturbation agent. For this, we compare MAOM-QAP and its variant with the perturbation agent disabled (the variant is denoted by MAOM-QAP'). We run both of them under the same condition as specified in Section 4 and report the comparative results in Tables 3 and 4. These tables disclose that on all the benchmarks, MAOM-QAP without the perturbation agent fails to reach the best-known result of any of the 21 instances. These results show that the perturbation agent reinforces the search performance of MAOM-QAP.

4.3. Impact of crossover agents on MAOM-QAP

In order to show the relative effectiveness of the crossover agents in our algorithm, we compare MAOM-QAP with and without the crossover agents. As before, we run both algorithms under the same condition as specified in Section 4 and report the comparative results in Tables 5 and 6 where MAOM-QAP" is MAOM-QAP without the crossover agents. We observe that the algorithm without the crossover agents (MAOA-QAP") performs much worse since it can

Table 3: Comparison of the MAOM-QAP algorithm with its variant without the perturbation agent (MAOM-QAP') on unstructured instances (type II) and Real-life like instances (type III)

Problem	BKS	MAOM-QAP		MAOM-QAP'	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
tai40a	3139370	**0.099(2)**	83.5	4.556(0)	0.00
tai50a	4938796	**0.320(1)**	135.2	7.64(0)	20.3
tai60a	7205962	**0.385(2)**	178.1	4.71(0)	22.1
tai80a	13499184	**0.426(0)**	225	3.898(0)	30.5
tai100a	21052466	**0.470(0)**	288	4.74(0)	28.45
Average		**0.341**	107.6	5.419	11.26
tai50b	458821517	**0.000(10)**	14.3	13.587(0)	0.00
tai60b	608215054	**0.000(10)**	38.2	8.758(0)	0.00
tai80b	818415043	**0.000(10)**	62.7	11.823(0)	0.00
tai100b	1185996137	**0.000(10)**	91.2	14.182(0)	9.16
tai150b	498896643	**0.077(0)**	9982	13.542(0)	83.1
Average		**0.015**	1030.8	11.447	9.22

Table 4: Comparison of the MAOM-QAP algorithm with its variant without the perturbation agent (MAOM-QAP') on grid-based (type IV) instances

Problem	BKS	MAOA-QAP		MAOA-QAP'	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
sko72	66256	**0.000(10)**	63.3	3.156(0)	0.00
sko81	90998	**0.000(10)**	208.5	4.581(0)	12
sko90	115534	**0.000(10)**	256.4	5.789(0)	11.2
sko100a	152002	**0.000(10)**	321	4.865(0)	15
sko100b	153890	**0.000(10)**	322.2	5.889(0)	13.8
sko100c	147862	**0.000(10)**	324.8	4.19(0)	14.1
sko100d	149576	**0.000(10)**	330	3.86(0)	30
sko100e	149150	**0.000(10)**	343.3	4.245(0)	25.4
sko100f	149036	**0.000(10)**	320	3.79(0)	15.4
wil100	273038	**0.000(10)**	355	5.228(0)	18.1
tho150	8133398	**0.011(0)**	523	3.699(0)	45
Average		**0.001**	229	4.143	13.33

Table 5: Comparison of the MAOM-QAP algorithm with its variant without the crossover agents (MAOM-QAP") on grid-based (type IV) instances

Problem	BKS	MAOM-QAP		MAOM-QAP"	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
sko72	66256	**0.000(10)**	63.3	1.99(0)	4.5
sko81	90998	**0.000(10)**	208.5	2.457(0)	12.4
sko90	115534	**0.000(10)**	256.4	2.75(0)	15.7
sko100a	152002	**0.000(10)**	321	2.486(0)	12.3
sko100b	153890	**0.000(10)**	322.2	1.25(0)	11.8
sko100c	147862	**0.000(10)**	324.8	3.724(0)	13.78
sko100d	149576	**0.000(10)**	330	2.785(0)	25.89
sko100e	149150	**0.000(10)**	343.3	1.42(0)	15.9
sko100f	149036	**0.000(10)**	320	3.75(0)	22.4
wil100	273038	**0.000(10)**	355	2.15(0)	28.2
tho150	8133398	**0.011(0)**	523	3.489(0)	34
Average		**0.001**	229	2.56	31.5

find the best-known results for only 4 out of the 21 instances. Thus, we conclude that the crossover agents are indispensable for the performance of our MAOM-QAP algorithm.

5. Conclusion and perspectives

In this paper, we introduced the distributed multi-agent optimization model (MAOM) applied to the Quadratic Assignment Problem. The decision maker agent is the central agent which decides the most suitable agent to activate and maintains a shared memory to record the elite solutions discovered during the search. Its decisions are influenced by a learning-based probabilistic strategy which dynamically adjusts the application probability of a particular action under a specific condition. On the other hand, the local search agents are introduced to ensure an intensified examination of specific search zones while the perturbation agents and crossover agents are used to diversify the search. The proposed method is thus characterized by its distributed architecture which supports a distributed implementation of the search algorithm, the interacting agents which ensures the role of intensified search and diversified search, and the adaptive decision making with reinforcement learning which allows the search algorithm to dynamically adjust its exploration strategy of the search space.

The computational study shows that the proposed approach performs well on the tested benchmark instances in terms of solution quality. On the other hand, this approach could be further improved by following several directions. First, the current version of the proposed model is only a proof-of-concept implementation. As such, its computational efficiency is not satisfactory compared with the current state of the art QAP methods. For a large part, this is due to the multi-agent platform Jade which is used to implement our proposed model.

Table 6: Comparison of the MAOM-QAP algorithm with its variant without the crossover agents (MAOM-QAP") on unstructured instances (type II) and Real-life like instances (type III)

Problem	BKS	MAOM-QAP		MAOM-QAP"	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
tai40a	3139370	**0.099(2)**	83.5	2.8(0)	2.08
tai50a	4938796	**0.320(1)**	135.2	3.78(0)	15.12
tai60a	7205962	**0.385(2)**	178.1	2.75(0)	18.4
tai80a	13499184	**0.426(0)**	225	3.82(0)	20.1
tai100a	21052466	**0.470(0)**	288	3.28(0)	20.5
Average		**0.341**	107.6	2.87	8.46
tai50b	458821517	**0.000(10)**	14.3	4.78(0)	0.00
tai60b	608215054	**0.000(10)**	38.2	5.96(0)	0.00
tai80b	818415043	**0.000(10)**	62.7	5.2(0)	5.2
tai100b	1185996137	**0.000(10)**	91.2	5.11(0)	8.4
tai150b	498896643	**0.077(0)**	9982	6.45(0)	23
Average		**0.015**	1030.8	5.29	3.66

One interesting perspective to improve the computational efficiency of the model is to envisage a dedicated implementation. For instance, the method could be implemented in a distributed manner across a number of machines where each machine is responsible for running one agent of the model. The configuration can even be dynamically changed at run-time by moving agents from one machine to another as and when required. Using such an implementation method, we can create other types of agents that can be qualified as mobile agents.

Second, reinforcement learning is another key ingredient of the proposed model and can also be ameliorated. Indeed, better learning will make the agents more informed and allow them to choose the most appropriate action to perform when the required condition is met. In the current work, reinforcement learning is realized with the help of a simple reward mechanism. More elaborated techniques are worth consideration. Moreover, forgetting mechanisms could also be integrated into the learning mechanism for a better balance of exploration and exploitation.

Third, the current work focuses on application of the proposed MAOM model to the QAP. Yet, we argue that the model is a general framework which could be adapted to solve other difficult problems. In fact, this can be achieved by keeping the MAOM architecture and specifying the required agents for intensification and diversification. In this regard, we are investigating this approach to solve combinatorial search problems like graph coloring (Sghir, Hao, Ben Jaafar, & Ghédira, 2015) and winner determination in combinatorial auctions.

Finally, this study demonstrates that multi-agnet systems constitute an interesting source of inspiration for designing distributed optimization algorithms. With the current trend of proliferation of cheap distributed computing facilities, such an approach would be of great interest for designing powerful intelligent systems for various search problems.

Acknowledgments

We are grateful to the anonymous referees for valuable suggestions and comments which helped us improve the paper. The corresponding author (J.K. Hao) is partially supported by the PGMO project (2013-2015, Jacques Hadamard Mathematical Foundation, France).

References

Baykasoğlu, A., & Kaplanoğlu, V. (2015). An application oriented multi-agent based approach to dynamic load/truck planning. *Expert Systems with Applications*, 42(15–16), 6008–6025.

Benlic, U., & Hao, J.K. (2013). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9), 4800–4815.

Benlic, U., & Hao, J.K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1): 584–595.

Burkard, R.E. (1991). Locations with spatial interactions: The quadratic assignment problem. In P. Mirchandani and L. Francis (Eds.), *Discrete Location Theory*, New York.

Couellan, N., Jan, S., Jorquera, T., & Georgé, J-P. (2015). Self adaptive support vector machine: a multi-agent optimization perspective. *Expert Systems with Applications*, 42(9), 4284–4298.

Drezner (2008). Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers and Operations Research*, 35(3), 717–736.

Duman, E., & Or, I. (2007). The quadratic assignment problem in the context of the printed circuit board assembly process. *Computers and Operations Research*, 34(1), 163–17.

Erdogan, G., & Tansel, B. (2007). A branch-and-cut algorithm for quadratic assignment problems based on linearizations. *Computers and Operations Research*, 34(4), 1085–1106.

Gonçalves, F.A.C.A., Guimarães, F.G., & Souza, M.J.F. (2014). Query join ordering optimization with evolutionary multi-agent systems. *Expert Systems with Applications*, 41(15), 6934–6944.

Glover, F., & Laguna, M. (1997). Tabu search. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Guo, Y., Goncalves, G., & Hsu, T. (2013). A multi-agent based self-adaptive genetic algorithm for the long-term car pooling problem. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 12(1), 45–66.

Hahn, P., Grant, T., & Hall, N. (1998). A branch-and-bound algorithm for the quadratic assignment problem based on the Hungarian method. *European Journal of Operational Research*, 108(3), 629–640.

James, T., Rego, C., & Glover, F. (2009). A Cooperative Parallel Tabu Search Algorithm for the Quadratic Assignment Problem. *European Journal of Operational Research*, 195(3), 810–826.

Martin, S., Ouelhadj, D., Smetb, P., Beullens, P., & Özcan, E. (2013). Cooperative search for fair nurse rosters. *Expert Systems with Applications*, 40(16), 6674–6683.

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4), 337–352.

Misevicius, A. (2004). An improved hybrid genetic algorithm: New results for the quadratic assignment problem. *Knowledge-Based Systems*, 17(2–4), 65–73.

Misevicius, A., Lenkevicius, A., & Rubliauskas, D. (2006). Iterated tabu search: An improvement to standard tabu search. *Information Technology and Control*, 35(3), 187–197.

Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. *Journal of the Association of Computing Machinery*, 23, 555–565.

Satunin, S., & Babkin E. (2014). A multi-agent approach to Intelligent Transportation Systems modeling with combinatorial auctions. *Expert Systems with Applications*, 41(15), 6622–6633.

Sghir, I., Hao, J.K., Ben Jaafar, I., & Ghédira, K. (2015). A distributed hybrid algorithm for the graph coloring problem. *12th Biennial International Conference on Artificial Evolution*, Lyon, France, October 2015, to appear in *Lecture Notes in Computer Science*, Springer.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3), 1519–1539.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17, 443–455.

Wang, S., & Wang, L. (2015). A knowledge-based multi-agent evolutionary algorithm for semiconductor final testing scheduling problem. *Knowledge-Based Systems*, 84, 1–9.

Weiss, G. (Ed.) (2013). Multiagent Systems. MIT Press, March 2013.

Zheng, X., & Wang, L. (2015). A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert Systems with Applications*, 42(15–16), 6039–6049.