



Improving ant colony optimization algorithm with epsilon greedy and Levy flight

Yahui Liu¹ · Buyang Cao² · Hehua Li³

Received: 19 January 2020 / Accepted: 18 March 2020 / Published online: 31 March 2020
© The Author(s) 2020

Abstract

Ant colony optimization (ACO) algorithm is a meta-heuristic and reinforcement learning algorithm, which has been widely applied to solve various optimization problems. The key to improving the performance of ACO is to effectively resolve the exploration/exploitation dilemma. Epsilon greedy is an important and widely applied policy-based exploration method in reinforcement learning and has also been employed to improve ACO algorithms as the pseudo-stochastic mechanism. Levy flight is based on Levy distribution and helps to balance searching space and speed for global optimization. Taking advantage of both epsilon greedy and Levy flight, a greedy–Levy ACO incorporating these two approaches is proposed to solve complicated combinatorial optimization problems. Specifically, it is implemented on the top of max–min ACO to solve the traveling salesman problem (TSP) problems. According to the computational experiments using standard TSPLIB instances, greedy–Levy ACO outperforms max–min ACO and other latest TSP solvers, which demonstrates the effectiveness of the proposed methodology.

Keywords Ant colony optimization · Epsilon greedy · Levy flight · Levy distribution

Introduction

Ant colony optimization (ACO) algorithm is a meta-heuristic algorithm based on the ants' foraging behaviour. ACO algorithm was first proposed in Dorigo's doctoral dissertation [11] and more details of this algorithm were provided in paper [14]. The ACO implementation to solve the traveling salesman problem (TSP) and a survey of other ACO applications were presented in paper [12].

Since the birth of the ACO algorithm, there were many researchers conducted in-depth studies and proposed vari-

ous improved versions. For instance, some variants of ACO such as elite ant colony algorithm [14], rank-based ant colony algorithm [6], max–min ant colony optimization algorithms [42] and ant colony system algorithm [12] were developed. These improved ACOs focused on either selecting which best solutions for pheromone updates or improving the candidate selection mechanism. Furthermore, ACO algorithm was applied not only for solving TSP [1,12,17,28,48] but also for other optimization problems such as vehicle routing problem (VRP) [5,18,33,38,39,53], quadratic assignment problem (QAP) [9,19], and job-shop scheduling problem (JSP) [24,25,54].

In the latest ACO survey papers [13,29], the authors stated that the most current research activities in this area were focusing on (1) incorporating ACO with other meta-heuristics such as simulate annealing [3,31,32], genetic algorithm [8], and tabu search [15], particle swarm optimization [21,30], and (2) various applications of ACO. Although there are many theoretical studies on the convergence [4,22,41], runtime and parameter tuning [10,34,43,55] for ACO, few improved ACO was proposed after the year 2006. Max–min ACO and ant colony system are still state-of-the-art ACOs according to Dorigo and Stützle's survey paper

✉ Yahui Liu
liuyahui@tongji.edu.cn

Buyang Cao
caobuyang@tongji.edu.cn

Hehua Li
hhli@sspu.edu.cn

¹ School of Software Engineering, Tongji University, Shanghai, China

² College of Architecture and Urban Planning, Tongji University, Shanghai, China

³ School of Economics and Management, Shanghai Polytechnic University, Shanghai, China

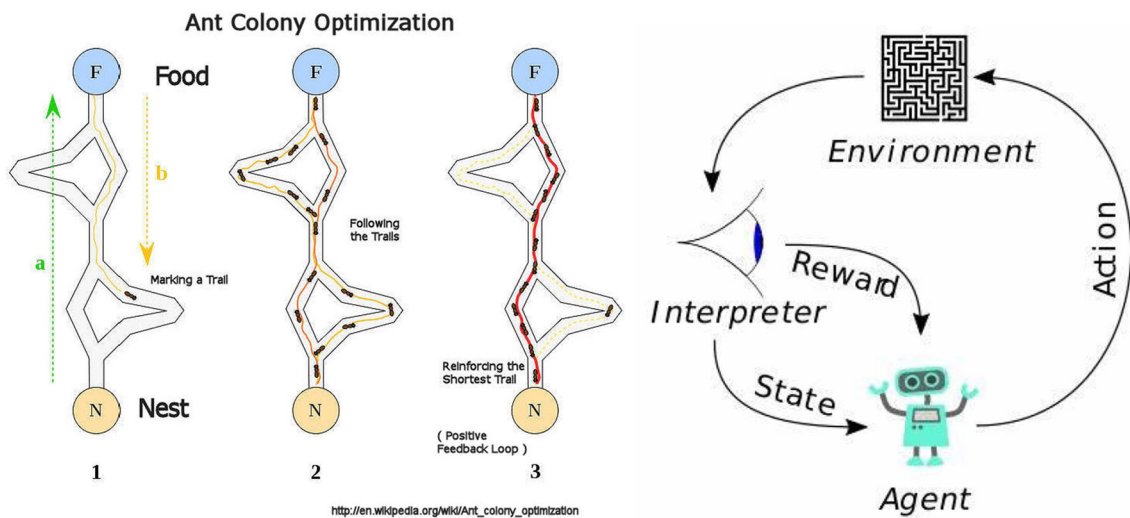


Fig. 1 ACO prototype (left) and reinforcement learning diagram (right)

in 2019 [13], so max–min ACO is selected in the benchmark of this paper.

ACO is a kind of reinforcement learning algorithm [12, 16]. In Fig. 1, the ants, pheromone, road network in ACO are equivalent to the agents, reward, environment in Reinforcement Learning. “One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation” [44]. The proposed greedy–Levy ACO is designed to resolve the exploration/exploitation dilemma.

In reinforcement learning, important exploration strategies include epsilon greedy (ϵ -greedy) and Boltzmann exploration (Softmax) policies [44]. The ϵ -greedy policy adopts the current best selection from candidates with the probability of ϵ , and a random selection with the probability of $1 - \epsilon$. Ant colony system (ACS) algorithm employs the ϵ -greedy policy and achieves better performance [12]. Greedy–Levy ACO integrates ϵ -greedy policy and employs the Levy flight mechanism attempting to improve the ϵ -greedy policy further.

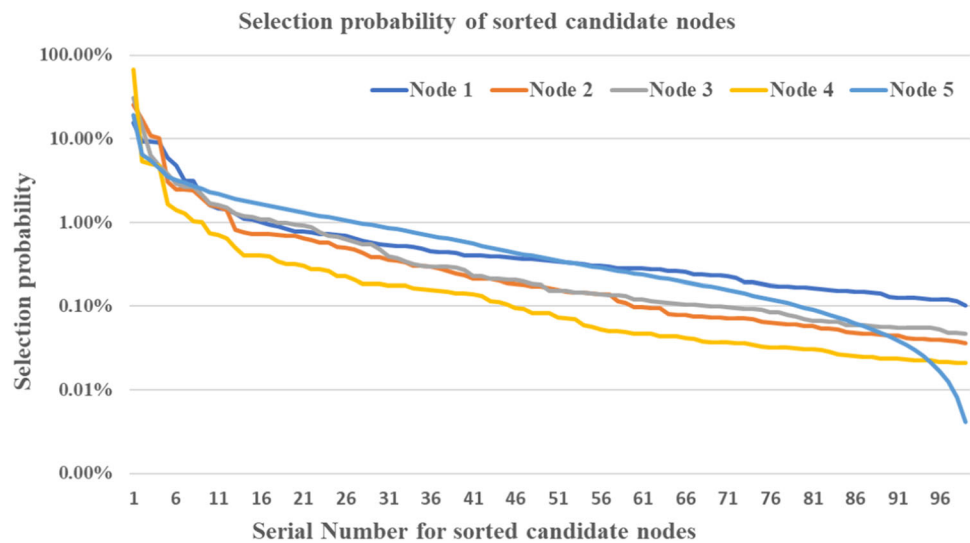
In swarm intelligence and evolutionary computation, local search focuses on the feasible neighborhood space of the current solution and seeks better solutions. For a local search, there is no guarantee to obtain a global optimum. Global search tries to explore all feasible solutions in the solution space to achieve the global optimum; however, it is quite time-consuming if not impossible. Combing global and local search is an important research method for swarm intelligence and evolutionary computation [20,35,36,47]. In this paper, the proposed algorithm attempts to balance global and local search by applying epsilon greedy and Levy flight in ACO so that better solutions can be found more efficiently.

The random mechanism is embedded in ACO algorithms, and the probability distribution for choosing the next node to be visited plays a key role in solving a TSP. Among the continuous distributions, one attracting our attention has the property called *fat tailed* or *heavy tailed*, which means the tail of this distribution is thicker than others such as normal or exponential distribution. In a fat-tailed distribution case, the tail portion will have a higher probability to be chosen by ACO algorithms for the diversity purpose. Furthermore, the increase in diversity of solutions would be more likely to find an optimal solution. Specifically, Levy distribution possesses the fat-tailed feature and it may help to deliver diversified solutions efficiently.

Levy flight [40] is a type of random walking pattern conforming to Levy distribution named after the French mathematician Paul Levy. The *step length* of Levy flight follows the fat-tailed distribution. Steps have isotropic random directions when walking in a multi-dimensional space. Many animals’ foraging movements have Levy flight features, e.g., they spend most feeding time around a current food source, and occasionally need long-distance travel to find the next food source efficiently [45,46]. Levy flight has also been applied to improve other meta-heuristic algorithms such as particle swarm optimization (PSO) [23,26,50], artificial bee colony algorithm [2], cuckoo search algorithm [51,52], etc. Levy flight mechanism has been employed in spatial search approaches, i.e. PSO and cuckoo search algorithm, but cannot directly be applied in ACO without any special design.

In this paper, both ϵ -greedy policy and Levy flight approaches are employed in the proposed greedy–Levy ACO aiming to improve searching speed and efficiency and resolve the exploration/exploitation dilemma.

Fig. 2 Selection probability of sorted candidate nodes for a instance with 100 nodes



Methods

Candidate selection mechanism in ACO

ACO algorithm consists of three major steps including constructing a solution, optional daemon actions, and pheromone updates. Constructing a solution means to find feasible solutions regarding a specific problem. Daemon actions are some optional operations, i.e., certain local search to improve current solution quality. Updating pheromone according to the solution quality is important for fast convergence and guiding the upcoming ants to seek better solutions in ACO.

Most ACOs employ a uniform-distribution-based random number between 0 and 1 as the probability to select one of the candidates when constructing a feasible solution. In the following discussions, a *candidate* means a node to be visited by an ant at a certain stage when solving a TSP, and the probability formula to select the next node from candidates is listed in formula (1); the parameter τ represents pheromone value and η represents the attractiveness which in most occasions is the reciprocal of edge length between node i and j , α and β are exponents for τ and η , respectively.

$$P_{ij} = \begin{cases} \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{s \in \text{allowed}} (\tau_{is})^\alpha (\eta_{is})^\beta}, & j, s \in \text{allowed} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The selection probabilities for candidates are exponentially distributed due to the power function of attractiveness η in formula (1). The list of candidates sorted in descending order of their selection probabilities is called *candidate list* in the following discussions. The selection probability for a candidate declines quickly with the decreasing of its attractiveness and close to zero if it is not located in the front part of the candidate list. Figure 2 depicts the selection probabilities computed for 5 nodes in the candidate list of a TSP instance

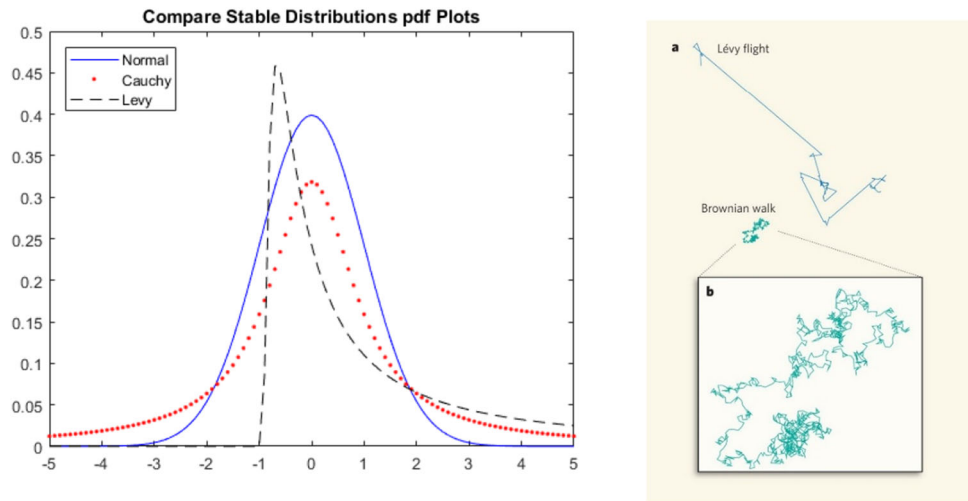
with 100 nodes, where the x -axis represents indices of nodes in the candidate list. Since nodes are sorted in descending order of their selection probabilities, a node with a smaller index means its selection probability is relatively higher than those nodes with greater indices. Selection probabilities of most nodes are less than 1% in Fig. 2 which means they almost have no chance to be chosen. This limits the selection for candidates who have a lower probability to be selected and compromise the exploration of solution spaces. The purpose of this paper is to improve the exploration strategy and, therefore, improves the diversity of solutions for achieving a better solution faster.

ϵ -Greedy policy

ACO is one type of reinforcement learning algorithm [12,16]. The ϵ -greedy and Softmax policies are important exploration strategies in reinforcement learning [44]. The ϵ -greedy policy defines the policy of the selection probability p in formula (2), which focuses on the exploitation with a probability of ϵ by using the best candidate while conducting the exploration with a probability of $1 - \epsilon$ by applying P_{ij} defined in formula (1). The ϵ -greedy policy balances the exploration and exploitation well via parameter ϵ and is widely employed in many artificial intelligent algorithms [35–37,49]. The ϵ -greedy policy is also adopted in an improved ACO named ant colony system (ACS) as the pseudo-stochastic mechanism [12]. The original ϵ -greedy policy uses uniform distribution to select a candidate in the case of $1 - \epsilon$. Levy flight mechanism will be employed to improve the case of $1 - \epsilon$ in this paper.

$$p = \begin{cases} \arg \max\{(\tau_{ij})^\alpha (\eta_{ij})^\beta\}, & \text{if } p \leq \epsilon \\ P_{ij}, & \text{otherwise} \end{cases} \quad (2)$$

Fig. 3 Normal vs Levy distribution (left) and Levy flight vs Brownian walk (right)



Levy flight and Levy distribution

The left part of Fig. 3 shows the difference between Levy, normal and Cauchy distributions. Levy distribution is a fat-tailed one in which possibility values at the tail of its curve is larger than other distributions. The right part of Fig. 3 from paper [45] depicts the Brownian motion upon uniform distribution and Levy flight following Levy distribution. The search area covered by Levy flight is much larger than the one by Brownian motion within the same 1000 steps. Part b in the right part of Fig. 3 illustrates the detailed trajectory of corresponding Brownian motion bouncing mainly around the current spot with a step length of 1. In Levy flight, flying distance is defined as the *step length* in this paper and is sometimes greater than 1.

The standard Levy distribution is given in formula (3):

$$S = \begin{cases} \frac{\mu}{v^{1/\beta}}, & \text{if } S > 1 \\ 1, & \text{else} \end{cases}, \quad \mu \sim N(0, \sigma_\mu^2), v \sim N(0, \sigma_v^2). \tag{3}$$

Formula (3) illustrates how the step length S is computed, which is the most important part of Levy flight. Parameters μ and v follow a normal distribution while β is a fixed parameter. Step length is a non-negative random number following Levy distribution and is associated with the direction that is uniformly distributed in two- or three-dimension depending on particular applications. There is no direction needs to be considered when Levy flight is one-dimensional. The step length will be applied as an altering ratio of selection probability to choose a candidate in greedy–Levy ACO.

Integration of epsilon greedy and Levy flight mechanism

Computation of original Levy flight using formula (3) is complicated and cannot directly be employed in ACO. In the proposed greedy–Levy ACO, a Levy flight conversion formula is designed for candidate selection mechanism using formula (6).

Formula (4) is an improved version of formula (3), two normal distribution-based random numbers μ and v are required in the latter while only one uniform distribution-based random variable P_{Levy} is required in the former to reduce computational cost. Altering ratio A and Levy flight threshold $P_{threshold}$ are fixed parameters in formula (4). Formula (5) is specially designed to ensure that the altered selection probability still ranges between 0 and 1. Formula (6) is derived by combining formulas (4) and (5).

$$S_{new} = \begin{cases} \frac{1}{A} \times \frac{1 - P_{threshold}}{1 - P_{Levy}}, & \text{if } S_{new} \geq 1 \\ 1, & \text{else,} \end{cases} \tag{4}$$

$$1 - P_{new} = \frac{1}{S_{new}} \times (1 - P_{now}), \tag{5}$$

$$P_{new} = \begin{cases} 1 - A \times \frac{1 - P_{Levy}}{1 - P_{threshold}} \times (1 - P_{now}), & \text{if } P_{Levy} \geq P_{threshold} \\ P_{now}, & \text{else,} \end{cases} \tag{6}$$

- S_{new} : new step length for Levy flight, $S_{new} \geq 1$;
- A : altering ratio for Levy flight, $A \geq 0$;
- $P_{threshold}$: parameter for Levy flight threshold, $0 < P_{threshold} < 1$;
- P_{Levy} : probability for turning on/off Levy flight altering, a uniform distribution based random number, $0 < P_{Levy} < 1$;
- P_{now} : original selection probability before Levy flight altering, a uniform distribution-based random number, $0 < P_{now} < 1$;

- P_{new} : final selection probability after Levy flight altering, $0 < P_{\text{new}} < 1$.

Formula (2) of ϵ -greedy presents two scenarios in selecting a candidate, (a) choosing the candidate with the maximum probability when $p \leq \epsilon$, and (b) selecting a candidate randomly when $p > \epsilon$. Formula (7) is the core design in this paper by replacing the random selection in formula (2) with the Levy flight mechanism in formula (6).

$$P_{ij} = \begin{cases} \arg \max\{(\tau_{ij})^\alpha (\eta_{ij})^\beta\}, & \text{if } p \leq \epsilon \\ 1 - A \times \frac{1 - P_{\text{levy}}}{1 - P_{\text{threshold}}} \times \left(1 - \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{is})^\alpha (\eta_{is})^\beta}\right), & \text{if } p > \epsilon, P_{\text{levy}} \geq P_{\text{threshold}} \\ \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{is})^\alpha (\eta_{is})^\beta}, & \text{else.} \end{cases} \quad (7)$$

Since the candidate selection mechanism is the only difference between greedy–Levy ACO and max–min ACO, we present the pseudocode of candidate selection mechanism for them in Algorithms 1 and 2, respectively. We refer the interesting reader to the max–min ACO paper [42] for details.

Algorithm 1 The candidate selection mechanism in greedy–Levy ACO

- 1: Sort the nodes in the candidate list by their probabilities of being selected
 - 2: Remove the nodes cannot be selected in the candidate list, i.e. visited node
 - 3: Generate a uniform random number P_{now} between 0 and 1
 - 4: **if** $P_{\text{now}} \leq \epsilon$ **then**
 - 5: **return** The next node with the max probability of being selected from the candidate list.
 - 6: **else**
 - 7: Generate another uniform random number P_{levy} between 0 and 1
 - 8: **if** $P_{\text{levy}} \geq P_{\text{threshold}}$ **then**
 - 9: $P_{\text{new}} = 1 - A * \frac{1 - P_{\text{levy}}}{1 - P_{\text{threshold}}} * P_{\text{now}}$
 - 10: **else**
 - 11: $P_{\text{new}} = P_{\text{now}}$
 - 12: **end if**
 - 13: **return** The next node be selected using P_{new} from the candidate list
 - 14: **end if**
-

Algorithm 2 The candidate selection mechanism in max–min ACO

- 1: Generate a uniform random number P_{now} between 0 and 1
 - 2: **return** The next node be selected using P_{now} from the candidate list
-

Computational experiments

Data and environment settings

Computational experiments were carried out to benchmark the proposed greedy–Levy ACO against max–min ACO and the last TSP solvers.

The code of greedy–Levy ACO was implemented on the top of publicly available source code <http://www.aco-metaheuristic.org/aco-code/> developed by Thomas Sttze

(the author of max–min ACO). The source code covers max–min ACO and other improved ACO algorithms including basic ant system, elitist ACO, rank-based ACO, best–worst ACO and ant colony system. Though TSP is a classic and well-studied combinatorial problem, it is an NP-hard problem used quite often for ACO performance benchmark. Furthermore, max–min ACO source code used in this paper is merely designed for TSP and supports the standard TSPLIB instances with best-known solutions. Every benchmark case ran 100 trials for each TSPLIB instance considering the stochastic property of ACO algorithm, with same parameter setting for all trials on the same computer. Twelve instances including ch150, kroA200, kroB200, gr202, ts225, tsp225, pr226, gr229, gil262, a280, pr299 and lin318 from TSPLIB <https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp> were selected for the benchmark. Parameters used both in max–min and greedy–Levy ACOs were the same, specifically, the parameter vector (pheromone evaporation rate, alpha, beta, population size) was set to be (0.1, 1, 2, 50). The maximum running time for each trial was 86,400 s attempting to obtain the best-known solutions. Source code and running script of max–min ACO and greedy–Levy ACO are available at <https://github.com/akeyliu/greedylevyacotsp/>.

Both max–min ACO and greedy–Levy ACO in this paper employ a *pheromone re-initialization mechanism*. It monitors the solution procedure continuously and resets all pheromones to their max value when it finds the pheromones are concentrated on few edges and believes the solution procedure is trapped at a local optimum. This mechanism attempts to escape local optima and find the best-known solution within predefined iterations and/or computational time. It helps max–min ACO and greedy–Levy ACO to find the best-known solutions for many small- or medium-scale

TSPLIB instances including all ones used in this paper when enough time and iterations are given. During the benchmark, both max–min ACO and greedy–Levy ACO employ other improvement procedures including nearest neighbor and the 3-opt local search. The iterations when the best-known solutions are obtained were recorded for comparison in the experiment as an algorithm performance metric, which includes the iterations of 3-Opt local search which are the major iterations in max–min ACO and greedy–Levy ACO.

The computing environment for the benchmark was Windows 10 × 64, CPU 8 cores at 2.7 GHz, Memory 32 GB. The programming language for the implementation is C to keep consistent with max–min ACO algorithm.

Parameters tuning for greedy–Levy ACO

Larger epsilon greedy threshold, smaller Levy flight threshold and Levy flight altering ratio will speed up the convergence, however, easy to fall into local optimal solutions. On the other hand, more exploration will help to seek more solution space and yet require prolonged searching time or more iterations. It is critical to find the most reasonable parameter setting to balance the exploration/exploitation and converge to the optima faster.

Additional computational experiments were conducted for tuning the parameters specific for greedy–Levy ACO, namely epsilon greedy threshold ϵ , Levy flight threshold $P_{\text{threshold}}$ and Levy flight altering ratio A in formula (7). Twelve instances (ch150, kroA200, kroB200, gr202, ts225, tsp225, pr226, gr229, gil262, a280, pr299 and lin318) were chosen from the TSPLIB instances to perform parameters tuning. For each parameter setting to be evaluated, 100 trials were carried out, and a few metrics are applied to determine the best parameter setting.

Parameter tuning

Parameter ϵ in formula (2) varying with value 0, 0.5, 0.6, 0.7, 0.8, 0.9 were evaluated. The larger the ϵ value is, the less exploration is. Epsilon greedy will be switched off if ϵ value is set to 0.

Parameter Levy flight threshold in formula (2) varying from 0 to 1 with a step length of 0.05 was evaluated and analyzed. The smaller Levy flight threshold is, the more exploration is. Parameter Levy flight altering ratio in formula (2) varying from 0 to 2 with a step length of 0.2 was evaluated as well. The larger Levy flight altering ratio is, the more exploration is. Levy flight will be switched off if Levy flight threshold value is set to 0 or altering ratio is set to 0.

Greedy–Levy ACO degenerates to max–min ACO when epsilon greedy and Levy flight are both switched off.

There were 726 parameter combinations for epsilon, Levy flight threshold and altering ratio in the experiment. Each

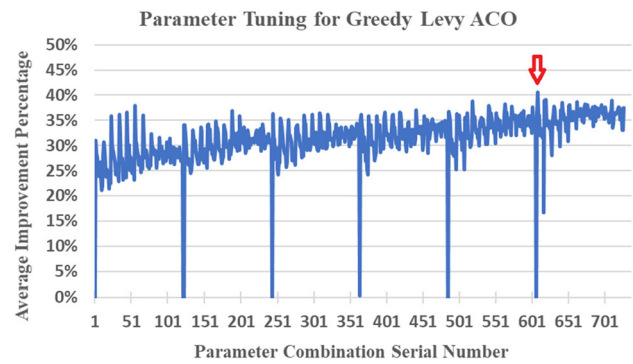


Fig. 4 Average performance improvement percentage for parameter tuning

parameter combination ran 100 trials for all instances and all trials found the best-known solutions.

To measure the effectiveness of certain parameter setting, the metrics called *iteration improvement percentage* was defined as followed:

For the given set of parameters, let $N_{mm(i)}$ be the average of iterations to find the best-known solution of the i th instance in 100 trials for max–min ACO while $N_{gl(i)}$ be the one required for greedy–Levy ACO, then the average of iteration improvement percentage over K instances is defined as $\sum_{i=1}^K (1 - N_{gl(i)}/N_{mm(i)})/K$, and K is 12 in our experiment. It is conceivable that the higher the average iterations improvement percentage is, the better the performance of greedy–Levy ACO is for the given parameter setting.

The average of iteration improvement percentage for 726 parameter combinations is illustrated in Fig. 4. No. 609 parameter combination is the best one as shown in Fig. 4 where epsilon is 0.9, Levy flight threshold is 0 and altering ratio is 0.4, and this parameter setting is employed for later computational experiments in the following sections.

Benchmarks between max–min ACO and greedy–Levy ACO

The computational experiments were conducted by employing instances ch150, kroA200, kroB200, gr202, ts225, tsp225, pr226, gr229, gil262, a280, pr299 and lin318 from the TSPLIB, and the results using the suggested parameter setting for greedy–Levy ACO and max–min ACO during the experiments are plotted in Fig. 5.

The histograms in Fig. 5 depict the iterations where the best-known solutions of the tested TSP instances are obtained. The x -axis represents the experiment trials while the y -axis indicates the iterations when the best-known solution is reached. The best-known solutions for all instances were obtained in the experiment. By glancing at these histograms, it appears that the required average of iterations to reach the best-known solutions using greedy–Levy ACO is

Fig. 5 Benchmark for 12 instances

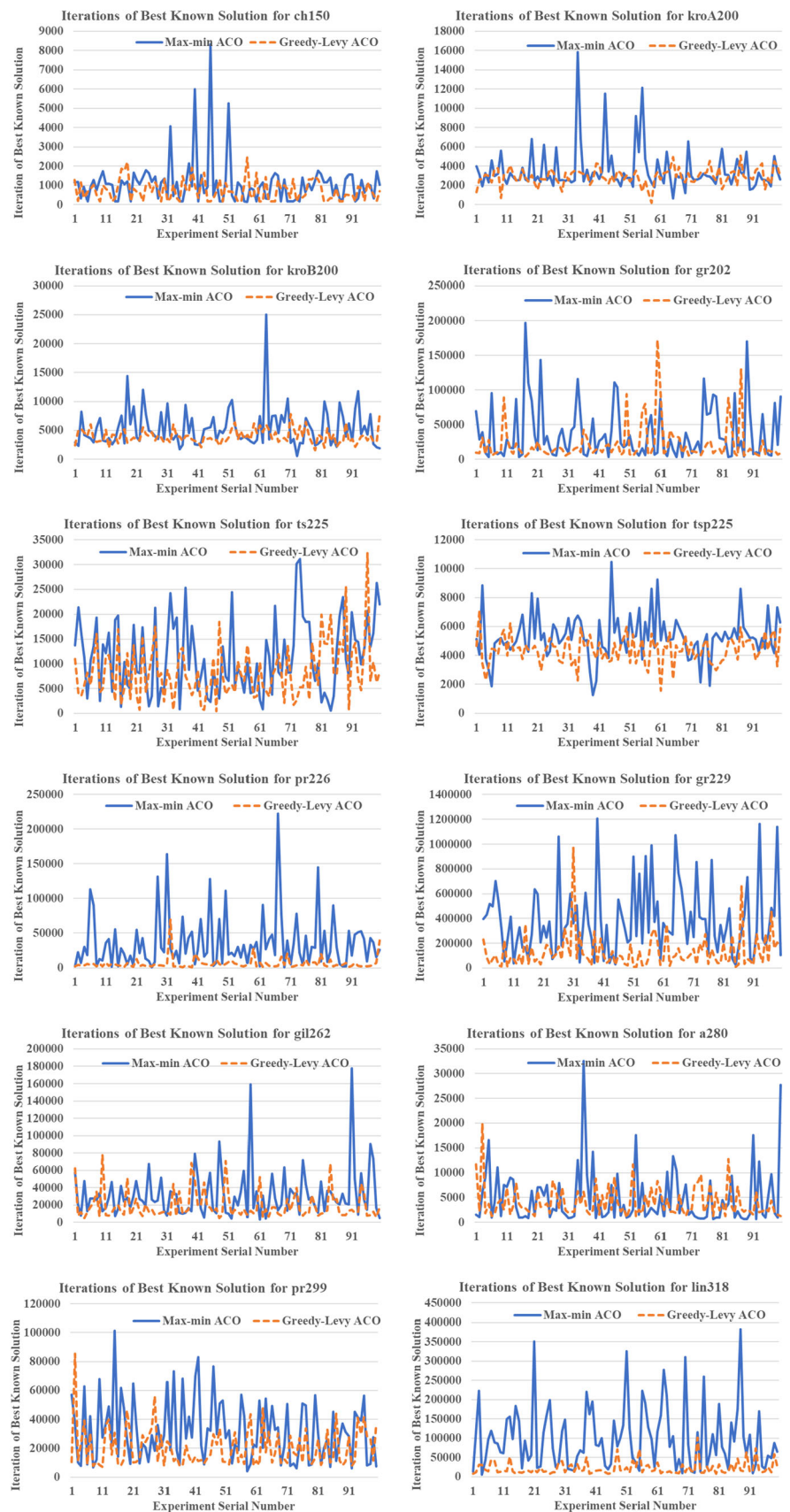


Table 1 Statistical testing (P value) for the benchmark analyses

Instance	Wilcoxon		Rank sums		Mann–Whitney U	
ch150	6.03E–04	***	5.29E–03	**	2.68E–03	**
a280	0.3173		0.9972		0.5	
kroA200	0.2818		0.8523		0.4275	
kroB200	0.1201		0.2096		0.1054	
gr202	0.0112	*	0.0274	*	0.0138	*
pr226	3.04E–07	***	3.92E–08	***	2.00E–08	***
ts225	3.09E–03	**	1.35E–03	**	6.81E–04	***
tsp225	3.29E–03	**	2.36E–03	**	1.20E–03	**
gr229	2.86E–05	***	4.35E–07	***	2.21E–07	***
pr299	1.02E–04	***	6.12E–04	***	3.10E–04	***
gil262	3.12E–04	***	1.13E–04	***	5.74E–05	***
lin318	9.53E–09	***	1.05E–10	***	5.37E–11	***

P value: < 0.05*; < 0.01**; < 0.001***

Table 2 Benchmark for max–min ACO and greedy–Levy ACO

Instance Name	Max–min ACO		Greedy–Levy ACO		Improvement percentage	
	Average iterations	Iteration variance	Average iterations	Iteration variance	Average iterations (%)	Iteration variance (%)
ch150	1087.2	1160.61	731.33	522.8509	32.73	54.95
a280	4788.89	5565.808	4042.42	3094.291	15.59	44.41
kroA200	3546.18	2201.672	2921.12	833.6642	17.63	62.13
kroB200	5311.75	3320.334	3956.49	1210.959	25.51	63.53
gr202	37,367.48	39,292.75	21,829.34	27,124.36	41.58	30.97
pr226	35,757.25	38,332.23	6260.73	8624.176	82.49	77.50
ts225	11,380.65	7249.736	8343.69	5525.991	26.69	23.78
tsp225	5292.14	1514.43	4386.75	936.5603	17.11	38.16
gr229	38,1295.5	27,8537	12,9097.82	13,8805.4	66.14	50.17
pr299	31,672.3	20,896.26	20,279.83	13,403.4	35.97	35.86
gil262	31,992.94	27,604	18,980.36	15,854.17	40.67	42.57
lin318	99,586.9	80,655.82	21,902.39	17,059.07	78.01	78.85
Average					40.01	50.24

lower than max–min ACO. It may imply that greedy–Levy ACO can achieve the best-known solutions faster than max–min ACO does.

To validate the above statement statistically, three non-parametric tests called Wilcoxon, rank sums and Mann–Whitney U tests were applied to check if greedy–Levy ACO and max–min ACO perform similarly. For each TSP instance, the iterations of these two algorithms to reach the best-known solutions for all 100 trials were input to the function `scipy.stats.wilcoxon()`, `scipy.stats.ranksums()` and `scipy.stats.mannwhitneyu()` in Python packages. The outcomes are presented in Table 1. P value for a280 and kroA200 is litter higher that means the performances of the compared algorithms are not so differently statistical in solving this instance because the iterations finding the best-known solution is small (about 2000–6000 iterations). For

other instances, the performances of underlying algorithms are significantly different since the associated P values are less than 0.05 or even 0.001. Together with the histograms, we conclude that greedy–Levy ACO can achieve best-known solutions faster than max–min ACO.

To demonstrate that the proposed algorithm can indeed reach the best-known solutions faster, further statistical analyses are conducted upon the achieved results. In Table 2, the computational results are listed for greedy–Levy ACO and max–min ACO. The following metrics are applied to both approaches:

- Average iterations: the average iterations to reach the best-known solution which used to evaluate the performance of the algorithm.

Table 3 Benchmark for ant colony system and greedy–Levy ACO

Instance Name	Ant colony system		Greedy–Levy ACO		Improvement percentage	
	Average iterations	Iteration variance	Average iterations	Iteration variance	Average iterations (%)	Iteration variance (%)
ch150	1456.26	1250.36	664.38	507.6224	54.38	59.40
a280	4796.58	5291.796	3916.24	2814.883	18.35	46.81
kroA200	3401.82	1758.022	2977.34	928.4029	12.48	47.19
kroB200	4968.78	2680.551	4165.26	1412.987	16.17	47.29
gr202	41,052.22	42,848.12	25,744.6	34,003.09	37.29	20.64
pr226	43,402.3	45,222.89	6557.5	7231.195	84.89	84.01
ts225	11,328.6	7534.245	6576.96	4590.283	41.94	39.07
tsp225	5542.8	1473.737	4356.44	953.5582	21.40	35.30
gr229	34,5145	255,690.3	131,221.8	129,044.4	61.98	49.53
pr299	29,620.52	23,064.15	19,985.02	11,755.06	32.53	49.03
gil262	38,359.66	26,804.65	16,907.28	12,542.05	55.92	53.21
lin318	98,783.58	102,143.3	23,979.12	19,811.32	75.73	80.60
Average					42.76	51.01

Table 4 Iterations for greedy–Levy ACO to achieve the best-known solution

Trials	berlin52	ch150	eil51	eil76	eil101	korA100	korB200	lin105	rat99	st70
1	135	711	142	505	1075	188	3976	188	307	139
2	155	512	136	515	599	176	4069	182	770	146
3	131	2079	133	383	1377	184	2391	203	758	160
4	141	1322	394	640	294	173	3549	190	445	285
5	139	508	883	654	303	164	2260	183	635	145
6	139	820	268	255	146	167	3187	197	805	144
7	137	1162	541	508	1062	353	4501	192	787	147
8	156	501	139	131	1078	354	3474	177	475	140
9	145	1044	133	400	932	174	2487	193	326	138
10	135	1047	134	270	1209	356	3609	198	314	144
11	136	1497	134	374	447	157	1097	181	786	145
12	148	1166	139	130	1047	332	2900	203	315	144
13	135	351	265	519	1509	182	2201	199	315	143
14	144	692	138	385	443	177	3915	213	787	138
15	141	670	141	466	297	185	225	204	160	135
16	145	168	140	251	636	334	2970	187	308	138
17	140	507	121	269	459	181	3018	198	475	413
18	145	1171	268	630	939	176	3279	201	1646	149
19	151	1138	136	521	896	351	2981	194	1214	152
20	139	827	135	381	1531	171	3174	185	646	141
Max iterations	156	2079	883	654	1531	356	4501	213	1646	413
Average iterations	141.85	894.65	226	409.35	813.95	226.75	2963.15	193.4	613.7	164.3

*The bold number means achieve the best-known solution within 1000 iterations

– Iteration variance: the variance of iterations to reach the best-known solution which applied to judge the stabilization of the algorithm.

Table 2 also presents the improvement magnitude of greedy–Levy ACO over max–min ACO for all tested TSP instances concerning the above metrics. Taking the results shown in Tables 1 and 2 into account, we may conclude that

Table 5 Best solutions for greedy–Levy ACO in 1000 iterations

Trials	berlin52	ch150	eil51	eil76	eil101	korA100	korB200	lin105	rat99	st70
1	7542	6528	426	538	631	21,282	29,409	14,379	1211	675
2	7542	6528	426	538	629	21,282	29,451	14,379	1211	675
3	7542	6549	426	538	630	21,282	29,394	14,379	1211	675
4	7542	6533	426	538	629	21,282	29,492	14,379	1211	675
5	7542	6528	426	538	629	21,282	29,463	14,379	1211	675
6	7542	6528	426	538	629	21,282	29,486	14,379	1211	675
7	7542	6544	426	538	630	21,282	29,487	14379	1211	675
8	7542	6528	426	538	630	21,282	29,409	14,379	1211	675
9	7542	6533	426	538	629	21,282	29,446	14,379	1211	675
10	7542	6543	426	538	630	21,282	29,533	14,379	1211	675
11	7542	6533	426	538	629	21,282	29,417	14,379	1211	675
12	7542	6533	426	538	630	21,282	29,420	14,379	1211	675
13	7542	6528	426	538	632	21,282	29,451	14,379	1211	675
14	7542	6528	426	538	629	21,282	29,507	14,379	1211	675
15	7542	6528	426	538	629	21,282	29,368	14,379	1211	675
16	7542	6528	426	538	629	21,282	29,436	14379	1211	675
17	7542	6528	426	538	629	21,282	29,501	14,379	1211	675
18	7542	6543	426	538	629	21,282	29,550	14,379	1212	675
19	7542	6533	426	538	629	21,282	29,469	14,379	1212	675
20	7542	6528	426	538	631	21,282	29,478	14,379	1211	675
best	7542	6528	426	538	629	21,282	29,368	14,379	1211	675
Average	7542	6532.6	426	538	629.6	21,282	29,458.35	14,379	1211.1	675
Worst	7542	6549	426	538	632	21,282	29,550	14,379	1212	675

greedy–Levy logic does help to achieve best-known solutions faster.

Ant colony system (ACS) is another state-of-the-art ACO algorithm applied with epsilon greedy strategy [13]. The result of the benchmark between ACS and greedy–Levy ACO is presented in Table 3. The epsilon parameter for ACS is 0.9 which is same in greedy–Levy ACO. The performance of greedy–Levy ACO was better than ACS from the benchmark.

Benchmarks against the latest TSP solver

To benchmark the proposed algorithm against state-of-the-art TSP solvers developed based on ACO, PSO–ACO–3Opt [30] and PACO–3Opt [21] that had been cited the most lately in Google scholar are selected. The algorithms hybridize particle swarm optimization, ant colony optimization and 3-Opt. The corresponding paper presented computational experiments for a set of TSP instances, in which each instance was solved with 20 trials and within max 1000 iterations per trial to evaluate the performance. Specifically, the TSPLIB instances berlin52, ch150, eil101, eil51, eil76, kroA100, kroB200, lin105, rat99 and st70 were employed to perform the corresponding computational experiments.

To conduct a fair comparison, the proposed greedy–Levy ACO was applied to solve the same 10 instances mentioned

above, where the best-known solutions were obtained. For each instance, the iterations to achieve the best-known solution for the corresponding 20 trials are listed in Table 4. The best solutions found in 1000 iterations are presented in Table 5. Furthermore, 3-Opt in both greedy–Levy ACO and max–min ACO were enabled.

PSO–ACO–3Opt and PACO–3Opt algorithm could not reach all the best-known solutions within the given 1000 iterations according to the outcomes in papers [21,30]. The details of the performance comparison between PSO–ACO–3Opt/PACO–3Opt and greedy–Levy ACO within 1000 iterations are presented in Table 6. Given 1000 iterations, the proposed greedy–Levy ACO can achieve better solutions comparing to PSO–ACO–3Opt/PACO–3Opt.

Conclusions

In this paper, the proposed greedy–Levy ACO algorithm was developed on the top of max–min ACO by applying ϵ -greedy policy and Levy flight mechanism. The parameters of greedy–Levy ACO were tuned carefully using associated instances. The computational experiments reveal the superiority of the proposed greedy–Levy ACO. It is observed that greedy–Levy ACO can reach the best-known solutions with

Table 6 Best solutions obtained in 1000 iterations by compared algorithms

Instance	PSO-ACO-3Opt				PACO-3Opt				Greedy-Levy ACO				
	Best-known Solution	Best	Average	Worst	Error%	Best	Average	Worst	Error%	Best	Average	Worst	Error%
Berlin52	7542	7542	7543.2	7548	0.02	7542	7542	7542	0.00	7542	7542	7542	0.00
Ch150	6528	6538	6563.95	6622	0.55	6570	6601.4	6620	1.12	6528	6532.6	6549	0.07
Eil51	426	426	426.45	428	0.11	426	426.35	427	0.08	426	426	426	0.00
Eil76	538	538	538.3	539	0.06	538	539.85	542	0.34	538	538	538	0.00
Eil101	629	629	632.7	638	0.59	629	630.55	639	0.25	629	629.6	632	0.10
kroA100	21,282	21,301	21,445.1	21,554	0.77	21,282	21,326.8	21,382	0.21	21,282	21,282	21,282	0.00
kroA200	29,368	29,468	29,646.1	29,957	0.95	29,533	29,644.5	29,721	0.94	29,368	29,458.35	29,550	0.31
Lin105	14,379	14,379	14,379.2	14,381	0.00	14,379	14,393	14,422	0.10	14,379	14,379	14,379	0.00
Rat99	1211	1224	1227.4	1230	1.35	1213	1217.1	1225	0.50	1211	1211.1	1212	0.01
St70	675	676	678.2	681	0.47	676	677.85	679	0.42	675	675	675	0.00
Average					0.49				0.40				0.05

$$\text{Error\%} = (\text{AverageSolution} - \text{BestKnownSolution}) / \text{BestKnownSolution}$$

fewer iterations comparing to max–min ACO algorithm (an average 40.01% deduction in iterations and 50.24% deduction in iteration variance for all tested TSPLIB instances). Further, other ACO researchers may be able to replicate the experiments or improve their ACO algorithm by integrating greedy–Levy mechanism based on the uploaded open-source code.

For future research, we are planning to apply greedy–Levy ACO to other optimization problems, i.e. rich VRP [7,27].

Acknowledgements This work was partially supported by the National Natural Science Foundation of China under Grant number 41771410.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Ariyasingha I, Fernando T (2015) Performance analysis of the multi-objective ant colony optimization algorithms for the traveling salesman problem. *Swarm and Evolutionary Computation* 23:11–26
2. Aydođdu I, Akın A, Saka MP (2016) Design optimization of real world steel space frames using artificial bee colony algorithm with levy flight distribution. *Adv Eng Softw* 92:1–14
3. Azar D, Fayad K, Daoud C (2016) A combined ant colony optimization and simulated annealing algorithm to assess stability and fault-proneness of classes based on internal software quality attributes. *Int J Artif Intell* 14(2):137–156
4. Badr A, Fahmy A (2004) A proof of convergence for ant algorithms. *Inf Sci* 160(1–4):267–279
5. Bell JE, McMullen PR (2004) Ant colony optimization techniques for the vehicle routing problem. *Adv Eng Inform* 18(1):41–48
6. Bullnheimer B, Hartl RF, Strauss C (1997) A new rank based version of the ant system. A computational study
7. Caceres-Cruz J, Arias P, Guimarans D, Riera D, Juan AA (2015) Rich vehicle routing problem: Survey. *ACM Comput Surv* 47(2):32
8. Dai Y, Lou Y, Lu X (2015) A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi-QoS constraints in cloud computing. In: 2015 7th international conference on intelligent human-machine systems and cybernetics, IEEE, vol 2, pp 428–431
9. Demirel NÇ, Toksarı MD (2006) Optimization of the quadratic assignment problem using an ant colony algorithm. *Appl Math Comput* 183(1):427–435
10. Doerr B, Neumann F, Sudholt D, Witt C (2011) Runtime analysis of the 1-ant ant colony optimizer. *Theoret Comput Sci* 412(17):1629–1644
11. Dorigo M (1992) Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano

12. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans Evol Comput* 1(1):53–66
13. Dorigo M, Stützle T (2019) Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*. Springer, Berlin, pp 311–351
14. Dorigo M, Maniezzo V, Colomi A et al (1996) Ant system: optimization by a colony of cooperating agents. *IEEE Trans Syst Man Cybern B Cybern* 26(1):29–41
15. Drias Y, Kechid S, Pasi G (2016) A novel framework for medical web information foraging using hybrid aco and tabu search. *J Med Syst* 40(1):5
16. Gambardella LM, Dorigo M (1995) Ant-q: a reinforcement learning approach to the traveling salesman problem. *Machine learning proceedings 1995*. Elsevier, New York, pp 252–260
17. Gambardella LM, Dorigo M (1996) Solving symmetric and asymmetric TSPs by ant colonies. In: *Proceedings of IEEE international conference on evolutionary computation*, IEEE, pp 622–627
18. Gambardella LM, Taillard É, Agazzi G (1999) Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In: *New ideas in optimization*, Citeseer
19. Gambardella LM, Taillard ÉD, Dorigo M (1999) Ant colonies for the quadratic assignment problem. *J Oper Res Soc* 50(2):167–176
20. Guimarães FG, Campelo F, Igarashi H, Lowther DA, Ramírez JA (2007) Optimization of cost functions using evolutionary algorithms with local learning and local search. *IEEE Trans Magn* 43(4):1641–1644
21. Gülcü Ş, Mahi M, Baykan ÖK, Kodaz H (2018) A parallel cooperative hybrid method based on ant colony optimization and 3-opt algorithm for solving traveling salesman problem. *Soft Comput* 22(5):1669–1685
22. Gutjahr WJ (2002) Aco algorithms with guaranteed convergence to the optimal solution. *Inf Process Lett* 82(3):145–153
23. Hariya Y, Kurihara T, Shindo T, Jin'no K (2015) Lévy flight PSO. In: *2015 IEEE congress on evolutionary computation (CEC)*, IEEE, pp 2678–2684
24. Heinonen J, Pettersson F (2007) Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Appl Math Comput* 187(2):989–998
25. Huang RH, Yang CL, Cheng WC (2013) Flexible job shop scheduling with due window a two-pheromone ant colony approach. *Int J Prod Econ* 141(2):685–697
26. Jensi R, Jiji GW (2016) An enhanced particle swarm optimization with levy flight for global optimization. *Appl Soft Comput* 43:248–261
27. Lahyani R, Khemakhem M, Semet F (2015) Rich vehicle routing problems: from a taxonomy to a definition. *Eur J Oper Res* 241(1):1–14
28. Li Y, Gong S (2003) Dynamic ant colony optimisation for TSP. *Int J Adv Manuf Technol* 22(7–8):528–533
29. López-Ibáñez M, Stützle T, Dorigo M (2016) Ant colony optimization: a component-wise overview. *Handbook of heuristics*, pp 1–37
30. Mahi M, Baykan ÖK, Kodaz H (2015) A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Appl Soft Comput* 30:484–490
31. Mohsen AM (2016) Annealing ant colony optimization with mutation operator for solving tsp. *Comput Intell Neurosci* 2016:20
32. Moussi R, Euch J, Yassine A, Ndiaye NF et al (2015) A hybrid ant colony and simulated annealing algorithm to solve the container stacking problem at seaport terminal. *Int J Oper Res* 24(4):399–422
33. Narasimha KV, Kivelevitch E, Sharma B, Kumar M (2013) An ant colony optimization technique for solving min-max multi-depot vehicle routing problem. *Swarm Evol Comput* 13:63–73
34. Neumann F, Witt C (2009) Runtime analysis of a simple ant colony optimization algorithm. *Algorithmica* 54(2):243
35. Qian C, Yu Y, Zhou ZH (2015) Pareto ensemble pruning. In: *Twenty-ninth AAAI conference on artificial intelligence*
36. Qian C, Shi JC, Tang K, Zhou ZH (2017) Constrained monotone k -submodular function maximization using multiobjective evolutionary algorithms with theoretical guarantee. *IEEE Trans Evol Comput* 22(4):595–608
37. Raykar V, Agrawal P (2014) Sequential crowdsourced labeling as an epsilon-greedy exploration in a Markov decision process. In: *Artificial intelligence and statistics*, pp 832–840
38. Reed M, Yiannakou A, Evering R (2014) An ant colony algorithm for the multi-compartment vehicle routing problem. *Appl Soft Comput* 15:169–176
39. Schyns M (2015) An ant colony system for responsive dynamic vehicle routing. *Eur J Oper Res* 245(3):704–718
40. Shlesinger MF, Klafter J (1986) Lévy walks versus lévy flights. *On growth and form*. Springer, Berlin, pp 279–283
41. Stutzle T, Dorigo M (2002) A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans Evol Comput* 6(4):358–365
42. Stützle T, Hoos HH (2000) Max-min ant system. *Future Gener Comput Syst* 16(8):889–914
43. Stützle T, López-Ibáñez M, Pellegrini P, Maur M, De Oca MM, Birattari M, Dorigo M (2011) Parameter adaptation in ant colony optimization. *Autonomous search*. Springer, Berlin, pp 191–215
44. Sutton RS, Barto AG (2018) *Reinforcement learning: an introduction*. MIT Press, New York
45. Viswanathan GM (2010) Ecology: fish in lévy-flight foraging. *Nature* 465(7301):1018
46. Viswanathan GM, Afanasyev V, Buldyrev S, Murphy E, Prince P, Stanley HE (1996) Lévy flight search patterns of wandering albatrosses. *Nature* 381(6581):413
47. Weise T, Wu Y, Chiong R, Tang K, Lässig J (2016) Global versus local search: the impact of population sizes on evolutionary algorithm performance. *J Glob Optim* 66(3):511–534
48. Hf Wu, Chen XQ, Mao QH, Zhang QN, Zhang SC (2013) Improved ant colony algorithm based on natural selection strategy for solving TSP problem. *J China Inst Commun* 34(4):165–170
49. Wunder M, Littman ML, Babes M (2010) Classes of multiagent q -learning dynamics with epsilon-greedy exploration. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, Citeseer, pp 1167–1174
50. Yan B, Zhao Z, Zhou Y, Yuan W, Li J, Wu J, Cheng D (2017) A particle swarm optimization algorithm with random learning mechanism and levy flight for optimization of atomic clusters. *Comput Phys Commun* 219:79–86
51. Yang XS, Deb S (2009) Cuckoo search via lévy flights. In: *2009 world congress on nature and biologically inspired computing (NaBIC)*, IEEE, pp 210–214
52. Yang XS, Deb S (2014) Cuckoo search: recent advances and applications. *Neural Comput Appl* 24(1):169–174
53. Yu B, Yang ZZ, Yao B (2009) An improved ant colony optimization for vehicle routing problem. *Eur J Oper Res* 196(1):171–176
54. Zhang J, Hu X, Tan X, Zhong JH, Huang Q (2006) Implementation of an ant colony optimization technique for job shop scheduling problem. *Trans Inst Meas Control* 28(1):93–108
55. Zhou Y (2009) Runtime analysis of an ant colony optimization algorithm for TSP instances. *IEEE Trans Evol Comput* 13(5):1083–1092