Swarm refinement combinatorial ABC for solving n-queens problem

Joan T. Matamalas and Albert Clapés

***Abstract*—In this work, we propose a swarm refinement combinatorial ABC for solving n-queens problem. We implement an ABC from scratch and compare two candidate solutions' initializations, the random permutation and the refinement initialization. Our proposal intends to be a new approach in combinatorial optimization methods that are trying to solve the commonly raised n-queens problem, complementing others already existing methods. We show that the ABC method is suitable to solve CSPs like the n-queens problem, just adapting some parts of the original real-valued version of the algorithm.**

I. Introduction

Artificial **B**ee **C**olony (ABC) is a relatively recent new member of swarm intelligence. It was first introduced by Karaboga [10] in 2005. ABC tries to model natural behaviour of real honeybees in food foraging. Honeybees use several mechanisms like *waggle dance*[1] to optimally locate food sources and to search new ones. This makes them good candidate for developing new intelligent search algorithms. The algorithm is inspired by the previous work of Tereshko who mathematically modeled the behavior of the honeybee colony [19].

The n-queens puzzle is a problem that consists in placing n queens in a n by n board in such a way none of them can attack any of the others, i.e. no two queens share the same row, column or diagonal. This problem falls in the special case of problems (NP-hard) whose cannot be solved in polynomial time. The n-version case is a generalization of the original posed problem, $n = 8$ (seen in Fig. 1), firstly brought by the german chess player Max Bezzel in 1848, in the Berliner Schachzeitung [3]. Over many years, many mathematicians tackled this problem. Nauck was the first to solve completely the 8-queens problem, in 1850, by finding all the 92 solutions [16]. Gauss is often cited as the originator of this problem or the first to solve it, but this is a case of 'broken telephone'; he had found only 72 of the 92 total solutions given by Nauck. Later, in 1874, Pauls gave the first proof that n non-attacking queens can be placed on the n by n chessboard and proved the problem was complete [147,148].

[1]A particular figure-eight dance of the honey bee. By performing this dance, successful foragers can share with other members of the colony, information about the direction and distance to patches of flowers yielding nectar and pollen, to water sources, or to new housing locations.

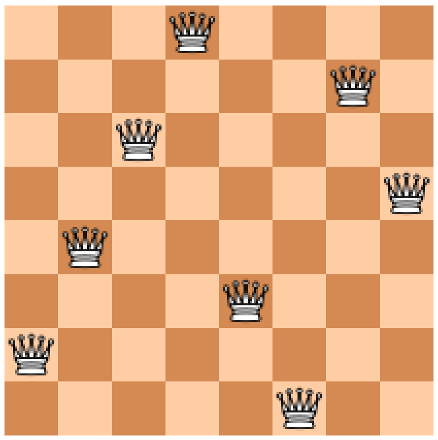

Fig. 1: A solution for the original 8-queens problem.

Combinatorial search problems, as this one, form a set of problems which need a considerable effort and time to be solved. The difficult lie in the fact there is not a formula for solving them exactly. Every possibility has to be examined in order to find a satisfactory solution (constraint satisfaction problem) or the best solution (constraint optimization problem); and, particularly, the n-queens problem is a constraint satisfaction problem (CSP). The naïve approach to solve this kind of problems by brute-force techniques can be computationally expensive as there are $\binom{n^2}{n}$ possible arrangements of n on a board n by n. Constraining each queen to a single column (or row), though still brute force, the number of possible solutions is reduced to n^n. And, eventually, generating the permutations that are solutions of the rooks puzzle and then checking for diagonal attacks further reduces the possibilities to 'just' $!n$.

Expressing the candidate solution as an n-tuple $(i_1, ..., i_n)$, in which the index indicates the column and the value the row of a certain queen, implicitly constraints the solution space. Thus, finding a solution can be seen as finding a permutation of a sequence of integer numbers expressing a valid solution (an arrangement of non-colliding queens), which is not trivial as, for instance, placing n rooks would be. The conditions for a permutation to represent a valid solution are equivalent to the arithmetical question of whether the first n positive integers can be arranged in the tuple, such that the difference between any two entries is not equal to the absolute value of the difference in their indices. This type of arithmetic representation was used by Gauss in [8] in analyzing the original 8-queens problem. Therefore, generally speaking, the n-queens problem is a typical permutation problem. It has been set as a benchmark for backtracking algorithms, dividing conquer paradigms and evolutionary computational algorithms.

The n-queens problem has three different variants: (a) finding the complete set of solutions, (b) finding a subset (family) of solutions, or (c) finding the very first solution. So far, there is no closed form expression for the total number of solutions for a board of arbitrary size, but $n!$ is clearly an upper bound. The number of solutions for particular values of n is available in Sloane's [17], where *Sequence A000170* is the total number of solutions. Regarding the later problem, the problem is also still opened, and after having set some background, next, we focus in it.

In the context of computing the very first solution, the classical approach is the trial and error to then perform backtracking if collision is detected when placing a queen. Apart from this approach, another very active area of research of the state-of-the-art literature are global and non-linear combinatorial optimization techniques.

The Particle Swarm Optimization (PSO) is a population-based stochastic optimization technique inspired by the social behavior of bird flocks [15]. Unlike in Genetic Algorithms (GA), there are not evolution operators (crossover or mutation), but instead the particles, representing potential solutions, fly through the problem space following the better-performing particles. In 2003, Hu and Eberhart proposed a PSO approach to solve the n-queens problem [9]. In traditional PSO, the velocity is added to the particle on each dimension to update the particle; if the new velocity is larger, the particle may explore more distant areas. Similarly, the new velocity in this permutation scenario represents the possibility that the particle changes; if the velocity is larger, the particle is more likely to change to a new permutation sequence.

In 2011, Draa et al., proposed a hybridization between differential evolution (DEA) algorithm and quantum computing principles, called Quantum-Inspired Differential Evolution (QDEA), for solving the n-queens problem [5]. A QDEA is a QDA with quantum coding solutions. This representation reduces the computation time by decreasing the number of chromosomes, since each chromosome does not encode only one solution but all the possible solutions by putting them within a superposition [6].

A Genetic algorithm (GA) is an adaptive search technique, which grew out of Holland's study of adaptation in artificial and natural systems. The signal parent genetic algorithm (PGA), introduced in 2012, is a new kind of derived genetic algorithm, which has advantages of optimization of combination problem. PGA deletes the crossover operator and designs new mutation operators instead, which is operator on only signal parent to generate new chromosome. It can simplify the GA operation, promote the computing efficient and prevent premature convergence. Wang and Liu, in [20], proposed an improved PGA to solve the n-queens problem, which is based on gene block of knowledge.

Later approaches made use of the refinement technique to greatly reduce the conflicts among queens. In 2012, Wang and Lin, proposed a PSO with swarm refinement for solving the n-queens problem [21]. This technique consists in placing queens randomly on non-occupied columns, and after a certain number of queens are placed without conflict, the remaining ones are placed randomly on free columns regardless of conflicts on diagonals.

In this work, we propose a swarm refinement combinatorial ABC for solving n-queens problem. We implement an ABC from scratch and compare two candidate solutions' initializations, the random permutation and the refinement initialization. Our proposal intends to be a new approach in combinatorial optimization methods that are trying to solve the commonly raised n-queens problem, complementing others already existing methods.

The paper is organized as follows: Section II introduces some foundations about the method, the algorithm insights, and the different candidate solution generation methods. Then, in Section III, the experimental results are given, a statistical validation is performed, and discussion takes place. Finally, the paper concludes with some conclusions and future work in Section IV.

II. Proposed method

A. Foundations of ABC

The **A**rtificial **B**ee **C**olony (ABC) is one of the called swarm intelligence algorithms. Algorithms that try to find the optimal value of some fitness function inspiring his functionality on the social behaviour of some biological systems like ants, fish or, in our case, bees.

The ABC foundations are related to the honeybee (*Apis mellifera*) nectar recollection behaviour. The 'intelligence' of the system emerges from the low and the high level defined by the sociocognitive theory.

The low level components related to the behaviour of each individual [14] are:

- **Evaluation:** the tendency to rate stimuli as positive or negative, attractive or repulsive.
- **Compare:** the capability to make comparisons between the own state and the state of the nearest individuals.
- **Imitate:** behaviour imitation of the individuals that have desired properties.

The high component levels are related to the pattern formation between individuals and the problem solving capability. This properties are embedded in the honeybee social organization, eusociality, the highest level of social organization. Eusociality is characterized by overlapping adult generations, cooperative brood care and division of labor by reproductive and (partially) non-reproductive groups [4].

Tereshko defined the model of the behaviour of honeybee colony based in their communication in the hive using for this purpose reaction-diffusion equations [19]. The three basic components of this model are:

- **Food source:** the election of the food source depends on its evaluations based in some factors like: proximity, quality, visibility, and so forth.
- **Employed foragers bees:** bees that are currently exploiting a source food. This kind of bee knows information about the food source and share it with other bees in an area near to the hive entrance, performing a *waggle dance*.
- **Unemployed foragers bees:** This kind of bees doesn't have any assigned source food and in order to find one they can:
  - Search source foods randomly on the environment (between a 5% and a 10% of the total number of unemployed bees plays this role, they are commonly known as scout bees).
  - They can choose among all the actual food sources using to perform the election the information gave by the employed foragers bees through the *waggle dance*. The election of one particular food source has an autocatalytic effect because if on bee recruits another bee, the recruit will in turn reinforce the train and recruit other nestmates [19].

Based on this publication, Devis Karaboga proposed the ABC algorithm in 2005 [10]. The insights of these algorithm are discussed in the next Section II-B. However a introductory graphical representation of the algorithm is shown in Fig. 2.

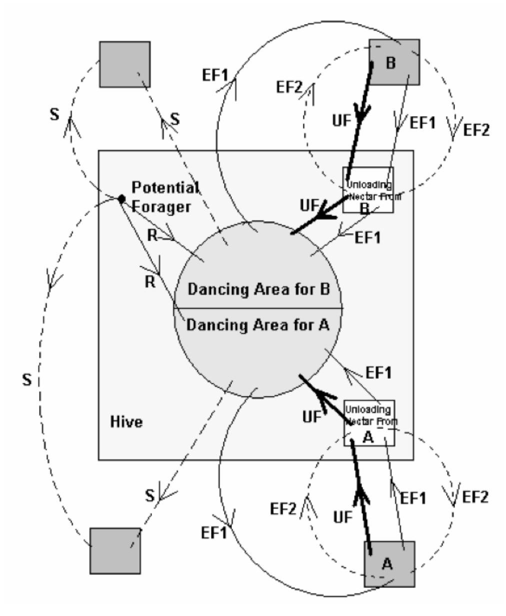

Fig. 2: Graphical representation of ABC (*S: Scout, UF: Unemployed forager, EF: Employed forager*)

Now, when the functionality of each kind of bee has been shown, the components of the low level, previously introduced, can be detailed as follows:

- **Positive feedback:** as the goodness of a food source increases, the number of bees that exploit it also increases *(autocatalytic effect).*

- **Negative feedback:** If there aren't any improvement in the quality of the food source after *n* evaluation, the source is discarded.
- **Fluctuations:** The scouts bees carry out the exploration process seeking food in random points of the environment.

An introductory view of the algorithm is shown in Algorithm 1.

```
Data: N number of initial food sources and a
      fitness function
Result: The optimized value of the fitness function
initialization;
while not finalization conditions do
    Set the employed foragers bees. One on each
    food source position;
    Allocate the unemployed foragers bees on the
    food sources proportionally to their fitness;
    Send scouts to find new food sources;
    Store the best food position explored so far;
end
```

Algorithm 1: Basic ABC pseudo-code

B. Algorithm insights

One of the main advantages of the ABC is the reduced set of control parameters needed to run the algorithm:

- Number of food sources, candidate solutions. This value is fixed to 50 because Karaboga showed that this value perform well in a set of different situations [11]. This food sources are codified as a permutation of *n* integers, each position of the permutation represents a row of the chessboard and the number that contains the column where the queen is placed. Using this representation, as was said in the introduction, the optimization has to lead to a non diagonal collisions solution. The process to initialize this food sources are explained in the Section II-C. An array of the same size of the solution is initialized with all its values to zero. This array contain the number of times that a given food source is exploited without get an improvement.
- Number of maximum failed exploits searches[2]. When this number is reached the food source will be abandoned. In the initial version of the ABC [12], a real optimization version, this value is set using the number of dimensions of the problem, *D*, and the number of food sources *SN*: $limit = SN \cdot D$. The algorithm proposed in this paper tries to perform a combinatorial optimization, taking this into account Karaboga introduced [13] a new version for combinatorial problems of the limit value computation: $limit = \frac{SN \cdot D}{3}$

A deep explanation of the steps involved is are exposed in the next lines:

1) Each possible food source have an employed forager bee assigned. This bee exploits the food source changing one dimensional value *j* of the solution selected uniformly random. This change is made by imitation of one of the other bees in the environment. In the next lines, two methods to perform this action are presented. The first one correspond to the real number optimization version of the algorithm, and was proposed by Karaboga [12]. The second one is the combinatorial version, and it is was proposed explicitly for this paper. The formulation is based in some way in the version proposed in [13] in order to solve a COP[3] like TSP. We must recall, however, that the motivation of this work is to solve the n-queens problem that is a CSP[4] and, thus, this implies having a slightly different approach.
   a) **Real valued approach:** the value of the dimension *j* is computed using the formula $v_i^j = x_i^j + \phi(x_i^j - x_k^j)$ where x_i is the actual position of the bee, x_k is the position of another food source on the environment (x_i and x_j must be different) chosen uniformly random, and ϕ is a value drawn from a uniformly random distribution from the interval [-1, 1].
   b) **Combinatorial approach:** the previous method is adapted to perform a combinatorial permutation of the problems. A food source x_k is randomly chosen from the environment and the value of the its dimension *j*, x_k^j, is placed in the position *j* of x_i^j. This action requires a permutation to keep the consistency of the solution, Fig. 3.
2) When the new candidate solution is generated, a comparison with the previous one is performed. There are two possibilities:
   a) If the new solution is worse than the previous one, the new one is discarded and the counter of failed exploitations is incremented in one unit.

[2]An exploits that doesn't increase the fitness of the solution

[3]Constraint Optimization Problem

[4]Constraint Satisfaction Problem

b) Otherwise, the new solution is adopted as a new food source position and the counter of fail exploits is set to zero.

3) The probability that a given unemployed forager bee goes to a given food source is given by the fitness of the food source[5]:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (1)$$

Based in this probability, the SN unemployed bees are distributed among the several food sources, using a rank based system implemented with a Stochastic Universal Sampling (SUS) [2].

4) The step 1 and 2 are repeated for each *recruited* bee.

5) The best solution so far is stored.

6) If there are any food sources that their counter have reached the limit value of exploitations, these food sources are removed and new food sources are generated, Section II-C.

When the problem is a minimization problem, like the n-queens problem, some modifications have to been added to the fitness function. Final functions will be: $fit_i = \frac{1}{1+fx_i}$ where $f(x)$ is the function previously detailed.

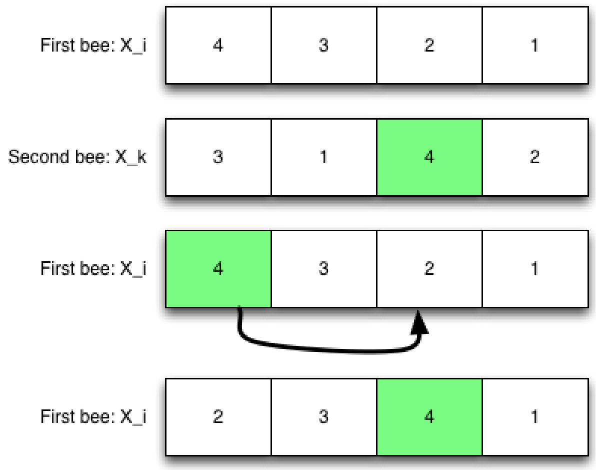

Fig. 3: Exploit dimension update process.

C. Candidate solution generation methods

As it has been exposed in the previous Section, two methods of generation of the candidate solutions will be used and then compared (Section III-B).

The first approach is a naïve one, random permutation of an array of n elements.

The second one is a little more sophisticated and it is based on the method introduced by Sosic [18], refinement, and used by Wang [21] in a similar problem, like was exposed in Section I.

1) Refinement: The steps used in the refinement initialization, Fig. 4, are as follows [21]:

1) **Place the first queen:** The first position of a food source is an integer randomly generated from the range of 1 to n, the total number of queens.
2) **Place non-collision queens:** After placing the first queen, the next queen must be placed in a zero collision position with respect to the first queen. Then, the algorithm continues placing the queens until no zero collision positions can be found.
3) **Place rest of the queens:** It is possible that there are queens that cannot be placed in zero collision position. In this case, the rest of the places are randomly set.

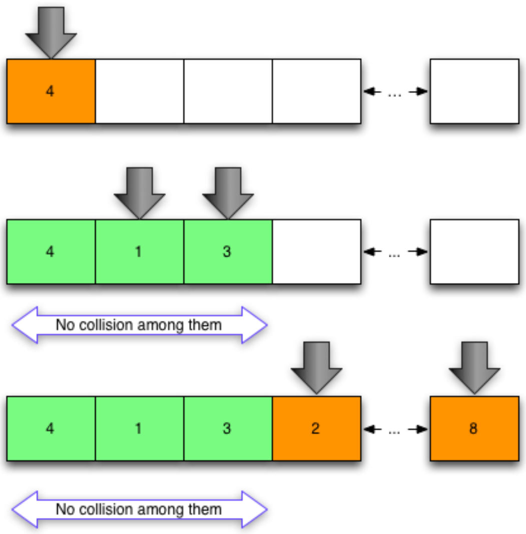

Fig. 4: Refinement steps

III. Experimental Results

The main objective of this work is to perform an evaluation of the ABC algorithm in order to resolve CSPs, like the n-queens problem. At the same time, an evaluation in terms of efficiency (speed) is performed

[5]In the case studied in this document, a food source that contains a solution with fewer collisions among queens is considered better than the other

between the two initialization methods exposed in Section II-C.

A. Metrics

Several metrics will be used in order to evaluate the experiment results:

- **Success rate:** This metric gives information about the number of correctly finished executions. This is an execution that found a solution where no collisions exist among the queens before the algorithm reaches the maximum limit of iterations. This value was set to 50,000 iterations.
- **Statistics about the number of iterations needed to find a solution:** These values include the average, the standard deviation, the standard error of means, minimum value, and maximum value.
- **Statistics about the number of seconds required to found a solution:** The use of refinement is computationally expensive compared with the naïve random permutation method. So, the same statistics used to evaluate the number of iterations are used to summarize the required time.

B. Experimental methodology

The algorithm will be tested for different number of dimensions[6], specifically: 20, 40, 60, 80, 100, 120. For each number of dimensions, the algorithm is run 31 time with different random seeds for each initialization method. The number of executions is chosen in order to work with a normal distribution derived from the implications of the Central Limit Theorem (CLT):

> The CLT says that if a sample is sufficiently large (n>30), no matters whatever is the distribution of the variable, the distribution of the sample mean will be approximately normal. In addition, the mean will be the same than the one of the variable, and the standard deviation of the sample mean will be approximately the standard error [1].

After running all the executions, a statistical tests will be performed comparing the results obtained for both initialization methods. The objects of this tests will be the average number of iterations until goal is reached and the time needed to do it. In order to perform this comparison, we perform a Student's independent two-sample and two-tailed t-test with a significance level $\alpha = 0.05$.

[6]The number of queens

Subject to there we have to perform several comparisons, one for each number of queens. The Boole's inequality has to be taken into account:

> In the probability theory, the Boole's inequality specify that for all finite family or numerable succession, the probability of at least one happens is less than the addition of the probability of the individual events [1].

In order to avoid the false positives derived from this inequality, the use of the Bonferroni correction [7] is recommended. The basic functionality is as follows:

- The set of comparisions is ordered in reverse order by their *p-value*.
- As a significance level α for each comparison the value of the initial alpha divided by the position of the comparison in the ranking is chosen.

This method has some drawbacks. The main problem is that as we decrease the probability of make false positives (FPs), we increase the probability of make false negatives (FNs). However, the number of comparisons performed will be low, so this implication can be somehow ignored.

C. Software organization

The algorithm is codified using the Python language, and to be executed it is required the installation of a set of third party open source frameworks. The requirements are:

- **NumPy:** This software is used to compute the main statistical values as average, stdDev or SEM.
- **SciPy:** Used to perform the t-test.
- **Matplotlib:** Used to plot the results.

The code is divided into 3 files:

- **nqExperiment.py:** This file stores the main execution of the experiment, runs the several executions, each one in a separate process in order to decrease the time of computation. Perform the statistical analysis, comparisions, plotting, and results summarizations.
- **nQueenABC.py:** This file stores the class that implements the ABC algorithm and all its functions needed to run the search process.
- **results.py:** In this file are codified the classes that represents the summarization of the executions for a given number of dimensions and initialization method containing the statistical data that summarizes it. Also contain a class useful to represent the results of a specific comparison.

1) Hardware used: Given the high computation cost of the experiments, a cloud computing solution was used. An instance was setup in a Amazon EC2 services, using a high computational configuration:

- 60.5 GB of RAM
- 88 EC2 Compute Units[7].
- 3370GB of storage
- 64-bits platform
- 10 Gigabit Ethernet

D. Results

In this section, the experimental results obtained from the execution will be exposed. The first one is the table summarizing, Table I, all the statistical values previously specified in the section III-A.

The first thing to comment is that a solution has been found in all the executions, achieving a success rate of 100%.

One should also notice the high standard deviation present in the version using the refinement initialization. This affects directly the minimum and maximum values, that means that in some executions the refinement version finish much faster than the random permutation version. However, the high max value of the refinement version is greater than the same value in the other version when the dimensionality increases.

This last evidence is proved by the performed Student's t-test whose results can be seen in the Table II. How can be seen, the version using refinement is statistically better when the number of dimensions do not exceed 40.

The evolution of the number of iterations needed to find one of the possibles solutions was plotted in Fig. 5. Taking into account the logarithmic scale of the plot, it can be noticed that the number of dimensions need rises exponentially as the number of dimensions increases. A normal effect in the NP-Problems.

The other variable compared is duration until the goal is reached. We perform this analysis because the time needed to generate a a candidate solution using a refinement function is computational expensive. Nevertheless, the metrics do not show any penalization for the use of this technique and displays a similar behaviour to the observed in the case of the metrics that involve the number of iterations. This phenomenon can be understood reviewing the plot with the duration evolution respect to the number of dimensions, Fig. 6, and also in the results of the Student-t test of the Table III.

[7]1 EC2 unit is equivalent to a 1-1.2GHz Opteron or Xeon

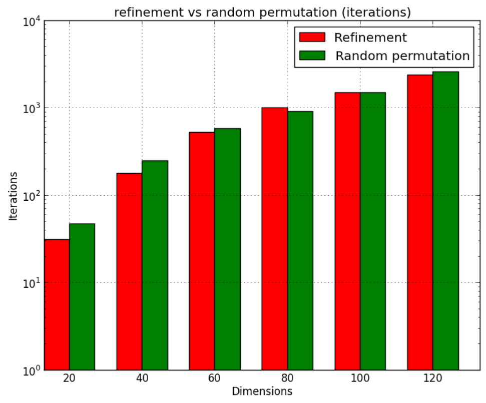

Fig. 5: Log number of iterations for each value of n

Dim.	t-value	p-value	Rank	α	Best Alg.
100	0.0398	0.968	1	0.05	-
120	-0.690	0.492	2	0.025	-
80	1.069	0.289	3	0.016	-
60	-1.118	0.267	4	0.0125	-
20	-2.839	0.006	5	0.01	refinement
40	-2.890	0.005	6	0.008	refinement

TABLE II: Comparative between refinement and random permutation (Iterations)

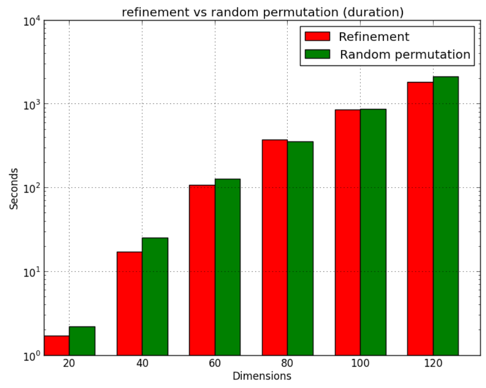

Fig. 6: Log number of seconds for each value of n

A possible explanation is that the limit value for exploiting a given food source is too high and the use of refinement becomes marginal when the number of dimensions increases over a determinate level. A tuning and selection of the limit value could be a possible research line in the future, as will be explained in the

	20-r	20	40-r	40	60-r	60	80-r	80	100-r	100	120-r	120
S.R	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
I. AVG	30.0968	45.8387	176.516	248.645	522.387	583.903	1011.26	907.71	1491.45	1485.32	2376.42	2605.61
I. STD	21.6174	21.3316	92.5213	100.597	233.663	189.981	462.745	258.745	666.246	516.711	1364.45	1200.23
I. SEM	0.0756397	0.0801627	1.24505	1.28633	7.15768	5.64502	22.0753	14.0191	49.1316	42.6524	148.202	126.809
I. MIN	1	4	46	73	84	278	198	451	641	429	278	1077
I. MAX	88	105	349	520	900	992	2345	1618	3300	3024	6788	5955
D. AVG	0.709162	1.19233	16.2135	24.005	105.458	125.355	369.592	353.745	847.484	858.322	1815.73	2104.16
D. STD	0.414296	0.439069	6.81941	7.04553	39.2042	30.9191	120.911	76.786	269.105	233.617	811.738	694.561
D. SEM	0.0756397	0.0801627	1.24505	1.28633	7.15768	5.64502	22.0753	14.0191	49.1316	42.6524	148.202	126.809
D. MIN	0.043875	0.139467	5.30934	8.37385	21.0555	69.3347	87.6609	195.055	433.617	294.636	267.676	1040.4
D. MAX	1.63519	2.19141	28.6559	41.7983	167.073	194.663	637.904	556.175	1555.86	1460.98	4296.99	4026.42

TABLE I: Statistical results for the several execution. The mark *r* on the top of the column represents a refinement result. S.R means Success Ratio. I and R represent Iterations and Duration

Section IV.

Dim.	t-value	p-value	Rank	α	Best Alg.
100	-0.166	0.868	1	0.05	-
80	0.606	0.546	2	0.025	-
120	-1.478	0.144	3	0.016	-
60	-2.182	0.032	4	0.0125	-
40	-4.352	5.323e-05	5	0.01	refinement
20	-4.383	4.771e-05	6	0.0083	refinement

TABLE III: Comparative between refinement and random permutation (Duration)

IV. Conclusions and future work

We proposed a swarm refinement combinatorial ABC for solving n-queens problem. We implemented an ABC from scratch and compared two candidate solutions' initializations, the random permutation and the refinement initialization. Our proposal intended to be a new approach in combinatorial optimization methods that are trying to solve the commonly raised n-queens problem, complementing others already existing methods. We have shown that the ABC method is suitable to solve CSPs like the n-queens problem, just adapting some parts of the original real-valued version of the algorithm.

As future work, a parameter tuning could interesting along other tests like using a higher number of dimensions to obtain a more accurate measurements about the actual performance of both initialization methods. Moreover, the exploit combinatorial method explained in the Section II-B could be changed to take into account a the goodness of the neighbors solutions in order to imitate them rather the uniformly random choice used now.

References

[1] Josep Gibergans Bàguena, Àngel J. Gil Estallo, and Carles Rovira Escofet. *Estadística*. FUOC, quarta edició edition, febrer 2009.

[2] J.E. Baker. Reducing bias and inefficiency in the selection algorithms. 1985.

[3] FWM Bezzel. Proposal of eight queens problem. *Berliner Schachzeitung*, 3:363, 1848.

[4] J.T. Costa and T.D. Fitzgerald. Social terminology revisited: Where are we ten years later? In *Annales Zoologici Fennici*, volume 42, pages 559–564. Helsinki: Suomen Biologian Seura Vanamo, 1964-, 2005.

[5] A. Draa, S. Meshoul, H. Talbi, and M. Batouche. A quantum-inspired differential evolution algorithm for solving the n-queens problem. *neural networks*, 1:12, 2011.

[6] A. Draa, H. Talbi, and M. Batouche. A quantum-inspired genetic algorithm for solving the n-queens problem. In *Proceedings of the 7th International Symposium on Programming and Systems*, pages 145–152, 2005.

[7] O.J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[8] C.F. Gauss. *Werke: Herausgegeben von der königlichen Gesellschaft der Wissenschaften zu Göttingen*, volume 1. G. Olms Verlag, 1981.

[9] Xiaohui Hu, R C Eberhart, and 2003 SIS '03 Proceedings of the 2003 IEEE Yuhui Shi Swarm Intelligence Symposium. Swarm intelligence for permutation optimization: a case study of n-queens problem. In *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pages 243–246, 2003.

[10] D. Karaboga. An idea based on honey bee swarm for numerical optimization. *Techn. Rep. TR06, Erciyes Univ. Press, Erciyes*, 2005.

[11] D. Karaboga and B. Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132, 2009.

[12] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[13] D. Karaboga and B. Gorkemli. A combinatorial artificial bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, pages 50–53. IEEE, 2011.

[14] J. Kennedy. Thinking is social experiments with the adaptive culture model. *Journal of Conflict Resolution*, 42(1):56–76, 1998.

[15] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[16] F. Nauck. Briefwechseln mit allen für alle. *Illustrirte Zeitung 15 (377)*, page 182, September 21 ed. 1850.

[17] N. Sloane. The on-line encyclopedia of integer sequences. *Towards Mechanized Mathematical Assistants*, pages 130–130, 2007.

[18] R. Sosic and J. Gu. Efficient local search with conflict minimization: A case study of the n-queens problem. *Knowledge and Data Engineering, IEEE Transactions on*, 6(5):661–668, 1994.

[19] V. Tereshko. Reaction-diffusion model of a honeybee colonys foraging behaviour. In *Parallel Problem Solving from Nature PPSN VI*, pages 807–816. Springer, 2000.

[20] Shuli Wang, Fang Sun, and Daohua Liu. An improved single parent genetic algorithm and its application to n-queens problem. In *Oxide Materials for Electronic Engineering (OMEE), 2012 IEEE International Conference on*, pages 705 –707, sept. 2012.

[21] Y.R. Wang, H.L. Lin, and L. Yang. Swarm refinement pso for solving n-queens problem. In *Innovations in Bio-Inspired Computing and Applications (IBICA), 2012 Third International Conference on*, pages 29–33. IEEE, 2012.