Fifth International Conference
Modelling and Development of Intelligent Systems
June 23 - 25, 2017
"Lucian Blaga" University
Sibiu - Romania

An Imperialist Competitive Algorithm Optimized to Solve the Travelling Salesman Problem

Stelian Ciurea

Abstract

We studied the way in which imperialist competitive algorithms (ICAs) can be adapted to solve combinatorial problems. We chose to solve the Travelling Salesman Problem (TSP) (probably the most popular NP-complete problem). In the first stage, we combined ICA with other optimization algorithms (e.g. 2-opt or greedy algorithms) and conducted approximately 13,000 tests to establish values of ICA parameters to ensure optimal behaviour. We then optimized the ICA obtained in the first stage by various methods to increase running speed and improve the results obtained. Finally, we tested the algorithm to solve 11 reference configurations. We consider that the algorithm obtained had a good behaviour: the error obtained for a configuration consisting of over 5,900 cities was 6.54%.

1 The imperialist competitive algorithm to solve the TSP

In September 2007, Esmaeil Atashpaz Gargari and Carlos Lucas presented a paper describing a new type of evolutionary algorithm drawing from history [10]. Called the "Imperialist Competitive Algorithm" (ICA), it has as a model the competition between the imperialist states as well as the way they have led the colonizing policy of political and historical events in the seventeenth, eighteenth and nineteenth centuries. ICA is part of the category of meta-heuristic algorithms based on sets of candidate solutions, also called populations (along with genetic algorithms, cluster algorithms, gravitational search algorithms, etc.).

The standard ICA as presented in [10] is the following:

1. Generate an initial set of countries;
2. Evaluate each country and determine the imperialist countries;
3. Occupy the colonies;
4. Assimilate the colonies;
5. If a colony has better results than the imperialist country then
    a. Interchange the colony with the imperialist country
6. The imperialist competition
    a. Compute the performance of the empires
    b. Occupy the weakest colony of the weakest empire by another empire
    c. If the weakest empire has no colonies left then
7. Remove this empire

8. Revolutions occur in some colonies
9. If the stopping requirements are met then
    a. Stop
10. Otherwise
11. Repeat the algorithm from step 4.

The imperialist country that has the best results following the last iteration is the solution to the problem. Initially, the imperialist competitive algorithm was designed to determine the minimum or maximum of certain functions with one or several real arguments, hence to determine solutions consisting of one or several real numbers. Soon after it was formulated, it started to be applied to solving problems having solutions consisting of whole numbers. In the specialist literature [3], [8], this type of imperialist competitive algorithm was called "discrete algorithm". The problems that fall into this category may be traditional graph theory problems or various practical optimization problems to which solutions with polynomial complexities are not known. Of these, the best-known and the one we chose to solve is the "TSP - Travelling Salesman Problem". We consider it no longer necessary to state it. Theoretically, this is the problem of finding a minimum cost Hamiltonian cycle in a complete graph in which edges have attached costs. An exact solution to this problem can be reached by generating all permutations of the set {1,2,3, ..., n} with 1 fixed point and by calculating the corresponding cost; each permutation is a Hamiltonian cycle. As the number of these permutations is equal to (n-1)!, this method can only be applied for low values of n. TSP is important both theoretically and practically because a number of concrete problems can be formulated as TSP; the most numerous examples can be taken from the integrated circuit manufacturing industry in which situations with higher values for n (n = 744710) [1], [5] have been reported. Various applications can be found in [8]. A pertinent description of the current state of the solutions to this problem, including a list of numerous applications, can be found in [7]. Of the different variants of the TSP, we chose to solve the one called the symmetric Euclidean variant: graph nodes are points in plan, the cost of moving from one node to another is the rounded Euclid distance between the two nodes, and this cost does not depend on the moving direction. The cost of a cycle is the total distance travelled, and this is the function of evaluating a permutation.
The distinguishing features of a discrete ICA occur while implementing the following operations:

1.1 Generating the Initial Set

Each country in the initial set will represent a permutation of the set {1,2,3, ..., n} with 1 fixed point. In the applications that we ran, we chose three values for the initial number of countries: 110, 220 and 550. *We chose a generation based on greedy strategies: the second number of the permutation (corresponding to the second node) was chosen randomly, then at every step we made a list of the x nodes that are closest to the node selected in the previous step and not yet visited (this means that the rounded Euclid distances between these x nodes and the previous selected node are the smallest).* Of these x nodes, the node to be visited in the current step is randomly chosen. This eliminates the disadvantage of poor diversity. The behaviour of the algorithm for $x \in \{1;4\}$ was studied.

1.2 Assimilation

This operation is applied to colony countries. It aims to explore the space of solutions in the neighbourhood of permutations for which the evaluation function has the best values. Through this operation, the parameters that form a colony country are modified so as to converge towards parameters that form a metropolitan country.

In the case of discrete ICAs, in the vast majority of papers that implement this type of algorithm, the assimilation operation is performed as follows: we assume that the solution of the problem that is solved by ICA consists of n natural numbers, each of these numbers belonging to a given finite range. The countries in the algorithm are then combinations of such n natural numbers. Let M be a metropolis of the algorithm and C one of its colonies having the following structure:

M :	m1	m2	…	mp1-1	mp1	…	mp2	mp2+1	…	mn

C :	c1	c2	…	cp1-1	cp1	…	cp2	cp2+1	…	cn

In order to perform the assimilation, two numbers p1 and p2, $1 \leq p1 \leq p2 \leq n$ are randomly generated, and through assimilation, the content of colony C becomes identical to that of metropolis M between positions p1 and p2. The values between positions 1 and p1-1 and p2+1 and n will be those that cannot be found between positions p1 and p2 arranged in the sequence they had in permutation C before the assimilation operation:

C′ :	c1′	c2′	…	cp1-1′	mp1	…	mp2	cp2+1′	…	cn′

$ci' \in \{1,2,...,n\}\setminus\{ mp1, mp1+1 , mp2-1, mp2\}$ and $sign(\sigma(ci)-\sigma(cj)) = sign(\sigma'(ci)-\sigma'(cj))$
where σ and σ′ denote the bijective functions corresponding to permutations C and C′, and $i,j \in \{1,2,...,n\}\setminus\{p1,..,p2\}$.
The complexity of the assimilation operation having this form is Θ(n), provided that the affiliation of a value to set { mp1, mp1+1 , mp2-1, mp2} should be verified in Θ(1) (which is possible if a suitable data structure is used – e.g. a one-dimensional array having the role of a "marker vector": practically, if we note this structure with a, then a[i] = 1 if
$i \in \{ mp1, mp1+1 , mp2-1, mp2\}$, and a[i]=0 if $i \notin \{ mp1, mp1+1 , mp2-1, mp2\}$

1.3 The Revolution Operation in Discrete ICA

We implemented this operation by applying the 2-opt algorithm outlined in [2] in order to rearrange the values of a permutation representing a country. According to this algorithm, for each pair of nodes i and j, $i,j \in \{1,2,...,n\}$ and $i<j$, we test whether, by replacing a cycle corresponding to the permutation of arcs (i,i+1) and (j,j+1) with arcs (i,j) and (i+1,j+1), a smaller length circuit is obtained, and, if so, the circuit actually changes. We should note that nodes i+1,i+2,...,j-1 are covered in the same sequence but in reverse order (see Figure 1). This allows for a quick calculation of the cost of the cycle that would result from this replacement. The structure of a country changes according to this algorithm only if the length of the resulting cycle is smaller. The advantage of applying the method is that very good results are obtained from the very first iterations. The disadvantage is the high probability that the algorithm might lock into a local optimum (usually a value close to the global optimum). The 2-opt algorithm has complexity $\Theta(n^2)$. The whole ICA will have the following complexity:

$$\Theta(no\_iterations\_maximum * no\_countries * n^2) \quad (1)$$

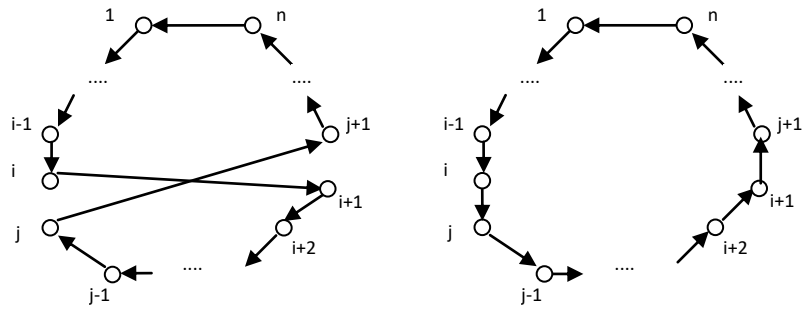

Figure 1-2: opt algorithm

The other operations typical of ICA were implemented according to the formulae presented in numerous articles (e.g. [5]).

2 Sensitivity Analysis on ICA parameters

At the beginning of the tests, we intended to set values for the ICA parameters as well as the implementation variant of the operation of generating the initial set of countries that would determine the optimal behaviour of the algorithm.

The parameters that we aimed to determine were the following: the size of the initial set (nrt), the revolution rate (rr), the approach step used in assimilation operation (p) and the weight with which a colony contributes to the value of an empire's performance (w). Taking into consideration the findings of various papers ([3], [8], [9], [10]), we chose the following sets of values out of which we aimed at selected the value that is the most suitable for these parameters:

- {110, 220, 550} for nrt;
- {4% , 10%} for rr;
- {1 , 3} for p;
- {0.1 , 0.01} for w;

Regarding the method of generating the initial set, we considered four variants based on the algorithm described above in Section 1.1 (x = 1, 2, 3 or 4).

We considered all combinations of these parameters and methods and obtained 96 variants of ICA. For a better appreciation, each of these variants was run for 25 initial sets.
The maximum number of iterations was set at 2000.
We conducted tests for eight configurations, an intrinsic one with 13 nodes and seven reference configurations (downloaded from [4]) with 29, 52, 76, 100, 152, 225 and 442 nodes).
The tests performed indicated the following:
a) The greedy generation algorithm with x=1 provides the best top and average performance;
b) The parameters influencing the performance obtained were the following:

- The revolution rate: a higher value results in better performance;
- The number of countries in the initial set: the performance increases slowly with the increase in the number of countries;
- The extent to which a colony contributes to the performance value of an empire, w, results in better performance if it is low: w = 0.01.

The runtime for the most knotted configuration (442) averaged 9.6 minutes for an initial set of 110 countries, 10.2 minutes for an initial set of 220 countries and 11.2 minutes for an initial set of 550 countries on a computer equipped with an Intel Core i3 microprocessor at 2.93 GHz.

Table 1 shows the results obtained: minL represents the best performance provided by one of the 25 initial sets and avrL represents the average performance of the 25 sets.

configuration	minL	avrL		configuration	minL	avrL
c13	51	51.0		kroA100	21282	21395.3
so29	27603	27603.0		pr152	73682	74135.3
berlin52	7542	7542.0		tsp255	3997	4050.6
pr76	108280	108786.8		d442	52123	52691.9

Table 1: Results obtained by means of the initial variant of ICA

3 Improving the combined algorithm ICA – 2-opt

3.1 Increasing algorithm performance through revolutions

The 2-opt algorithm is very powerful, but it is obvious that if we apply it to the same permutation the second time, its effect will be null. On the other hand, as ICAs run, the potential solutions we work with (the "countries", in the ICA terminology) are "attracted" to some of the best-performing permutations. This will cause the algorithm to stagnate after a fairly small number of iterations. For example, in the 25 tests performed for the 442-node configuration, the average of the last iterations whose lowest cost of a permutation was improved to 636.3.

This observation led us to the idea of improving the performance of the algorithm by modifying the revolution operation: we applied the 2-opt algorithm to 75% of the permutations selected for the revolution, while for the remaining 25% we performed the revolution operation by randomly interchanging some of the elements of the permutations.

The results obtained are presented in Table 2: ICA1 is the initial algorithm and ICA2 is the one with the modification of the implementation of the revolution operation.

configuration	ICA1		ICA2	
	minL	avrL	minL	avrL
so29	27603	27603.0	27603	27603.0
berlin52	7542	7542.0	7542	7542.0
pr76	108280	108786.8	108159	108293.1
kroA100	21282	21395.3	21282	21314.8
pr152	73682	74135.3	73682	73869.6
tsP255	3997	4050.6	3962	3997.7
d442	52123	52691.9	52128	52595.3

Table 2: The effect of the optimization of the revolution

The best behaviour of the ICA2 algorithm for all 8 configurations is evident.

3.2 Reducing the runtime

ICA2 had runtimes of approximately 1.11 hours for 1000-node configurations; a theoretical calculation indicates the value of 111 hours for configurations of 10,000 nodes, which represents a too long runtime. As a result, we aimed to reduce the runtime in various ways:

3.2.1 Software optimization (by means of programming techniques)

This optimization started from the observation made at the beginning of the Section 3 (i.e. if a permutation does not change by applying the 2-opt algorithm, then as long as the permutation does not change by other operations in ICA, there is no point in applying the 2-opt optimization). Hence, we use a one-dimensional array (array) in which we have marked, for each permutation (country), whether or not it was modified or not by applying the operations typical of ICA (assimilation or revolution). In doing so, we avoided the redundant testing of numerous permutations by the 2-opt algorithm. Table 3 shows the average number of permutations (denoted avrnbP) to which the 2-opt procedure was applied in the initial version (ICA2) and in the optimized variant (ICA3), as well as the average runtimes (avrT) in the above-mentioned system for the largest two configurations that we used in the tests performed up to that point. Obviously, the minimum lengths determined by applying the two variants of the algorithm are the same.

	ICA2		ICA3	
configuration	avrnbP	avrT (s)	avrnbP	avrT (s)
tsp225	68816	1257	36322	199
d442	91583	1402	39459	745

Table: 3 The effect of the software optimization of the algorithm

3.2.2 Optimization by increasing convergence speed

The convergence speed of the algorithm can be increased by using a variable weight for each colony that contributes to the performance of the metropolis empire. The variation principle of this parameter is that described in [9]. Thus, the algorithm we proposed for consideration computes the performance of the empires (step 6.a from the algorithm described in Section 1) using the formula:

$$results\_empire = results\_imperialist\_country + \Sigma weight\_colony \times rezult\_colony \quad (2)$$

where the result of a country is the distance travelled following the cycle induced by the permutation (each country represents a permutation).
We assigned to each colony a weight that is initially equal to 0.01. Nevertheless, each time a colony changes imperialist country, the value of the weight of that colony is diminished multiplying it by 0.75.

Table 4 shows the results of ICA with variable weight (ICA4) compared to the results obtained for ICA with constant weight (ICA3). In the tests conducted, the vm1084 configuration with 1084 nodes was also used, downloaded from[4]:

configuration	ICA3			ICA4		
	minL	avrL	avrnbIt	minL	avrL	avrnbIt
pr152	73682	73869.6	2000	73682	73829.3	1269
tsp225	3962	3997.7	2000	3931	3958.5	1429
d442	52128	52595.3	2000	52022	52281.2	1689
vm1084	251125	251403.0	2000	248263	249203.2	1993

Table 4: ICA4 with variable weight vs. ICA3 variable weight

3.2.3 Hardware optimization (by using a super-computer)

Like other evolutionary algorithms based on candidate solution sets, ICA contains an intrinsic parallelism in various stages, for example in the evaluation stage of these candidate solutions. When using a number of np processors, the complexity of the algorithm is as follows:

$$O(nritemax * \frac{nrt}{np} * n^2) \quad (3)$$

The master process and slave processes communicated through the MPI_Send and MPI_Recv functions in the mpi.h library. Table 5 shows the average runtimes obtained with the help of this computer for various configurations as well as the average runtimes by using a computer equipped with a single Intel Core i3 2.93 GHz processor.

	Average runtime (s)		Ratio timePC/timeHPC
configu-ration	PC Intel Core i3	Intel HPC System	
tsp225	195	88.2	2,2
d442	688	137.7	5.0
vm1084	4206	375.1	11.2

Table 5: Average runtime for various configurations

Although the effort to rewrite ICA for a parallel computer is not very substantial, the use of such a computer has an important disadvantage: access is quite problematic. For example, the HPC located at LBU Sibiu was not functional for rather long periods of time.

4 Final results obtained by running an optimized ICA

An ICA having the parameters and optimizations described above was run for different configurations, using other 10 sets of 550 countries. Table 6 shows the results obtained. η represents the efficiency of the algorithm and is an expression that indicates the percentage in which the algorithm improved the minimum length obtained during the first iteration of the algorithm. BKS represents the best known result for that configuration, drawn from [4]; err is the error (in percentage) of the best result compared to the BKS.

configuration	minL	avrL	avrT (s)	η%	BKS	err
so29	27603	27603.0	12.2	5.41	27603	0.00
berlin52	7542	7542.0	17.8	5.42	7542	0.00
pr76	108159	108194.8	67.6	8.34	108159	0.00
kroA100	21282	21297.7	47.5	8.71	21282	0.00
pr152	73682	73758.2	40.3	4.07	73682	0.00
tsp225	3931	3958.5	77.6	12,8	3916	0.38
d442	52022	52281.2	125,0	10,6	50778	3.62
vm1084	248263	249203.2	375,1	10,3	239297	3.74
fl1400	20456	20531.9	548.2	8.64	20127	1.63
d2103	81716	81983.8	761.4	6.88	80450	1.57
rl5915	602564	606257.6	4559.4	6,19	565530	6,54

Figure 1 illustrates how the number of colonies originally held by 50 empires varied in a test performed on a 442-node configuration. We note the convergence of the algorithm that ends after 1796 iterations through the capturing of all colonies by a single empire and the implicit disappearance of the other empires.

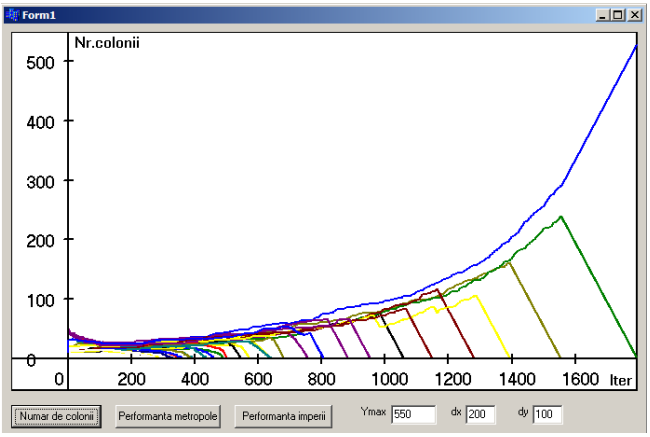

Figure 1: Number of colonies held by the empires of the algorithm dependent on the iteration

Figures 2 and 3 illustrate the evolution of the solution determined by the algorithm for various configurations. The optimum circuits determined during the first iteration, halfway through the number of iterations and at the end, are presented.

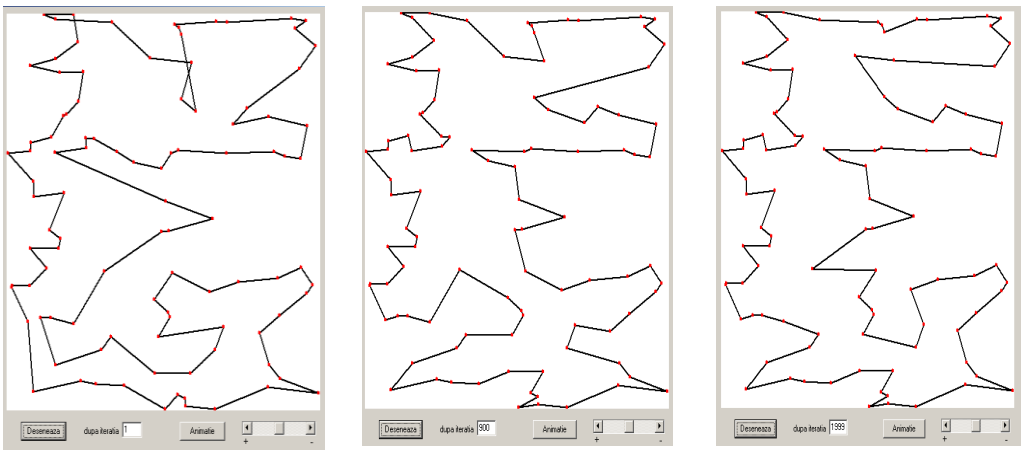

Figure 2: Evolution of the optimal circuit determined by ICA for the kroA100 configuration

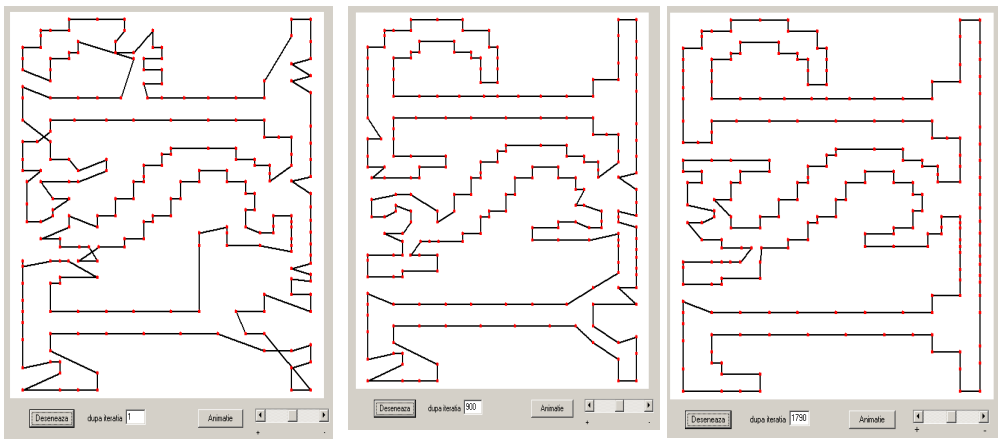

Figure 3: Evolution of the optimal circuit determined by ICA for the tsp225 configuration

5 Conclusions

ICA can be used to solve problems with solutions consisting of a string of natural numbers. The main feature of applying an ICA to this type of problem is that the implementation modalities for some of the characteristic operations differ depending on the actual problem to be solved. Discrete ICA performance can be greatly improved by its combination with other optimization algorithms, such as greedy strategies to generate the initial set of countries, or by implementing specific operations. Through various software and hardware methods, we improved the solutions and reduced the runtimes.

References

[1] Ahmadvand, M., Yousefikhoshbakht, M., Mahmoodi, D. N.: "Solving the Traveling Salesman Problem by an Efficient Hybrid Metaheuristic Algorithm", Journal of Advances in Computer Research, August 2012 , Volume 3, Number 3, pag. 75-84.
[2] Croes, G. A.: "A method for solving traveling salesman problems", Operations Res. 6 (1958) , pag. 791-812.
[3] Hojjat, E., Shahriar, L.: "Graph Colouring Problem Based on Discrete Imperialist Competitive Algorithm", CoRR, 2013.
[4] *** http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/
[5] *** http://www.math.uwaterloo.ca/tsp/vlsi/
[6] *** http://www.tsp.gatech.edu/apps/index.html
[7] Matai, R., Singh, Surya, Lal Mittal, M.: Traveling Salesman Problem: An Overview of Applications, Formulations and Solution Approaches, http://cdn.intechopen.com/pdfs-wm/12736.pdf (2010).
[8] Shirin, N., Hodais, S., Majid, V. J.: "A Binary Model on the Basis of Imperialist Competitive Algorithm in Order to Solve the Problem of Knapsack 1-0", International Conference on System Engineering and Modeling (ICSEM 2012), IPCSIT vol. 34 (2012), Singapore.
[9] Ciurea S., Trifa V.: "Imperialist Competitive Algorithm with Variable Parameters for the Optimization of a Fuzzy Controller", International Conference on System Theory, Control and Computing ICSTCC, Sinaia, 2014.
[10] Gargari Atashpaz, E., Caro, L.: "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition", IEEE Congress on Evolutionary Computation, CEC, 2007.

STELIAN CIUREA
"Lucian Blaga" University of Sibiu
Faculty of Engineering, Department of Computer and Electrical Engineering
E. Cioran Str, No. 4, Sibiu-550025, ROMANIA,
E-mail: stelian.ciurea@ulbsibiu.ro