HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

IT3160E

SEMESTER 20221

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

PROJECT REPORT

CLASS CODE: 136462

OUR GROUP INCLUDES FOUR MEMBERS:

Ta Ngoc Minh	20214918	minh.tn214918@sis.hust.edu.vn
Le Ngoc Binh	20214878	binh.ln214878@sis.hust.edu.vn
Ngo Viet Anh	20214875	anh.nv214875@sis.hust.edu.vn
Doan Ngoc Cuong	20210141	cuong.dn210141@sis.hust.edu.vn

SUPERVISOR: Professor Nguyen Nhat Quang, PhD.

Hanoi - 2022

Logistics and Maritime Transportation: *Natural-inspired Approach* Towards Multi-factorial Evolutionary Algorithms

[*]Project Report of the course IT3160E - Introduction to Artificial Intelligence

Ta Ngoc Minh[*]
SID No. 20214918
Email: minh.tn214918[†]
HUST-SoICT

Le Ngoc Binh
SID No. 20214878
Email: binh.ln214878[†]
HUST-SoICT

Ngo Viet Anh
SID No. 20214875
Email: anh.nv214875[†]
HUST-SoICT

Doan Ngoc Cuong
SID No. 20210141
Email: cuong.dn210141[†]
HUST-SoICT

***Abstract*—Logistics, especially marine transportation has made great strides over the decades. Along with that, the applications of technology, such as AI in this industry helps many logistics companies save lots of money and time by optimizing total cost of transportation. Moreover, the optimization for logistics has become a classic problem for optimization and operation research, since it requires lots of agents and the designed algorithms have to be multitaskable. This report is a part of our work in the class 136462, introduces about optimizing marine transportation by natural-approach algorithms, which is Multi-factorial Evolutionary Algorithm (MFEA) to minimizing both routes and number of used containers.**

***Index Terms*—Machine Learning, Logistics, Operation Research, Evolutionary Multitasking, Combinatorial Optimization.**

I. Setting up problem

The real-life problem that we take into consideration is in maritime transportation for such a big logistics company (eg., FedEx, USP, DHL). Let's imagine that we have to send many packages to several cities domestically and internationally. Each city has one package and we have a big ship to transport all of them. The remarkable thing is that when placing the packages on the ship, we must put them into containers instead of locating them directly on the ship.

For such a big number of cities and a big number of packages, for economic purposes, we must minimize the number of containers we have used and find the shortest way to reach each city only once during the transporting process.

In algorithmic expression, we need to solve two problems at the same time, which are TSP (Traveling Salesman Problem) and BPP (Bin-Packing Problem). Particularly, we have:

- **State:** Any route of the ship and any numbers of used containers.
- **Action:** Rearrange the packages from this container to another, and change the route of the ship.
- **Goal:** Find the shortest route and the least number of used containers.
- **Path cost:** The length of the route and the number of containers.

II. Our early approach

While considering the way of solving the two problems, we realize that: to find the best way, we need to have an initial way of arrangement and permute it to have a better solution. From that, we decide to use Evolutionary Algorithms (EAs), which are optimization meta-heuristics which work on Darwinian principles of *Natural Selection or Survival of the Fittest* [1]. The algorithm starts with a population of individuals and simulates sexual reproduction and mutation in order to create a generation of children (offspring). The process is repeated in an effort to maintain genetic material that makes a person more adapted to a particular environment while removing that which weakens it. Here, the word "environment" is used as a metaphor to refer to the setting of the objective function that is being optimized.

EAs nowadays are successfully applied to solve various optimization problems, especially in operation research. These problems can generally classified into 3 groups, 2 of them are: a), single-objective optimization (SOO) [2] where every point in the search space maps to a scalar objective value, or b) multi-objective optimization (MOO) [3], where every point in the search space maps to a vector-valued objective function.

Here, we use the third category of problems, which is *multi-factorial optimization* (MFO), towards applying multi-factorial evolutionary algorithm to solve maritime transport problem. To have a simple and optimal solution, we separate the problem into two problems that we have to multitask them, Traveling Salesman Problem - to find shortest route between cities and Bin Packing Problem - to find the optimal number of used containers.

III. Setting up Input data and Solution

A. Traveling Salesman Problem

The Traveling Salesman Problem (also Traveling Salesperson Problem - TSP) is given by setting up the question: "What

[†]@sis.hust.edu.vn ; [*]Team leader.

is the shortest path that visits each city precisely once and returns to the starting city, given a list of cities and the distances between each pair of them?" It is an *NP-hard* problem in combinatorial optimization, important in theoretical computer science and operations research [4].

Here, for sake of time, we only consider three packages of cities, which are *berlin52.tsp* - 52 locations in Berlin, *eil101.tsp* - 101-city problems by Christofides and Eilon, and *gr229.tsp* - Asia/Australia Sub-problem of 666-city TSP. All of the given cities are encoded in EUC-2D, which implies that we use Cartesian coordinates to encode them and calculate the distance using Euclidean distance rounded to the nearest whole number:

$$d = \left\lceil \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \right\rfloor$$

B. Bin Packing Problem

The bin packing issue is an optimization problem in which a finite number of bins or containers, each with a fixed set capacity, must be filled with goods of various size in a way that minimizes the number of bins used. The problem is computationally *NP-hard*, and the accompanying judgment problem-determining whether things fit into a given number of bins - is *NP-complete*.

There are three versions of rectangle bin packing problems have been discussed: the first version is to minimize the packing area, the second version is to minimize the height of a strip packing, and the final version is to minimize the number of bins used to pack the given items [5]. In this project, we consider the third version with the taken data from Florida State University to complete the algorithms and use bigger auto-generated test data with the dimension of 52, 101 and 229 respectively.

C. Multi-factorial Optimization

It is essential to first understand how constitutive MFO tasks relate to one another. It is important to note that the idea of evolutionary multitasking places no rigid restrictions on the inter-task interaction. The tasks of interest may be interrelated parts of a big multi-component problem or they may be independent activities.

However, for the sake of this problem, we'll focus on the first scenario, or where there is no prior knowledge of any inter-task connections. To put it another way, we take into account multitasking across optimization issues that are typically viewed separately (as distinct tasks). Therefore, finding the best trade-offs between the constitutive objective functions is not the goal of MFO. Instead, by utilizing the latent parallelism of population-based search, the objective is to fully and concurrently optimize each task.

With two selected small problems, which are minimization problems. The j^{th} task, denoted T_j, is considered to have the objective function is defined as $f_j : X_j \rightarrow \mathbb{R}$ where X_j is the search space. Hence, when we use an evolutionary multitasking paradigm of MFO, we try to find $\{x_1, x_2\} = \arg\min \{f_1(x), f_2(x)\}$ by population-based search [6].

We define a unified search space (USS) X encompassing X_1 and X_2 and each individual p_i in the population P can be decoded into task-specific solution. We denote the decoded form of p_i is $\{x_1^i, x_2^i\}$, where $x_1^i \in X_1$ and $x_2^i \in X_2$.

In MFO, taking advantage of genetic algorithms and many other natural-inspired algorithms, we implement the multitask-ability by adding five new definitions [6]:

Definition 1 (Factorial cost): For a given task T_j, the factorial cost Ψ_j^i of individual p_i is given by $\Psi_j^i = \lambda \delta_j^i + f_j^i$; where λ is a large penalizing multiplier, f_j^i and δ_j^i are the objective value and the total constraint violation, of p_i with respect to T_j. If p_i is feasible with respect to T_j (zero constraint violation), we have $\Psi_j^i = f_j^i$.

Definition 2 (Factorial rank): The factorial rank r_j^i of pi on task T_j is simply the index of p_i in the list of population members sorted in ascending order with respect to Ψ_j.

Definition 3 (Scalar fitness): The list of factorial ranks $\{r_1^i, r_2^i\}$ of an individual p_i is reduced to a scalar fitness φ_i based on its best rank over all tasks; i.e., $\varphi_i = \frac{1}{\min\limits_{j \in \{1,2\}} \{r_j^i\}}$.

Definition 4 (Skill factor): The skill factor τ_i of p_i is the one task, amongst all other tasks in MFO, on which the individual is most effective, i.e., $\tau_i = \arg\min \{r_j^i\}$.

Definition 5 (Multi-factorial optimality): An individual p^*, with a list of objective values $\{f_1^*, f_2^*\}$, is considered *optimum* in multifactorial sense if $\exists j \in \{1, 2\}$ such that $f_j^* \leq f_j(x_j)$, for all feasible $x_j^* \leq f_j(x_j)$.

IV. The Multi-Factorial Evolutionary Algorithm

The MFEA is inspired by the bio-cultural model of multi-factorial inheritance. It originates in the realm of memetic computation, as the algorithm's work is based on the transmission of biological and cultural components (genes and memes) from parents to offspring [7], [8].

Algorithm 1. Basic structure (modified for this problem).

1. Generate an initial population of individuals and store it in *current-pop* (P).
2. Evaluate every individual with respect to every task in the multitasking environment.
3. Compute the skill factor (τ) of each individual.
4. **while** (stopping condition is not reach) **do**:
   i. Apply genetic operators on current-pop to generate an offspring-pop (C). Refer to **Algorithm 2**.
   ii. **if** the evolution between k generations is negligible **then** regenerate the population.
   iii. Evaluate the individuals in offspring-pop for selected optimization tasks only (see **Algorithm 3**).
   iv. Concatenate *offspring-pop* and *current-pop* to form an *intermediate-pop* ($P \cup C$).
   v. Update the scalar fitness (φ) and skill factor (τ) of every individual in *intermediate-pop*.
   vi. Select the fittest individuals from *intermediate-pop* to form the next *current-pop* (P).
5. **end while.**

A. Population initialization

If you run two optimization tasks at the same time, let j^{th} tasks have dimensions D_j. So define a USS whose dimension ($D_{\text{multitask}}$) is equal to $\max_j\{D_j\}$. Thus, during the population initialization step, each individual has $D_{\text{multitask}}$ random variables (each within a specified range $[0, 1]$) in the vector is provided [6]. This vector forms the chromosome (complete genetic material) of the individual. The i^{th} dimensions of the USS are represented by the random keys y_i and the fixed regions represent the box constraints of the uniform space. While dealing with the T_j tasks, we just look at the first D_j random keys on the chromosome.

Here, after we initialize the population of 100 individuals, we decode the individual by sorting them in ascending order and get their indexes. From that, the route of TSP problem is the sorted list of the cities indexes and we put objects into containers using this order.

Moreover, to avoid localization issue (which will be discussed in depth in Section V-B), we determine a parameter k such that, if the evolution is negligible after k consecutive generations, the population is regenerated but the best previous solution is stored for the sake of comparison.

B. Genetic mechanisms

Canonical EA uses a pair of genetic operators, namely *crossover* and *mutation* [9]. These are similar to the origin of their biological names. A key feature of MFEA is that two randomly selected candidate parents of her must meet certain conditions in order to intersect. The principle followed is that of non-random mating or assortative mating, stating that individuals prefer to mate with individuals belonging to the same cultural background [10].

In the MFEA, the skill factor (τ) is considered a mathematical representation of an individual's cultural bias. Therefore, two randomly selected parent candidates are free to intersect if they have the same ability coefficient. Conversely, if the ability coefficients are different, crossover occurs or mutation occurs only according to a given random mating probability (*rmp*). Instructions for generating descendants according to these rules are given in Algorithm 2 [6].

Algorithm 2. Assortative mating.

Consider two parent candidates p_a and p_b randomly selected from *current-pop*.

1. Generate a random number *rand* between 0 and 1.
2. **if** ($\tau_a == \tau_b$) **or** ($rand < rmp$) **then**
   i. Parents p_a and p_b crossover to give two offspring individuals c_a and c_b.
3. **else**
   i. Parent p_a is mutated slightly to give an offspring c_a.
   ii. Parent p_b is mutated slightly to give an offspring c_b.
4. **end if.**

The parameter *rmp* is used to balance exploitation and exploration of the search space. A value of *rmp* close to 0 implies that only culturally alike individuals are allowed to crossover, while a value close to 1 permits completely random mating. Here, we choose $rmp = 0.3$ for enhanced exploration of the entire search space.

C. Selective evaluation

While evaluating an individual for task T_j, the first step is to decode its random-keys into a meaningful input for that task. For the case of discrete optimization, the chromosome decoding scheme is usually problem dependent [6].

Algorithm 3. Vertical cultural transmission via imitation.

An offspring c will either have two parents (p_a and p_b) or a single parent (p_a or p_b) - see **Algorithm 2**.

1. **if** (c has 2 parents) **then**
   i. Generate a random number *rand* between 0 and 1.
   ii. **if** $(rand < 0.5)$ **then**
   c imitates $p_a \rightarrow$ The offspring is evaluated only for task τ_a (the skill factor of p_a).
   iii. **else**
   c imitates $p_b \rightarrow$ The offspring is evaluated only for task τ_b (the skill factor of p_b).
   iv. **end if**
2. **else**
   c imitates its single parent $\rightarrow$ The offspring is evaluated only for that task which is its parent's skill factor.
3. **end if**
4. Factorial costs of c with respect to all unevaluated tasks are artificially set to ∞ (a very large number).

It is worth noting that incorporating cultural effects in a prescribed manner significantly reduces the total number of functional assessments required. In fact, in the K-factor problem, the job rating is almost K times lower than if one person were rated for all tasks [6].

D. Selection Operation

As shown in Algorithm 1, the MFEA follows an elitist strategy which ensures that the best individuals survive through the generations.

In order to identify the best individuals, we employ the technique presented in Section II for comparing population members while multitasking.

In particular, for sorting and selecting, we compute the fitness of each individual by the following algorithm. Here, for easy setup and fast calculation, we use QuickSort Algorithms for sorting.

For TSP problem, the fitness is calculated by formula $\frac{1}{\text{route}}$ where *route* is the total distance move from the first city to the last and return first city.

Algorithm 4. TSP fitness calculation.

Consider individual ind has the dimension dim.

1. Sorting ind in ascending order and return their indexes.
2. **for** $i = 0$ **to** $(dim - 1)$ **do**
   i. $route = route + distance(ind_i, ind_{i+1})$.
3. **end for.**

For BPP problem, we can easily consider that the fitness is calculated by $\frac{1}{\text{total bin}}$, and we put the objects to the bin with the order from first element to last element of each permutation of these objects.

Algorithm 5. BPP fitness calculation.

Consider individual ind has the dimension dim.

1. Sorting ind in ascending order and return their indexes.
2. **for** $i = 0$ **to** dim **do**
   i. **if** $current + weight(ind_i) > capacity$
   $total = total + 1$ and $current = weight(ind_i)$.
   ii. **else**
   $current = current + weight(ind_i)$.
3. **end for.**
4. **if** $current > capacity$ *// handle last object*
   $total = total + 1$.
5. **end if.**

V. Implementation

A. Implementation and Github repository

For further information about our works and submission, please visit our Github repository ngocminhta/mfeaJava. The implementation in this project is done in Java for the sake of executing time.

We set up for MFEA to loop for 1000 generations, and to avoid the localization during the optimization process, we use a variable $changeBest$ to regenerate the population (but still keep the current best solution) if the results do not change in the last 100 generations.

To illustrate the way of decoding, we assume we have an individual with 8 dimensions, optimizing for 8-city TSP problem and 4-object BPP problem.

0.71	0.26	0.53	0.11	0.05	0.96	0.47	0.78

We sort them in ascending order and return their indexes (instead of generating new list of sorted variables), we have:

6	3	5	2	1	8	4	7

This is the solution for TSP problem (since TSP have total 8 cities). However, since BPP only has 4 objects, we get the order of the objects by eliminating the number greater than 4.

3	2	1	4

If the bin has capacity of 100 and 4 objects has weight of 60, 70, 15, 7 respectively, we have the total number of bin is 2, which are $\{15, 70\}$ and $\{60, 7\}$.

B. Issues and difficulties

During the implementation process, we have to find out the way of encoding, decoding, evaluating, etc. However, two main and biggest difficulties that we get into trouble is localization and handling the differences between length of two problems.

Looking at the localization problem, it is so hard for us to calculate the number n such that the for-loop will not be localized after n generations. Besides, if we denote a fixed parameter ε such that iteration condition is

$$|best_{\text{current}} - best_{\text{previous}}| < \varepsilon$$

This takes us into consideration that it is so hard to compute variable ε without any loss of generality.

Hence, we come into a solution is that we save the best solution as $best$, and after k times of loop, if the variations are negligible, we regenerate a new population to eliminate the localization. Specifically, while implementing this problem, after trying many variables of parameter k, we initialize 2000 generations with $k = 200$.

The second problem is handling the differences between the length of the two problems. As we have proposed above, the dimensions of TSP and BPP are the same (each city has one package). However, in real-life, each city can have many packages, so we have to handle the differences between their lengths.

The recommended solution is that: from an individual of length d, we sort them by their indexes in ascending order (without any permutation between them), then eliminate all the elements whose value (index) is greater than the task dimension.

The route decoded for the TSP problem will be the route from the first element to the final and return to the first one. Similarly, we put the objects corresponding to the order of their indexes into the bin, then calculate the total bin needed.

After having solved all the issues, we consider enhancing the practicability of MFEA for this problem. However, while implementing to solve multiple TSPs and BPPs at the same time, the dimensions of decoded solutions just keep the biggest one for each type of task. For example, given two tasks of BPP have the dimensions of 52 and 101 respectively, the solutions' dimension of BPP is 101 for both of them. To solve that, it is easy for us to create a new list to store the dimension of each task respectively. From that, in the decoding process, we recall the dimension of the tasks whose order of the individual's skill factor to get the decoded solution.

VI. Summaries and Improvements

A. Results and Summaries

After running several times and improving the algorithm, we get the final result for optimizing tasks and have some conclusions. For readers easier in understanding, we denote the fitness is the total route of TSP and total bin of BPP.

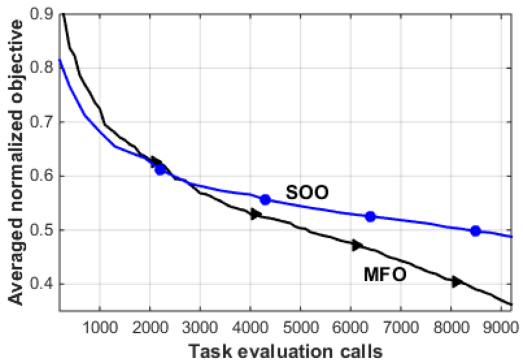
For the main problems, we use the TSP and BPP with the same dimension and get the final result in the table below

	Task	TSP route	BPP total bin
1	*berlin52.bpp52_1*	20,129.0	34
2	*berlin52.bpp52_2*	19,682.0	34
3	*berlin52.bpp52_3*	19,550.0	34
4	*eil101.bpp101_1*	2,776.0	56
5	*eil101.bpp101_2*	2,778.0	56
6	*eil101.bpp101_3*	2,780.0	56
7	*gr229.bpp229_1*	1,141,927.0	139
8	*gr229.bpp229_2*	1,156,161.0	139
9	*gr229.bpp229_3*	1,167,953.0	139

To make sure it has many applications in other problems, we optimize TSP and BPP with different dimensions and get the results in the table

	Task	TSP route	BPP total bin
1	*berlin52.bpp101*	19,437.0	56
2	*berlin52.bpp229*	19,193.0	139
3	*eil101.bpp52*	2,778.0	34
4	*eil101.bpp229*	2,750.0	139
5	*gr229.bpp52*	1,163,050.0	34
6	*gr229.bpp101*	1,163,820.0	56

Since MFEA is much complex and requires more infrastructure for computation than traditional genetic algorithms (GAs), this implies that whether we need to use MFEA to multitask many problems simultaneously, or we can implement each task as a GA problem. To answer for this problem, we have made a comparison between using single-objective optimization and multi-objective optimization. In comparison with solving each task by genetic algorithms [11] (here we choose TSP with 101 cities), we get the generation-fitness graph of two algorithms.



The graph implies that, from transferable knowledge between two tasks, the initial point of multitasking is higher (worse solution) but it converges much faster and get more optimal solution than solving each task independently.

In conclusion, the basic optimization for maritime transport and logistics has been easily solved using MFEA.

B. Proposal of Improvements

Despite the fact that the problem can be easily solved using the presented algorithm, the real-life problem is much more complex than that, so, we are continuing research about multitasking and especially transfer learning in order to make our research more applicable.

Besides, taking advantage of multitaskability, we are trying to implement the MFEA to solve the VRP problem (Vehicle Routing Problem), which is a NP-hard combinatorial optimization problem which asks *"What is the optimal route that a fleet of vehicles should take to deliver to a particular group of customers?"*. For this particular problem, we propose an early approach solution is that: If we handle all the given cities and cluster them into many groups. After that, we apply the problem for each cluster and combine them to get a solution of VRP.

ACKNOWLEDGMENTS

This work is supported and supervised by professor Quang N. Nguyen under the course of Introduction to Artificial Intelligence. Also, our research and project is partially supported by other professors and students at School of Information and Communication Technology, Hanoi University of Science and Technology.

REFERENCES

[1] T. Back, U. Hammel and H. . -P. Schwefel, "Evolutionary computation: comments on the history and current state," in IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 3-17, April 1997, doi: 10.1109/4235.585888.

[2] J. Yin, A. Zhu, Z. Zhu, Y. Yu and X. Ma, "Multifactorial Evolutionary Algorithm Enhanced with Cross-task Search Direction," 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 2244-2251, doi: 10.1109/CEC.2019.8789959.

[3] M. Asafuddoula, T. Ray and R. Sarker, "A Decomposition-Based Evolutionary Algorithm for Many Objective Optimization," in IEEE Transactions on Evolutionary Computation, vol. 19, no. 3, pp. 445-460, June 2015, doi: 10.1109/TEVC.2014.2339823.

[4] Braun, H. (1991). On solving travelling salesman problems by genetic algorithms. In: Schwefel, HP., Männer, R. (eds) Parallel Problem Solving from Nature. PPSN 1990. Lecture Notes in Computer Science, vol 496. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0029743.

[5] Shian-Miin Hwang, Cheng-Yan Kao and Jorng-Tzong Horng, "On solving rectangle bin packing problems using genetic algorithms," Proceedings of IEEE International Conference on Systems, Man and Cybernetics, 1994, pp. 1583-1590 vol.2, doi: 10.1109/ICSMC.1994.400073.

[6] A. Gupta, Y. -S. Ong and L. Feng, "Multifactorial Evolution: Toward Evolutionary Multitasking," in IEEE Transactions on Evolutionary Computation, vol. 20, no. 3, pp. 343-357, June 2016, doi: 10.1109/TEVC.2015.2458037.

[7] X. Chen, Y. -S. Ong, M. -H. Lim and K. C. Tan, "A Multi-Facet Survey on Memetic Computation," in IEEE Transactions on Evolutionary Computation, vol. 15, no. 5, pp. 591-607, Oct. 2011, doi: 10.1109/TEVC.2011.2132725.

[8] R. Mills, T. Jansen and R. A. Watson, "Transforming Evolutionary Search into Higher-Level Evolutionary Search by Capturing Problem Structure," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 5, pp. 628-642, Oct. 2014, doi: 10.1109/TEVC.2014.2347702.

[9] Goldberg, D.E., Holland, J.H. Genetic Algorithms and Machine Learning. Machine Learning 3, 95–99 (1988). https://doi.org/10.1023/A:1022602019183.

[10] Rice J, Cloninger CR, Reich T. Multifactorial inheritance with cultural transmission and assortative mating. I. Description and basic properties of the unitary models. Am J Hum Genet. 1978 Nov;30(6):618-43. PMID: 747189; PMCID: PMC1685878.

[11] Potvin, JY. Genetic algorithms for the traveling salesman problem. Ann Oper Res 63, 337–370 (1996). https://doi.org/10.1007/BF02125403.