Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(4): 2514–2540.
DOI: 10.3934/era.2024116
Received: 31 January 2024
Revised: 11 March 2024
Accepted: 22 March 2024
Published: 27 March 2024

*Research article*

# GPU-accelerated non-dominated sorting genetic algorithm III for maximizing protein production

**Donghyeon Kim and Jinsung Kim***

School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea

* **Correspondence:** Email: kimjsung@cau.ac.kr; Tel: +8228205554.

**Abstract:** Maximizing protein expression levels poses a major challenge in bioengineering. To increase protein expression levels, numerous factors, including codon bias, codon context bias, hidden stop codons, homologous recombination, suitable guanine-cytosine ratio, and hairpin loop structure, are crucial and quantified by six objective functions: CAI, CPB, HSC, HD, GC3, and SL. Optimizing these six objectives simultaneously constitutes a multi-objective optimization problem, aiming to identify the favorable Pareto solutions rather than a singular optimal solution. However, achieving satisfactory solutions requires numerous cycles and solutions, thus leading to a large number of functional evaluations. While there are frameworks for multi-objective optimization problems, they often lack efficient support for objective function computation in protein encoding. In this paper, we proposed a method to design a set of coding sequences (CDSs) based on non-dominated sorting genetic algorithm III (NSGA-III), accelerated using NVIDIA graphical processing units (GPUs). Experimental results indicated that our method is 15,454 times faster than the Pymoo framework and is evaluated using 100 solutions and 100 cycles. Since our GPU implementation facilitated the use of larger solutions and more cycles, we were able to design a superior set of CDSs by increasing solutions to 400 and cycles to 12,800. In addition, our NSGA-III-based method consistently surpassed the NSGA-II approach when the number of cycles exceeded 3200 by utilizing 100 solutions. Finally, we observed that a gradual reduction of the mutation probability as the number of cycles increased yielded better quality results than maintaining a fixed mutation probability.

**Keywords:** multi-objective optimization; bioengineering; NSGA-III; protein encoding; GPU computing; computational optimization

## 1. Introduction

Many real-world optimization problems in engineering and industry involve multiple objectives [1]. In multi-objective optimization problems, improving one objective often leads to the deterioration of another. This dynamic makes it impossible to find a single, optimal solution that simultaneously enhances all objectives [2]. As a result, finding the best trade-off solutions in multi-objective optimization problems is crucial for decision-making [3]. However, in many applied engineering optimization problems with multiple objectives, the cost of evaluating each objective is considerably high [4, 5]. Therefore, it is of paramount importance to address multi-objective optimization problems efficiently.

In bioengineering, encompassing the development of drugs, vaccines, and diagnostic tests, protein production plays a pivotal role [6–11]. For example, the S-glycoprotein is vital for antiviral vaccines against severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), but its production in mammalian cells is very expensive [12]. Another example is the production of rapid diagnostic test (RDT), which is often hampered by the high cost of producing and purifying specific proteins required for kit assembly in large quantities [13]. Therefore, efficient protein production is of great importance. To address this challenge, several studies [14–16] have explored methods involving the close integration of gene copies in organisms. This strategy theoretically enables protein production proportional to the number of inserted gene copies; however, it also poses a risk of homologous recombination owing to the close proximity of these copies. Homologous recombination, a phenomenon occurring when repetitive sequences are in close proximity, can consequently lead to a decrease in protein production [17, 18]. Therefore, it is essential to consider homologous recombination when designing a set of coding sequences (CDSs) to maximize the protein expression levels.

The hamming distance (HD) was introduced to measure the degree of similarity between CDSs, with higher HD values indicating greater differences in sequences [19]. In addition to homologous recombination, several other important considerations exist when designing the CDSs for high-protein expression levels. The codon usage bias refers to the different frequencies of codon usage in the host organism [20]. Codons with high-usage frequencies in the host organism generally contribute to efficient protein production [21–23]. Therefore, to maximize protein expression levels, it is crucial to construct CDSs using codons with high-usage frequencies in the host organism. To assess the codon usage bias of a CDS, the codon adaptation index (CAI) was introduced by [24] and has been extensively utilized. However, considering only codon usage bias presents limitations in detecting differences in usage frequencies among the observed codon pairs in the host organism [25]. The codon context bias refers to the different usage frequencies of adjacent codon-codon pairs in the host organism [26–28]. To address this, the concept of codon pair bias (CPB) was introduced [29] to evaluate the codon context bias of a CDS, thus demonstrating that the use of codon pairs with high-usage frequencies in the host organism maximizes protein expression levels. Therefore, it is important to consider both codon usage and context biases.

Hidden stop codons (HSCs) are stop codons within a CDS that can prematurely stop protein translation due to -1 frameshift or +1 frameshift caused by unstable ribosomal ribonucleic acids (rRNAs). If a -1 or +1 frameshift occurs, it may result in the production of a cytotoxic or nonfunctional protein that differs from the target protein and leads to a waste of resources [30]. However, the presence of numerous hidden stop codons in a CDS can rapidly terminate mistranslation caused by incorrect ribosomal frameshifts, thereby facilitating more efficient protein production [31, 32].

The hairpin loop, also known as a stem loop, is a secondary structure formed in single-stranded RNA when two regions oriented in opposite directions pair through complementary base pairing, thus forming a double helix. This structure serves various biological functions, such as providing recognition sites for protein binding and playing a role in enzymatic reactions as a substrate [33]. However, in messenger RNA (mRNA), the hairpin loop interferes with translation elongation by binding to the A-site of the ribosome, thereby hindering the binding of transfer RNA (tRNA) [34]. In other words, the presence of a hairpin loop within a CDS is undesirable for protein expression, as the probability of mistranslation increases with the length of the stem in the hairpin loop [35]. Therefore, to maximize protein expression levels, it is essential to consider the hairpin loop when designing the CDS.

Lastly, it is essential to consider the appropriate guanine-cytosine at the third position (GC3) in the design of the CDS [36]. Excessive or insufficient GC content can lead to poor protein synthesis. Consequently, in designing CDS for protein encoding, Gonzalez-Sanchez et al. [37] aimed to match the GC3 content with the host organism's GC3 ratio. Therefore, to enhance protein expression efficiency, our method also aims to align the CDS's GC3 content with that of the host organism.

In this study, we propose a method aimed at accelerating the optimization process for protein encoding based on the non-dominated sorting genetic algorithm III (NSGA-III). NSGA-III [38], an enhanced version of non-dominated sorting genetic algorithm II (NSGA-II) [39], is the metaheuristic algorithm designed for solving many-objective (four or more) optimization problems. The performance of NSGA-II and NSGA-III was analyzed in the domain of building performance optimization for the dormitory building type 1 (DBP-1) and dormitory building type 2 (DBT-2) problems, encompassing 10 and 9 design variables, respectively [40]. The experimental results demonstrated that NSGA-II yields a higher hypervolume than NSGA-III at a low number of cycles. However, with an increased number of cycles, NSGA-III consistently exhibited a higher hypervolume than NSGA-II for both problems, suggesting superior performance with more iterations. Additionally, for the sustainable-reconfigurable manufacturing system (S-RMS) problem involving 100 operations and 20 reconfigurable machines, increasing the number of cycles from 1000 to 2000 resulted in the outperformance of NSGA-III compared with that of NSGA-II [41]. Furthermore, the NSGA-III method achieved a success rate of over 93.08% in the trajectory planning problem for the Kiwi fruit harvesting manipulator and outperformed the NSGA-II method by approximately 21% [42]. However, it is not always guaranteed that NSGA-III will be superior to NSGA-II for all problems [43].

Consequently, we also conducted a comparative performance analysis of NSGA-II and NSGA-III for our specific protein encoding problem. In our experiments, when the number of cycles exceeded 3200, NSGA-III outperformed NSGA-II in all aspects of our protein encoding problem. While NSGA-III can improve the quality of the solutions by increasing the number of cycles and solutions, it also presents the challenge of increased running time [44]. Our GPU-based method enables the faster execution of a greater number of solutions and cycles. We compared the performance of the NSGA-III method as provided by the Pymoo framework [45] with that of our GPU-based approach. Pymoo, a Python-based framework, offers a variety of state-of-the-art single and multi-objective optimization algorithms. The experimental results indicated that our method provided solutions of similar quality to Pymoo, but with an average execution time that was 15,454 times faster. Additionally, leveraging our method's rapid execution time, we experimented by increasing the number of solutions to 400 and the number of cycles to 12,800. The experimental results indicated that our method yielded higher quality solutions by increasing the number of solutions and cycles, demonstrating superior hypervolume and a smaller minimum distance to the ideal point compared to the Pymoo framework. Furthermore, we conducted experiments to observe the effect of mutation probability on solutions by gradually reducing the mutation probability as the number of cycles increased. As a result, reducing the mutation probability by 10% every 100 cycles (starting from 40%) outperformed the fixed 20% mutation probability obtained using 100, 200, and 400 solutions with 12,800 cycles.

Our contributions to this research are as follows:

- Extension of the protein encoding problem to include two new optimization criteria: condon pair bias (CPB) and hidden stop codon (HSC).
- Introduction of a novel multi-objective formulation that incorporates six conflicting objectives.
- Design and implementation of an accelerated approach using NVIDIA GPUs, based on the NSGA-III, for solving the protein encoding problem.
- Development and comprehensive explanation of new problem-aware mutation methods.
- Comparative analysis with the Pymoo framework (employing NSGA-III) for multi-objective optimization and the NSGA-II approach applied to protein encoding.
- Evaluation of the proposed approach across various numbers of solutions and cycles, applied to nine real-world protein datasets, using hypervolume and minimum distance to ideal point metrics.

The structure of this paper is as follows. Section 2 presents a review of the related literature. In Section 3, the methodology of our approach is outlined, encompassing the mutation methods and elucidation of the six objective functions pertinent to the protein encoding problem. Section 4 engages in a comparative analysis of the outcomes derived from our method with those obtained using Pymoo, in addition to contrasting our method's application to the NSGA-II in addressing our specific problem. Conclusions drawn from this study are outlined in Section 5. Lastly, Section 6 sets forth the prospective directions for our future research endeavors and includes a discussion.

## 2. Related works

The methods proposed by Terai et al. [19], along with several others like multi-objective artificial bee colony (MOABC) [46], asynchronous parallel-multi-objective artificial bee colony (AP-MOABC) [47], multi-objective variable neighborhood search (MOVNS) [48], multi-objective shuffled frog leaping algorithm (MOSFLA) [49], many-objective mutation-based protein encoding (MaOMPE) [35], and multi-objective butterfly optimization algorithm (MOBOA) [37], have been developed to design a collection of CDSs. The primary goal of these methods is to enhance the expression levels of a protein that involves multiple objective functions. These studies approach the design of the CDSs set as a multi-objective optimization problem, aiming to identify a set of trade-off solutions by optimizing multiple objective functions simultaneously rather than seeking a single (best) solution. While these methods aim to create a set of CDSs for efficient protein production, they employ different objective functions and strategies to generate new solutions and select promising solutions in a single cycle.

### 2.1. Method proposed by Terai et al. and MOABC

Terai et al. proposed a method utilizing NSGA-II for designing a set of CDSs, optimizing three objective functions: CAI, HD, and the length of repeated or common substrings (LRCS). This method initializes $N$ solutions, which then serve as the $N$ original solutions for the first cycle; it then conducts successive cycles to find effective solutions. Following the initialization of $N$ solutions, the method processes crossover, mutation, and selection steps in each cycle. Both the crossover and mutation steps generate $N$ new solutions from the initial $N$. Subsequently, the selection step identifies the $N$

best solutions from a pool of 2$N$ solutions, utilizing non-dominated sorting and crowding distance sorting. These selected $N$ solutions subsequently become the original solutions for the ensuing cycle. Moreover, the MOABC method, employing the same objective functions as those proposed by Terai et al. [19], is based on the artificial bee colony (ABC) algorithm [50] for protein encoding. This method also initializes $N$ solutions, then progresses through cycles consisting of the employed bees step, onlooker bees step, and scout bees step. In both the employed bees step and the onlooker bees step, each step generates $N$ new solutions. The new $N$ solutions generated in the employed bees step are compared with the original $N$ solutions, and the superior ones replace the originals. However, in the onlooker bees step, the newly generated $N$ solutions are added to the existing pool, thus resulting in 2$N$ solutions in total. During the scout bees step, solutions that fail to yield promising results from the 2$N$ pool are replaced with solutions that are randomly generated and mutated. Following the scout bees step, non-dominated sorting and crowding distance sorting are applied to the 2$N$ solutions to select $N$ superior solutions for the subsequent cycle. MOABC requires the generation of new solutions per cycle that are more than two times those generated according to the method of Terai et al. [19]. Although MOABC demonstrated superior hypervolume and minimum distance to the ideal point compared with the method proposed by Terai et al. [19] within the same objective space and number of cycles, it necessitated more than two times the number of objective function computations per cycle. This increased computation is due to the need to evaluate objective functions for the newly generated solutions. Subsequent research by the same authors introduced AP-MOABC, an extension of MOABC with an OpenMP-based master-worker parallelization model, which effectively reduced the running time. In MOABC, the number of solutions replaced during the scout bees step varies depending on how many times each solution fails to generate a satisfactory solution. Consequently, some solutions may experience delays, thus leading to an imbalance in workload distribution within the GPU's thread block. Furthermore, as the replaced solution requires mutations proportional to the current cycle count, this imbalance exacerbates with an increase in cycles, rendering MOABC less suitable for GPU implementation.

## 2.2. *MOVNS and MOSFLA*

Both MOVNS and MOSFLA also employ CAI, HD, and LRCS objective functions. MOVNS is based on the variable neighborhood search algorithm [51, 52], whereas MOSFLA utilizes the shuffled frog leaping algorithms [53]. MOVNS employs the number of fitness evaluations as its termination condition, instead of relying on the number of cycles. When a new solution is generated, the evaluation count increases owing to the calculation of its objective function values. If this count reaches the maximum number of evaluations, MOVNS terminates. In this method, a new solution is created by mutating an original solution. If the newly generated solution dominates the original premutation solution, the latter is saved, and the new solution becomes the original solution for the next generation step. However, if the repeated generation of new solutions from an original solution fails to yield a dominating new solution over several iterations, the best among the saved solutions is selected as the new original solution. Once a solution has been utilized as an original solution for generating new solutions, it cannot be reused. When all saved solutions have been used in this manner, a new solution is randomly generated. Finally, when the maximum fitness evaluation is reached, the saved solutions are presented as the results. This method does not explore multiple new solutions concurrently because it generates a new solution one at a time. Given that exploring only one solution per step is highly inefficient for GPUs, MOVNS is not suitable for parallel execution.

In MOSFLA, the $N$ original solutions are divided into $m$ memeplexes, and then each memeplex generates the $N/m$ new solutions. Each memeplex may require up to three generations of solutions to produce a single new solution. This iterative process occurs because if the generated solution cannot dominate the worst solution in the memeplex, MOSFLA attempts to generate another solution. Initially, a new solution is generated based on the best solution within the memeplex. If this new solution, derived from the memeplex's best solution, does not dominate the memeplex's worst solution, another solution is generated based on the global best solution from the $N$ original solutions. If this attempt also fails to dominate the worst solution, a solution is randomly generated.

Furthermore, each time a new solution is generated within a memeplex, non-dominated sorting and crowding distance sorting are conducted to update the memeplex's local best and worst solutions. After all memeplexes have generated their respective new $N/m$ solutions, the entire set of $2N$ solutions undergoes non-dominated sorting and crowding distance sorting to select the $N$ best solutions. Although MOSFLA divides solutions into memeplexes, which theoretically could be processed in parallel, the need to update the local best and worst solutions within each memeplex after every solution generation means that only one solution is generated at a time within each memeplex. This limitation impedes the effective utilization of computing resources when using GPUs.

These two methods applied a novel strategy to enhance the HD objective function when generating a new solution from the original solution. Contrary to the method proposed by Terai et al. [19], and contrary to MOABC and AP-MOABC, which randomly mutate codons of a pair of CDSs with the minimum HD values, both MOVNS and MOSFLA permit codon changes only if they can potentially improve the HD objective function value. Furthermore, their experimental results indicated that this new approach, used to improve the HD objective function, could also simultaneously optimize the LRCS objective function.

### *2.3. MaOMPE and MOBOA*

MaOMPE, based on NSGA-III, aims to optimize four objective functions: CAI, HD, guanine–cytosine (GC) content, and stem length (SL). Although MaOMPE also applies the NSGA-III, similiar to our method, it does not consider the HSC and CPB objective functions. MaOMPE employed the Das-Dennis method [54] for setting reference points owing to its ability to generate well-distributed points on the hyperplane. MOBOA utilized the CAI, HD, and GC3 (GC content at the third nucleotide) objective functions, employing the butterfly optimization algorithm [55]. This method also generates $N$ new solutions from the $N$ original solutions and then selects the $N$ best solutions. In the process of performing cycles, this method introduced the TabooList and the BestList. The BestList retained the top six solutions from the $N$ original solutions, while the TabooList held solutions that were to be excluded from the BestList. To generate the $i$-th new solution, MOBOA can randomly select a solution from the BestList or use the $i$-th solution of $N$ original solutions. If a new solution, generated using a solution from the BestList, is dominated by its counterpart in the BestList, the latter solution is moved to the TabooList. The vacant slot of BestList is then filled with the next best solution from the $N$ original solutions. If all $N$ original solutions end up in the TabooList, meaning that no solutions can be included in the BestList, the TabooList is reset. After generating $N$ new solutions, non-dominated sorting and crowding distance sorting were performed to select the $N$ best solutions from the total of $2N$ solutions. Scanning the accumulated TabooList each time the BestList is updated creates overhead, and this method is constrained to generate only one solution at a time owing to the potential updating of the BestList. Consequently, these constraints inhibit the parallel generation of solutions and prevent efficient utilization of thread blocks when running MOBOA on GPUs.

Similar to MOVNS and MOSFLA, these two methods devised approaches to generate solutions with guaranteed improvements in the target objective functions. Therefore, when employing mutation methods to enhance specific objective functions, these methods ensured the generation of solutions with improved values for those targeted objectives.

## 3. Problem definition

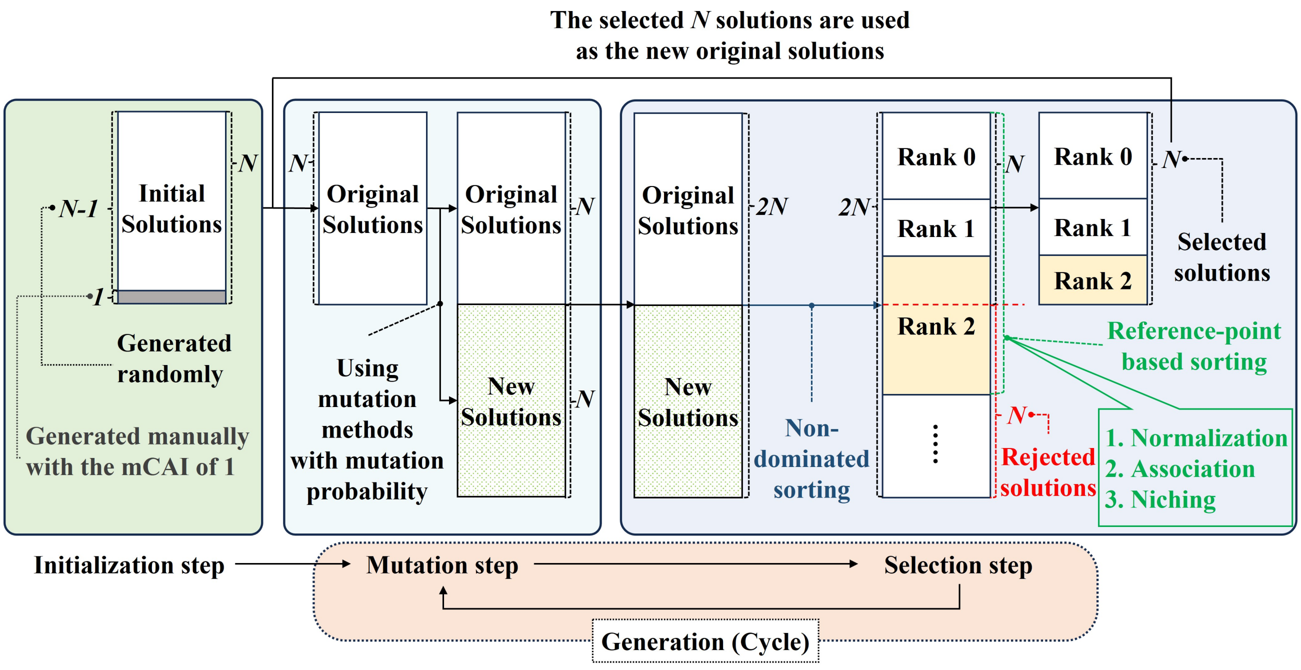


**Figure 1.** Schematic of our implementation procedure based on the non-dominated sorting genetic algorithm III (NSGA-III).

We propose a method to design a set of CDSs aimed at maximizing protein expression levels by taking into account six objective functions: CAI, CPB, HSC, HD, GC3, and SL. As our method incorporates six objective functions, we designed it based on the NSGA-III, an approach well-suited for solving many-objective (four or more objective) optimization problems. The procedure of our method comprises three key components: initialization, mutation, and the selection steps, as illustrated in Figure 1. The main difference between our method and the original NSGA-III paper is that when generating $N$ new solutions from $N$ original solutions, we only use the mutation method and do not perform crossover. This is because Terai et al. [19] demonstrated that crossover is not effective in generating good solutions in protein encoding problems.

In the initialization step, an initial set of $N$ solutions was generated. These $N$ solutions were then utilized as the original solutions in the first cycle. Among these initial $N$ solutions, $N-1$ solutions were randomly generated, while the remaining solution was composed of codons with the highest weight. Additionally, reference points were established in the initialization step to facilitate reference-point-based sorting in the selection step. To set these reference points, we adopted the Das-Dennis method [54] as utilized in the MaOMPE approach. These reference points were initialized on the unit simplex, determined by the partitioning number of the objective axes.

The number of reference points is determined by a combinatorial problem involving repetition, where the number of objectives corresponds to the total number of elements, and the number of partitions corresponds to the size of each subset. In Eq (1), $M$ represents the number of objective functions, $p$ denotes the number of partitions, and $H$ indicates the number of reference points. For example, if $M$ equals 2 and $p$ equals 2, then $H$ is 3. This implies that the reference points are $(1, 0)$, $(0.5, 0.5)$, and $(0, 1)$. Given that the number of reference points should not exceed $N$, and to ensure a number close to $N$ as suggested in the NSGA-III paper, we set the number of partitions such that the resulting number of reference points is less than $N$ but as close to it as possible.

$$H = C_p^{M+p-1} \tag{1}$$

The mutation step generates $N$ new solutions based on $N$ original solutions. To generate each new solution, one of the seven mutation methods is selected and utilized randomly. The *curand_uniform()* function is employed to generate a random number for each codon. If the randomly generated number of codons is less than the mutation probability, and if there is more than one synonymous codon available, then that codon may be mutated to another synonymous codon. In other words, a codon will not undergo mutation if its random number exceeds the mutation probability or if its corresponding amino acid, such as *methionine*, is encoded by using only a single synonymous codon. Additionally, except for the first mutation method, a codon will be not altered if the mutation does not lead to an improvement in the targeted objective function. The details of our mutation methods are elaborated in 3.8.

Following the mutation step, the selection step undertakes non-dominated sorting and reference-point-based sorting to select the $N$ best solutions from a pool of $2N$ solutions. Non-dominated sorting assigns rank values to solutions based on dominance tests and then arranges these solutions in ascending order based on their respective ranks. Solutions assigned to a rank of 0 are termed non-dominated solutions, thus indicating they are not dominated by any other solution. However, solutions with a rank of 1 may be dominated by those with a rank of 0. Therefore, among solutions, those with a lower rank value are preferred. However, as solutions within the same rank do not dominate each other, determining a superior solution among them is not feasible. To select preferred solutions within the same rank, reference-point-based sorting is employed. Reference-point-based sorting is implemented to discern better solutions within the same rank, particularly when the count exceeds $N$. This process consists of a normalization step, an association step, and niching steps. The normalization step involves the normalization of each objective function of the solutions to mitigate bias toward any specific objective function. For this step, we utilized the hyperplane normalization method [56], which involves the identification of the extreme points of each objective function and the creation of a hyperplane using these points. The normalization of the objective function values is then performed using the intercepts on each objective axis of the hyperplane. During the association step, the perpendicular distance between the line extending from the origin to the reference points and the normalized objective function values of the solution is calculated. Each solution is associated with the reference point that has the shortest perpendicular distance among the reference points. The niching step is the procedure of selecting solutions to be included in the $N$ best solutions. In this step, a solution is selected among those associated with the reference points, specifically choosing from those with the fewest associated solutions already included in the $N$ best solutions. After selecting the solution, the number of solutions included in the $N$ best solutions is updated in the reference point, thus continuously selecting the solutions from the reference points with the fewest solutions included in the $N$ best solutions. Once $N$ best solutions are selected, these $N$ solutions are used as the $N$ original solutions in the next cycle. Upon completion of all cycles, the $2N$ solutions are saved to a file.

## *3.1. Codon adaptation index (CAI)*

The CAI value of a CDS indicates the proportion of codons frequently used in the host organism that are present in the CDS. Codons that are frequently used in the host organism are assigned a higher weight. The weight of each codon is calculated by dividing its usage frequency in the host organism by the usage frequency of the most frequently used synonymous codon. Consequently, the weight assigned to the most frequently used synonymous codon is set to 1. The CAI value is determined by calculating the geometric mean of the weights of all codons within the CDS. In Eq (2), the term $weight(codon_{i,n})$ represents the weight of the $n$-th codon in the $i$-th CDS, and $N$ denotes the total number of codons within a CDS.

$$CAI(CDS_i) = \sqrt[N]{\prod_{n=1}^{N} weight(codon_{i,n})} \tag{2}$$

A higher CAI value typically correlates with improved protein expression levels. The minimum CAI (mCAI) value, calculated from the CDSs within a solution, is utilized as the objective function, as shown in Eq (3). Herein, $I$ represents the total number of CDSs in a solution. This approach is adopted because relying on the average CAI values of the CDSs in the solution may include a CDS that is unable to maximize protein expression levels, owing to its significantly lower CAI value compared with the average. Consequently, the objective function aims to increase the mCAI value within the solution. We utilized the same codon weight values as those used in previous studies [19, 35, 37, 46–49], which targeted the *S. cerevisiae* organism.

$$mCAI = \min_{1\le i\le I} CAI(CDS_i) \tag{3}$$

## *3.2. Codon pair bias (CPB)*

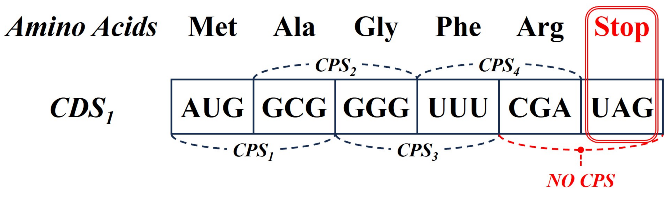


**Figure 2.** Codon pair score (CPS) of the codon pairs of the $CDS_1$.

The CPB value of a CDS indicates the number of codon pairs that are over-represented in the host organism and utilized in the CDS. Constructing a CDS with codon pairs that are over-represented in the host organism can contribute to the maximization of the protein expression levels; these CDSs typically exhibit high CPB values. Therefore, a higher CPB value in a CDS is preferred to optimize protein expression levels.

$$CPS = \ln\left(\frac{F(AB)}{\frac{F(A)\times F(B)}{F(X)\times F(Y)} \times F(XY)}\right) \tag{4}$$

To calculate the CPB value of a CDS, it is essential to determine the codon pair score (CPS) values for the codon pairs. The CPS value of each codon pair can be calculated using Eq (4). In this equation, $A$ and $B$ represent the codons, while $X$ and $Y$ denote the amino acids that are encoded by $A$ and $B$, respectively. Furthermore, $F(A)$ and $F(B)$ denote the frequencies of codons $A$ and $B$ in the host organism, while $F(X)$ and $F(Y)$ indicate the respective frequencies of amino acids $X$ and $Y$ in the host organism. Given that the three-stop codons do not encode any amino acid, CPS values can be determined for a total of 3721 codon pairs. A positive CPS value signifies that a codon pair is over-represented in the host organism, whereas a negative CPS value suggests under-representation. We utilized the CoCoPUTs database [57] to acquire the frequencies of the codons, codon pairs, amino acids, and amino acid pairs.

$$CPB(CDS_i) = \sum_{n=1}^{N-2} \frac{CPS(codon\_pair_{i,n})}{N-2} \tag{5}$$

As shown in Eq (5), the CPB value of a CDS is calculated as the arithmetic mean of the CPS values of the codon pairs within the CDS. $N$ represents the total number of codons in a CDS, while $N-2$ signifies the total number of codon pairs in the CDS excluding the stop codon. In Figure 2, $CDS_1$ comprises six codons, which results in a total of five codon pairs: AUGGCG, GCGGGG, GGGUUU, UUUCGA, and CGAUAG. Due to the inclusion of the stop codon UAG, the final codon pair CGAUAG is without a CPS value, unlike the preceding four codon pairs that each have their own CPS value. Consequently, the value of $N-2$ for $CDS_1$ is four.

$$mCPB = \min_{1\le i\le I} CPB(CDS_i) \tag{6}$$

Similar to mCAI, using the average of CPB values in a solution may obscure the presence of a CDS with a low CPB value, which is not conductive to maximizing protein expression levels. Therefore, we utilized the minimum CPB (mCPB) as the objective function to increase the mCPB value within the solution.

### 3.3. Hidden stop codon (HSC)

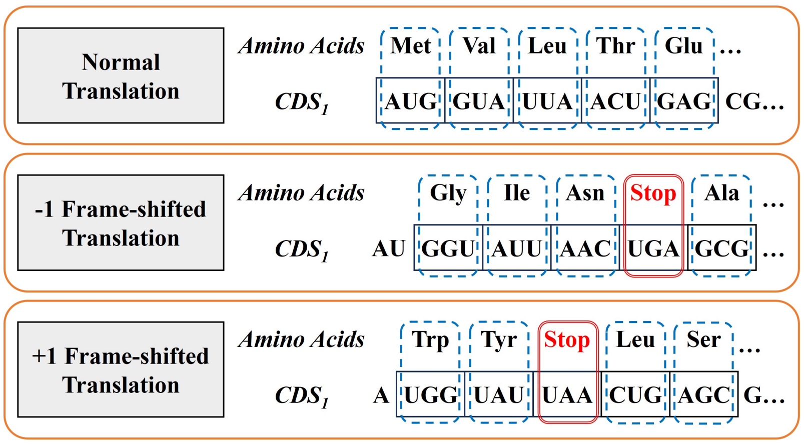

Normal Translation
Amino Acids
Met
Val
Leu
Thr
Glu
...
$CDS_1$
AUG GUA UUA ACU GAG CG...
-1 Frame-shifted Translation
Amino Acids
Gly
Ile
Asn
Stop
Ala
...
$CDS_1$
AU GGU AUU AAC UGA GCG ...
+1 Frame-shifted Translation
Amino Acids
Trp
Tyr
Stop
Leu
Ser
...
$CDS_1$
A UGG UAU UAA CUG AGC G...

**Figure 3.** Schematic showing the -1 or +1 frameshifted and normal translation of the $CDS_1$.

HSC is a metric that counts the number of stop codons capable of terminating protein translation in the events of a -1 or +1 frameshift within the CDS. Additionally, -1 or +1 frameshifts in the CDS can result in the production of a protein that differs from the intended target protein. Therefore, promptly terminating the frameshifted translation upon the occurrence of a -1 or +1 frameshift can potentially prevent wastage of energy and resources. This approach maximizes protein expression levels by avoiding unnecessary expenditure in the translation process.

$$HSC(CDS_i) = \sum_{n=1}^{N-1} hsc(codon\_pair_{i,n}) \tag{7}$$

Figure 3 illustrates the normal translation of $CDS_1$ and the scenarios wherein a -1 or +1 frameshift occurs. Each translation variant results in a different amino acid sequence: the normal translation yields a sequence starting with Met (*Methionine*)-Val (*Valine*), the -1 frameshifted translation changes this to Gly (*Glycine*)-Ile (*Isoleucine*), and the +1 frameshifted translation alters it to Trp (*Tryptophan*)-Tyr (*Tyrosine*). In the case of a -1 frameshift, translation prematurely terminates at the UGA stop codon, whereas a +1 frameshift leads to early termination at the UAA stop codon. The total number of hidden stop codons in the $i$-th CDS of the solution is calculated using Eq (7). The term $hsc(codon\_pair_{i,n})$ represents the count of hidden stop codons in the $n$-th codon pair of the $i$-th CDS.

$$mHSC = \min_{1\leq i\leq I} \frac{HSC(CDS_i)}{N-1} \tag{8}$$

Given that hidden stop codons prematurely terminate the translation process during a -1 or +1 frameshift, the objective is to increase the minimum count of hidden stop codons present in the solution. In other words, it is preferable to increase the value of the minimum HSC (mHSC) objective function to maximize protein expression levels. The mHSC value is calculated by normalizing the count of hidden stop codons in the CDS with the lowest number of such codons, then dividing this figure by the total number of codon pairs, which is represented as $N - 1$. This division by $N - 1$ is based on the fact that in the context of stop codon combination, a single codon pair can contain at most one hidden stop codon.

### *3.4. Hamming distance of CDS pair (HD)*

The HD value represents the degree of difference between the sequences of a pair of CDSs. Specifically, this value is calculated by comparing nucleotides at corresponding positions in the CDS pair.

$$HD(CDS_i, CDS_j) = \sum_{1 \le l \le L} \sigma(CDS_{i,l}, CDS_{j,l}) \tag{9}$$

As shown in Eq (9), $CDS_{i,l}$ refers to the $l$-th nucleotide in the $i$-th CDS of the solution, while $CDS_{j,l}$ denotes the $l$-th nucleotide in the $j$-th CDS of the solution. Furthermore, as HD measures the difference between two distinct CDSs, $i$ and $j$ cannot be the same. If $CDS_{i,l}$ and $CDS_{j,l}$ are equal, then the value of $\sigma(CDS_{i,l}, CDS_{j,l})$ is 0; otherwise, it is 1.

$$mHD = \min_{1 \le i < j \le I} \frac{HD(CDS_i, CDS_j)}{L} \tag{10}$$

Therefore, a higher HD value indicates a greater difference between the CDSs and a lower probability of homologous recombination. Given that homologous recombination can impair protein expression levels, and our objective is to maximize these levels, a higher HD value is preferred. Similar to the context of mCAI, mCPB, and mHSC, we used the minimum HD (mHD) as the objective function, as shown in Eq (10), aiming to increase the mHD value of the solution.

### *3.5. GC content at the third nucleotide (GC3)*

Our objective is to align the ratio of guanine (G) or cytosine (C) at the third position of codons in the CDS with the corresponding ratio observed in the host organism.

$$GC3\,(CDS_i) = \frac{\sum_{1 \le n \le N} \delta\,(CDS_{i,n_3})}{N} - GC3_{ideal} \tag{11}$$

$GC3\,(CDS_i)$ quantifies the difference in the G or C content ratio at the third position of codons in the $i$-th CDS relative to the G or C content ratio at the third position of codons as observed in the host organism, as shown in Eq (11). The value of $\delta(CDS_{i,n_3})$ is 1 if a G or C is present at the third position of the $n$-th codon in the $i$-th CDS of the solution; otherwise, it is 0. Herein, $N$ represents the total number of codons in a CDS. $GC3_{ideal}$ refers to the ratio of G or C content at the third position of codons as observed in the host organism. This ratio, obtained from the Kazusa Codon Usage Database [58], has a value of 38.10%.

$$MGC3 = \max_{1 \le i \le I} |GC3\,(CDS_i)| \tag{12}$$

As the goal is to minimize the divergence in G or C content of the CDSs relative to the host organism, the maximum GC3 value among the CDSs in the solution is employed as the objective function, contrasting with the use of mCAI, mCPB, mHSC, and mHD. In Eq (12), the maximum GC3 (MGC3) is calculated as the absolute value of the GC3 of each CDS. Therefore, as the G or C ratio at the third position of the CDS approaches the $GC3_{ideal}$ value, the MGC3 value converges toward 0. To optimize protein expression levels, we aim to minimize the MGC3 value.

### 3.6. Stem length (SL)

Stem length is a metric used to measure the length of the stem structure within a CDS. A hairpin loop, characterized by the formation of a double helix, occurs when two complementary regions within the same CDS base-pair to form a stem. The presence of a hairpin loop structure can impede proper translation, and a longer stem within the hairpin loop increases the probability of incorrect translation. Therefore, minimizing the length of the stem within the hairpin loop is crucial for maximizing protein expression levels.

$$max\_len\_SL = \frac{L-4}{2} \tag{13}$$

The variable *max_len_SL* represents the maximum stem length capable of forming a hairpin loop within the same CDS, as shown in Eq (13). The subtraction of the value of 4 accounts for the requirement of at least four nucleotides between two stems to form a hairpin loop, and *L* denotes the length of a CDS.

$$SL(CDS_i) = \max_{1 \le p \le q-(l+4) < q+l \le L \quad and \quad 1 \le l \le max\_len\_SL} \beta\left(S_{i,p,l}, S_{i,q,l}\right) \tag{14}$$

As shown in Eq (14), $SL(CDS_i)$ indicates the length of the longest stem in the $i$-th CDS of the solution, $p$ and $q$ are the starting indices of the nucleotides for two stems within the same CDS, and $l$ represents the stem length.

$$MSL = \max_{1 \le i \le I} \frac{SL(CDS_i)}{max\_len\_SL} \tag{15}$$

Similar to the MGC3 objective function, the maximum SL value among the CDSs in the solution was employed. The maximum SL (MSL) objective function was calculated by normalizing the longest stem length in the solution (by dividing it $max\_len\_SL$), as shown in Eq (15). Therefore, our objective was to minimize the value of MSL in the solution.

### 3.7. Our approach to implement our method on GPUs

Our implementation on GPUs consists of an initialization, a mutation, and a selection kernel, as shown in Algorithm 1. The initialization kernel executes only once at the beginning, generating the initial $N$ solutions. In our implementation, each thread block is responsible for creating one solution. Each thread generates the codons for its assigned solution, and the generated solution is stored in shared memory. This approach is adopted because a solution is read multiple times when calculating the objective function values, thus making it more efficient to store the solution in shared memory; this offers faster access than global memory. Additionally, the setting of reference points for reference-point-based sorting, used in the selection kernel, is performed in the initialization kernel. After the initialization kernel, the mutation kernel and selection kernel are executed iteratively. The mutation kernel employs mutation methods to generate $N$ new solutions from the $N$ original solutions and calculates the objective function values of these newly generated solutions.

In our implementation, each thread block is tasked with generating a solution, and as each thread block is responsible for generating a solution, the threads within a block compute the objective function values of the generated solution. Each thread within a thread block performs its parts of the computation for specific objective function values and stores intermediate values in the shared memory; these values can be accessed by all threads within the block. The divide-and-conquer strategy was then used to calculate the final objective function value using the intermediate values stored in shared memory. In our method, the divide-and-conquer strategy was exploited for all calculations of the six objective function values.

**Algorithm 1** Pseudocode of our method

```
1:  begin
2:      kernel initialization                                        ▷ Initialization step
3:          set reference points
4:          generate initial $N$ solutions
5:          calculate objective function values of initial $N$ solutions
6:      end kernel
7:      for cycle = 1 to max_cycles do
8:          kernel mutation                                          ▷ Mutation step
9:              generate $N$ new solutions from $N$ original solutions
10:             calculate objective function values of $N$ new solutions
11:         end kernel
12:         kernel selection                                         ▷ Selection step
13:             perform non-dominated sorting
14:             if last front exceeds $N$ then
15:                 perform reference-point-based sorting
16:             end if
17:         end kernel
18:     end for
19:     write final $2N$ solutions to the file
20: end
```

After the mutation kernel, a total of $2N$ solutions along with their objective function values were stored in global memory. This approach was adopted to enable all threads from each thread block to access all solutions when selecting the $N$ best solutions in the selection kernel. In the selection kernel, non-dominated sorting and reference-based sorting were performed. In non-dominated sorting, each thread handles one solution and determines its rank value by comparing it with other solutions. During the non-dominated sorting process, reference-point-based sorting is conducted to select solutions for inclusion among those exceeding $N$ within the same rank. In reference-point-based sorting, each thread normalizes the objective function values of a solution. After normalization, each thread calculates the perpendicular distances between its assigned solution and all reference points. Each solution is associated with the reference point with the shortest perpendicular distance. Each thread is then responsible for a reference point to select the $N$ best solutions. The solutions are randomly selected from the reference points that already have the fewest associated solutions selected for the $N$ best solutions. Once the $N$ best solutions are chosen, the selection step concludes, and these superior solutions are fed into the next mutation kernel as $N$ original solutions. After all cycles have concluded, all the $2N$ solutions are saved to a file.

### 3.8. Mutation methods

As illustrated in Figure 1, the mutation step generates new solutions from the original ones to create optimized solutions that have higher objective values. We employ six distinct mutation methods, each aimed at improving one of six objective functions, including one method to randomly mutate codons. Specifically, when generating a new solution, one of these seven mutation methods is randomly selected and applied. Additionally, if the randomly generated number for a codon is less than the specified mutation probability, the codon may be mutated to another synonymous codon of the same amino acid.

1) *Random:* Codons of all CDSs in a solution are randomly mutated to different synonymous codons.
2) *mCAI:* Codons within the CDS that exhibit the minimum CAI value in a solution undergo mutation. Each codon is randomly mutated to one of its synonymous codons that have a greater weight than the original codon.
3) *mCPB:* Codons within the CDS that exhibit the minimum CPB value in a solution undergo mutation. As mutating a codon impacts the cumulative CPS of the adjacent left and right codon pairs containing it, the codon is randomly mutated to one of the synonymous codons, resulting in a higher cumulative CPS for these pairs compared with the original CPS sum. To enable parallel mutation of codons by threads without dependency on neighboring codons, even-numbered cycles are used to mutate even-numbered codons, while odd-numbered cycles are designated for mutating odd-numbered codons. In this scenario, the mutation probability is doubled to compensate for half the codons eligible for mutation in the CDS.
4) *mHSC:* Codons within the CDS that have the minimum HSC value in a solution undergo mutation. Similar to CPB, mutating a codon impacts the combined HSC of the adjacent left and right codons. Therefore, only even-numbered codons are mutated during even cycles, while odd-numbered codons are mutated during odd cycles. The codon is randomly mutated to one of the synonymous codons that have a greater combined HSC value for the adjacent left and right codons than the original HSC sum. Furthermore, the probability of such mutations occurring is doubled.
5) *mHD:* Codons from one CDS (within a pair), which exhibits the minimum HD value in a solution, are subject to mutation. Among all synonymous codons, preference for mutation is given to the codon that can increase the minimum HD value of the solution. If the minimum HD value of the solution remains unchanged, and if there are synonymous codons capable of increasing the HD value of the CDS pair that originally had the minimum HD value, this codon is chosen for mutation. This approach is adopted because it could potentially decrease the HD value in conjunction with another CDS pair; by contrast, mutating the codon increases the HD value of the CDS pair with the original minimum HD value.
6) *MGC3:* Codons within the CDS that exhibit the maximum GC3 value in a solution undergo mutation. If the CDS's GC3 value exceeds the host organism's ideal GC3 value, the codon is randomly mutated to one of the synonymous codons, which would reduce the GC3 value. Conversely, if the GC3 value of the CDS falls below the ideal GC3 value of the host organism, the codon is randomly mutated to one of the synonymous codons, which would increase the GC3 value. With the exceptions of *methionine* and *tryptophan*, all amino acids can be encoded by synonymous codons that either include or lack G or C at the third position. Therefore, we dynamically adjusted the mutation probability, anticipating that the mutation of a single codon could reduce the discrepancy between the ideal and the current GC3 values by one. This dynamic mutation proba-

bility can be determined by the MGC3 value of the solution. This method was chosen to achieve efficient parallel mutation of codons across all threads.

7) *MSL:* The codons within one stem of the CDS, which exhibits the maximum SL value in a solution, are subject to mutation. Initially, the codon in the middle of the stem is randomly mutated to one of its synonymous codons. If this mutation results in a decrease in the SL value, the mutation process is terminated. Otherwise, the mutation occurs in the codon to the left side of the middle codon. If this does not decrease the SL value, mutation then occurs in the codon to the right part of the middle codon. This process of repeated mutations of the left and right codons progresses from the middle codon toward the ends of the stem and terminates immediately if there is a decrease in the SL value. The rationale for initiating mutations from the middle codon of the stem relies on the fact that a mutation of this codon has the potential to reduce the stems by up to half.

## 4. Experimental results

In our experiments, we used an AMD Ryzen 9 7950X, 16-core processor with 64 GB DDR5 RAM, and NVIDIA GeForce RTX 4090 (128 Ada SMs, 128 CUDA cores/SM, 24 GB global memory). The implementation was compiled using GCC 11.4.0 and CUDA 12.3 (driver version 545.23.08), running on the Ubuntu 22.04.3 LTS; Python (version 3.10.12) was also used.

**Table 1.** Protein instances used in the experiments.

| Code | Name | CDSs | Length (AA) | CDSs × Length |
|---|---|---|---|---|
| Q5VZP5 | DUS27_HUMAN | 2 | 1158 | 2316 |
| A4Y1B6 | FADB_SHEPC | 3 | 716 | 2148 |
| B3LS90 | OCA5_YEAS1 | 4 | 679 | 2716 |
| B4TWR7 | CAIT_SALSV | 5 | 505 | 2525 |
| Q91X51 | GORS1_MOUSE | 6 | 446 | 2676 |
| Q89BP2 | DAPE_BRADU | 7 | 388 | 2716 |
| A6L9J9 | TRPF_PARD8 | 8 | 221 | 1768 |
| Q88X33 | Y1415_LACPL | 9 | 114 | 1026 |
| B7KHU9 | PETG_CYAP7 | 10 | 38 | 380 |

**Table 2.** Nadir and ideal points for the calculation of quality indicators.

| Objective functions | Nadir value | Ideal value |
|---|---|---|
| mCAI | 0 | 1 |
| mCPB | -0.15 | 0.2 |
| mHSC | 0 | 0.4 |
| mHD | 0 | 0.5 |
| MGC3 | 0.6 | 0 |
| MSL | 1 | 0 |

Table 1 shows nine protein instances used in our experiments; these were selected based on their length of amino acid sequences and the number of CDSs to capture a range of complexities in protein encoding. These protein instances are the same as those utilized in MOABC [46], MOSFLA [49], MaOMPE [35], and MOBOA [37]. The amino acid sequences for each protein were obtained from the UniProt databases [59]. Additionally, Table 2 provides the ideal and nadir values, which are used to normalize the objective function values to a range from zero to one, thereby ensuring an unbiased evaluation of solution quality post-optimization. The normalized objective function value is calculated using Eq (16). In this context, a lower objective function value, indicative of a more favorable outcome, corresponds to a normalized value approaching zero.

$$x_{normalized} = \frac{x - x_{ideal}}{x_{nadir} - x_{ideal}} \tag{16}$$

The hypervolume and the minimum distance to the ideal point serve as pivotal quality indicators for evaluating the solutions in multi-objective optimization. Hypervolume, recognized as the most extensively adopted metric, assesses the quality of solutions by calculating the volume occupied by Pareto solutions in the objective space. In the context of the six-objective protein encoding problem addressed here, the hypervolume corresponds to the volume of the six-dimensional objective space encompassed by these Pareto solutions. A higher hypervolume value is indicative of superior multi-objective quality. Conversely, the minimum distance to the ideal point quantifies the proximity between the normalized objective function values of solutions and the utopian point, defined in this study as $(0, 0, 0, 0, 0, 0)$. This metric is instrumental in selecting an optimal solution among the Pareto solutions with a preference for a smaller value. These metrics usually provide a comprehensive assessment of solution quality in multi-objective optimization problems.

**Table 3.** Experimental results of our GPU-accelerated NSGA-III and Pymoo-based NSGA-III (100 solutions and 100 cycles).

| | Our NSGA-III | | | Pymoo (NSGA-III) | | |
|---|---|---|---|---|---|---|
| Protein | Hypervolume | Min. Distances | Seconds | Hypervolume | Min. Distances | Seconds |
| Q5VZP5 | 21.46% | 0.9245 | 1.8 | 21.44% | 0.9180 | 28,766.4 |
| A4Y1B6 | 14.05% | 0.9989 | 1.0 | 14.19% | 0.9816 | 15,782.8 |
| B3LS90 | 13.11% | 0.9940 | 1.1 | 13.16% | 1.0223 | 19,346.7 |
| B4TWR7 | 7.74% | 1.0983 | 0.8 | 7.72% | 1.0601 | 12,742.2 |
| Q91X51 | 7.94% | 1.0917 | 0.7 | 8.21% | 1.0793 | 11,511.0 |
| Q89BP2 | 8.54% | 1.0516 | 0.7 | 9.16% | 1.0373 | 10,132.1 |
| A6L9J9 | 6.82% | 1.0976 | 0.3 | 6.57% | 1.0715 | 3614.4 |
| Q88X33 | 7.76% | 1.0559 | 0.2 | 8.49% | 1.0133 | 1064.5 |
| B7KHU9 | 6.16% | 1.0978 | 0.1 | 6.64% | 1.0509 | 140.8 |
| Average | 10.40% | 1.0039 | 0.7 | 10.62% | 0.9955 | 11,455.7 |

Table 3 presents the experimental results comparing our GPU-accelerated NSGA-III method with the NSGA-III method provided by the Pymoo framework on CPUs. The Pymoo framework offers various single and multi-objective optimization algorithms based on Python. In other words, Pymoo's NSGA-III refers to the application of the NSGA-III method from Pymoo to our protein encoding problem. The experiments were performed with 100 solutions and 100 cycles. In addition, a mutation probability of 40% was employed as it yielded the best results compared with the mutation probabilities of 1.25%, 2.5%, 5%, 10%, 20%, 40%, and 50%. As shown in Table 3, both our NSGA-III and Pymoo's NSGA-III demonstrated similar hypervolume and minimum distance to the ideal point outcomes on average. However, as Table 3 illustrates, our NSGA-III executed approximately 15,454 times faster than Pymoo's NSGA-III. Consequently, our NSGA-III can deliver solutions of similar quality to those of Pymoo's NSGA-III but in a significantly reduced running time. Furthermore, the expedited execution time implies that our method can process a larger number of solutions and cycles, indicating its capability to accomplish more optimization within a shorter timeframe compared to Pymoo.

Figure 4 shows the comparative analysis hypervolume and minimum distance to the ideal point outcomes for NSGA-III (6-obj), NSGA-II (6-obj), and NSGA-II (3-obj) methods for the CAI, CPB, HSC, HD, GC3, and SL objective functions. NSGA-III (6-obj) represents our method, whereas NSGA-II (6-obj) denotes the application of NSGA-II to the same objective functions utilized in our method. The primary distinction between these two methods lies in the selection step: NSGA-III (6-obj) employs non-dominated sorting and reference-based sorting, whereas NSGA-II (6-obj) utilizes non-dominated sorting and crowding distance sorting. The NSGA-II (3-obj) method, as proposed in [60], is a GPU-accelerated approach based on Terai's method, considering the CAI, HD, and LRCS objective functions. NSGA-II (3-obj) provides solutions that are very similar to Terai's but run much faster. Given that NSGA-II (3-obj) employed a mutation probability of 5%, both NSGA-III (6-obj) and NSGA-II (6-obj) also adopted the same mutation probability. In the experiments, the number of solutions was set to 100, and the cycles were conducted at the intervals of 100, 200, 400, 800, 1600, 3200, and 6400. In all cases, NSGA-II (3-obj) exhibited a poorer hypervolume and a greater minimum distance to the ideal point compared with both NSGA-III (6-obj) and the NSGA-II (6-obj).

Furthermore, increasing the number of cycles did not result in a significant improvement in the quality of solutions in NSGA-II (3-obj). In other words, even with an increased number of cycles, the NSGA-II (3-obj) method, akin to Terai's method, failed to yield satisfactory solutions in the CAI, CPB, HSC, HD, GC3, and SL objective space. When comparing NSGA-III (6-obj) and NSGA-II (6-obj), NSGA-II (6-obj) yielded better hypervolume outcomes with fewer cycles. However, NSGA-III (6-obj) consistently outperformed NSGA-II (6-obj) in hypervolume in all the cycles for both the Q91X51 and Q89BP2 proteins. In addition, from cycle 3200 onward, the NSGA-III (6-obj) showed a consistently better hypervolume for all the proteins than NSGA-II (6-obj). Finally, NSGA-III (6-obj) demonstrated superior results in terms of the minimum distance to the ideal point compared with NSGA-II (6-obj) across all proteins and cycles.

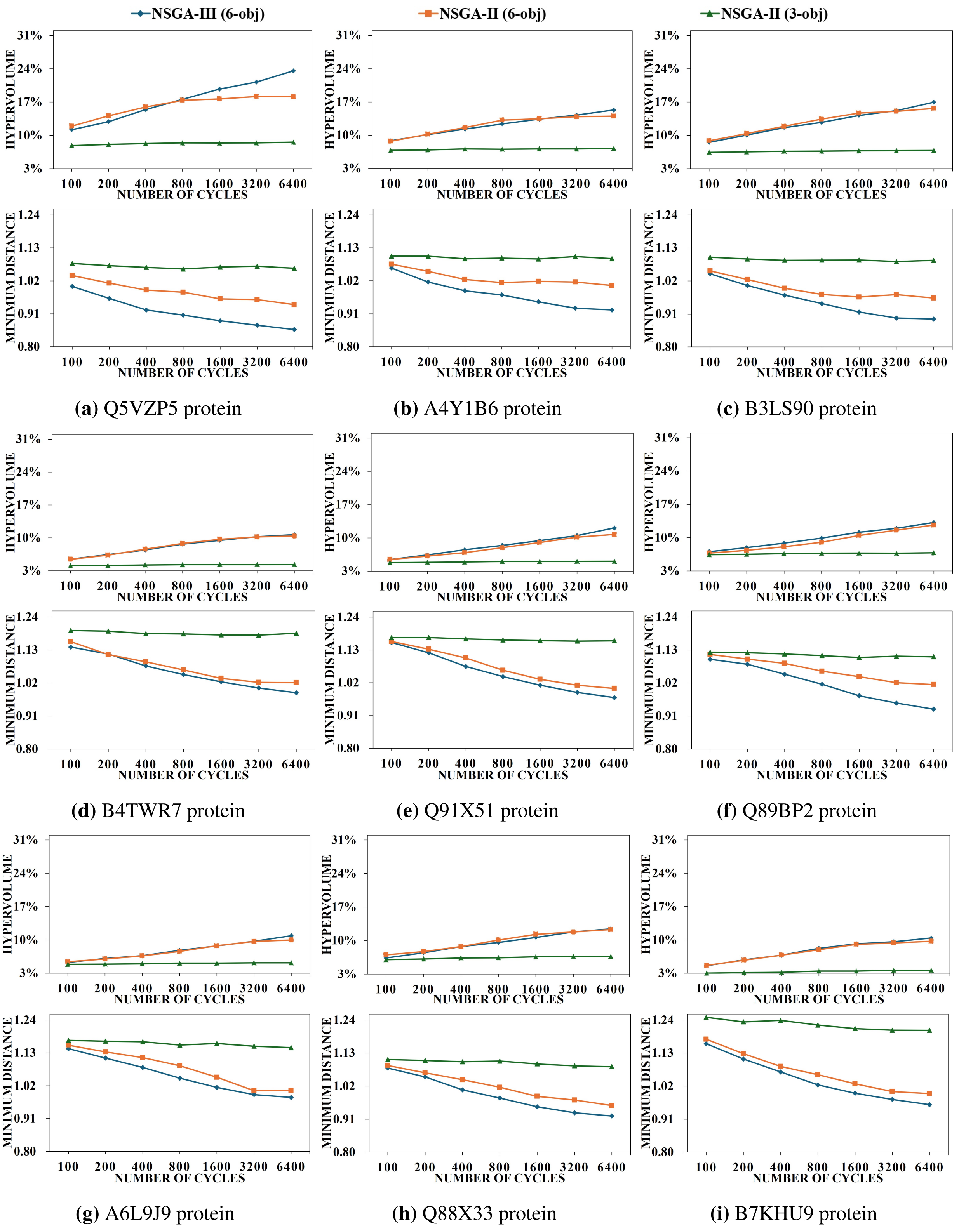
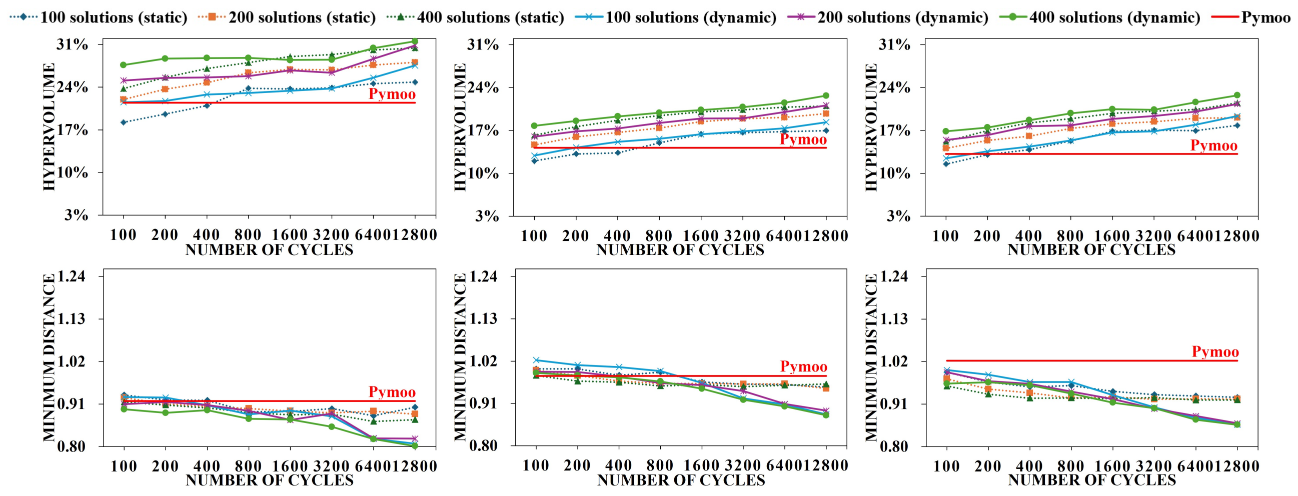


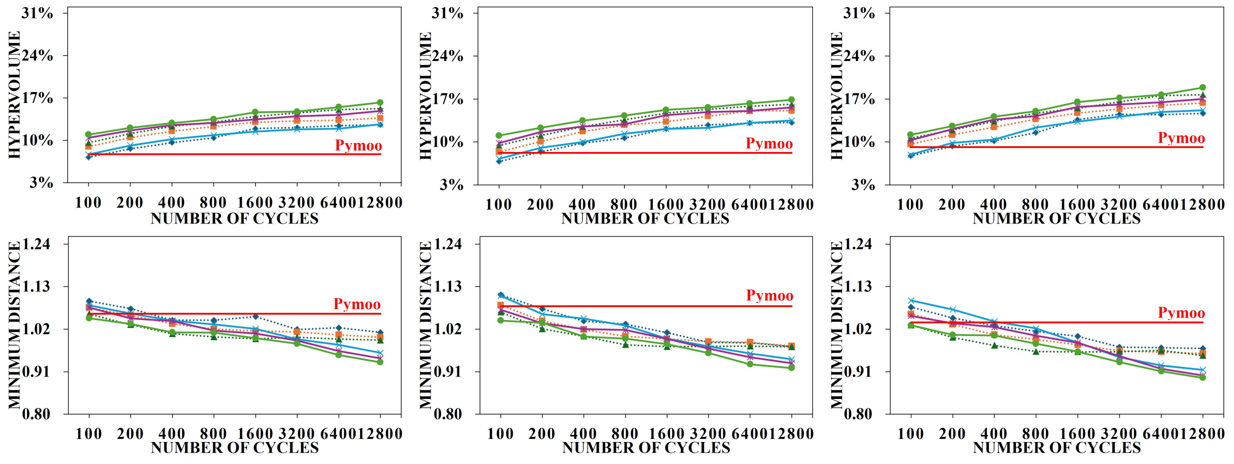
**Figure 4.** Experimental results of NSGA-III (6-obj), NSGA-II (6-obj), and NSGA-II (3-obj) methods.



**(a)** Q5VZP5 protein

**(b)** A4Y1B6 protein

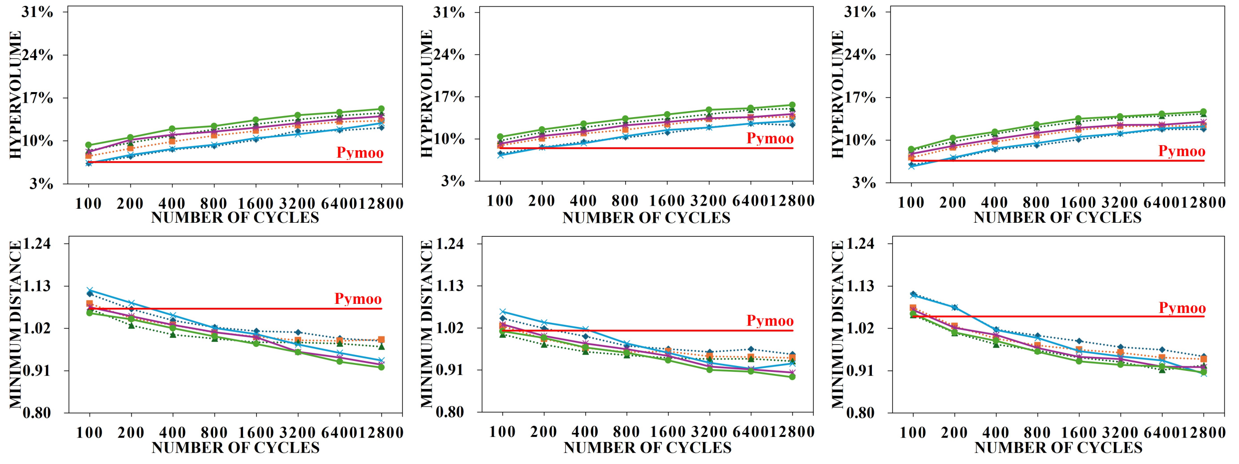
**(c)** B3LS90 protein



**(d)** B4TWR7 protein

**(e)** Q91X51 protein

**(f)** Q89BP2 protein



**(g)** A6L9J9 protein

**(h)** Q88X33 protein

**(i)** B7KHU9 protein

**Figure 5.** Experimental results of our methods with static (20%) and dynamic mutation probabilities and Pymoo (100 solutions and 100 cycles with 40% mutation probability).

As shown in Figure 5, owing to the exceptionally short execution times of our method, we tested it with various numbers of cycles and solutions to demonstrate its advantages. The experiments used 100, 200, and 400 solutions coupled with 100, 200, 400, 800, 1600, 3200, 6400, and 12,800 cycles. At 12,800 cycles, a mutation probability of 20% yielded the best results among the tested values of 1.25%, 2.5%, 5%, 10%, 20%, 40%, and 50% for all solution counts of 100, 200, and 400. However, for 100 solutions at 100 cycles, a 40% mutation probability provided better solutions, indicating that relatively higher mutation probabilities were more effective in yielding high-quality solutions when the number of cycles was low. Similarly, for 200 solutions and 400 solutions, using a mutation probability of 40% at 100 cycles yielded better hypervolume and minimum distance to the ideal point outcomes. Nevertheless, at 12,800 cycles, a 20% mutation probability yielded the best results. It was observed that the use of relatively lower mutation probabilities became more effective in enhancing the solutions as their quality improved to a certain extent. Therefore, we also present extended experimental results illustrating the effects of gradual reduction of the mutation probability. The mutation probability starts at 40% and decreases by a certain percentage every 100 cycles until it falls below 1%, at which point the mutation probability is fixed at 1%. Mutation decrease ratios of 10%, 20%, 30%, and 40% were tested.

The experimental results demonstrated that reducing the mutation probability by 10% every 100 cycles yielded the best outcomes for all solution counts of 100, 200, and 400 at 12,800 cycles. Moreover, the application of dynamically changing mutation probabilities led to superior hypervolume and minimum distance to the ideal point values at 12,800 cycles compared with the use of a fixed 20% mutation probability. In other words, with an equal number of solutions, dynamically reducing mutation probabilities outperformed the use of a fixed mutation probability at 12,800 cycles. Consequently, initiating higher mutation probabilities in the initial cycles and then gradually reducing them contributed to the improvement in the quality of solutions. The red straight lines in Figure 5 represent the hypervolume and minimum distance to the ideal point values obtained with Pymoo using 100 solutions and 100 cycles with a 40% mutation probability. In our expanded experiments, our method yielded solutions with significantly better hypervolume and minimum distance to the ideal point values outcomes by increasing both the number of solutions and cycles compared with those achieved by Pymoo. Therefore, our method effectively generates high-quality solutions by increasing both the number of cycles and the number of solutions.

## 5. Conclusions

In this study, we proposed a method for designing a set of CDSs aimed at maximizing protein expression levels by taking into account numerous influential objective functions: CAI, CPB, HSC, HD, GC3, and SL. Our approach, based on NSGA-III, was accelerated using GPUs to optimize efficiently these multiple objectives. As a result, our method generated the set of CDSs with similar quality in a significantly shorter time compared with the use of the Pymoo framework. This acceleration enables an increase in both the number of solutions and the number of cycles, thereby facilitating the maximization of protein expression levels. In other words, our method has the advantage of accommodating a larger number of solutions and cycles. We experimented by increasing the number of solutions to 400 and the number of cycles to 12,800. As a result, the increase in the number of solutions and cycles led to better hypervolume and minimum distance to the ideal values, demonstrating our method's capacity to provide higher-quality solutions.

In our extended experiments, we observed that a higher mutation probability was effective for relatively short cycles, while a lower mutation probability was preferred for larger cycles. In other words, when solutions attain a certain quality level, employing relatively low-mutation probabilities significantly improves solutions compared with the use of higher mutation probabilities. Consequently, our experiments, which began with a 40% mutation probability and were progressively reduced by 10% every 100 cycles until reaching 1%, showed that this gradual reduction in mutation probability yielded better solutions than maintaining a fixed 20% mutation probability. In addition, our other experimental results revealed that NSGA-II may have been more effective at lower cycle counts. However, beginning at 3200 cycles, NSGA-III consistently outperformed NSGA-II in providing better solutions for all proteins. Finally, our method is available as open-source software, accessible at: https://github.com/CAU-HPCL/Protein_encoding_v2_NSGA-III.

## 6. Discussion and future work

**Multi-criteria decision-making (MCDM) method.** In real application cases, we need a best compromised solution among Pareto optimal solutions. A common approach for this is the technique for order of preference by similarity to ideal solution (TOPSIS) [61]. TOPSIS is a concept that selects the alternative that is closest to the positive ideal solution (PIS) and farthest from the negative ideal solution (NIS), considering both the best and worst alternatives simultaneously to facilitate rational human decision making. This method allows users to set weights for each objective, thereby enabling them to choose their desired optimal solution.

**GPU-accelerated multi-objective optimization.** There are numerous challenges in bioengineering beyond simply maximizing protein expression levels. For instance, polymerase chain reaction (PCR) is a technology extensively employed in almost all biotechnology research areas for the mass replication of specific deoxyribonucleic segments. Optimizing primers is crucial for enhancing PCR performance and minimizing biases that could result in inaccurate conclusions during quantitative analysis [62]. Additionally, the early detection of cancer biomarkers can decrease mortality rates through treatments tailored to these biomarkers. Gene expression, a significant biomarker, can be effectively quantified using RNA sequencing technology; however, identifying cancer biomarkers poses a challenge owing to the generation of high-dimensional data. This issue can be addressed by selecting the most relevant genes, with multi-objective optimization methods being particularly well-suited to this gene selection challenge [63]. Furthermore, multi-objective optimization algorithms have found broad applicability in addressing problems across various domains extending beyond the scope of bioengineering. For instance, the adaptive polyploid memetic algorithm was utilized to address the truck scheduling problem, aiming to reduce operational costs at the cross-docking terminal [64]. Song et al. [65] proposed an improved particle swarm optimization algorithm to facilitate smooth path planning for mobile robots taking into account issues of local trapping and premature convergence. Our future work is aimed at modifying various multi-objective optimization algorithms to enable parallel processing and applying them to solve a diverse array of problems using GPUs. Furthermore, we intend to develop enhanced search strategies to identify effectively the Pareto-optimal solutions to various problems.

## Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

## Acknowledgments

This research was supported by the Chung-Ang University Graduate Research Scholarship in 2022. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. NRF-2022R1G1A1013586).

## Conflict of interest

The authors declare there is no conflict of interest.

## References

1. A. Zhou, B. Qu, H. Li, S. Zhao, P. N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: a survey of the state of the art, *Swarm Evol. Comput.*, **1** (2011), 32–49. https://doi.org/10.1016/j.swevo.2011.03.001
2. N. Gunantara, A review of multi-objective optimization: methods and its applications, *Cogent Eng.*, **5** (2018), 1502242. https://doi.org/10.1080/23311916.2018.1502242
3. P. Eskelinen, K. Miettinen, Trade-off analysis approach for interactive nonlinear multiobjective optimization, *OR Spectrum*, **34** (2012), 803–816. https://doi.org/10.1007/s00291-011-0266-z
4. H. Fang, M. Rais-Rohani, Z. Liu, M. F. Horstemeyer, A comparative study of metamodeling methods for multiobjective crashworthiness optimization, *Comput. Struct.*, **83** (2005), 2121–2136. https://doi.org/10.1016/j.compstruc.2005.02.025
5. F. Di Pierro, S. Khu, D. Savić, L. Berardi, Efficient multi-objective optimal design of water distribution networks on a budget of simulations using hybrid algorithms, *Environ. Modell. Software*, **24** (2009), 202–213. https://doi.org/10.1016/j.envsoft.2008.06.008
6. S. Fields, O. Song, A novel genetic system to detect protein–protein interactions, *Nature*, **340** (1989), 245–246. https://doi.org/10.1038/340245a0
7. S. Varambally, S. M. Dhanasekaran, M. Zhou, T. R. Barrette, C. Kumar-Sinha, M. G. Sanda, et al., The polycomb group protein ezh2 is involved in progression of prostate cancer, *Nature*, **419** (2002), 624–629. https://doi.org/10.1038/nature01075
8. G. Blander, L. Guarente, The sir2 family of protein deacetylases, *Annu. Rev. Biochem.*, **73** (2004), 417–435. https://doi.org/10.1146/annurev.biochem.73.011303.073651
9. S. P. Kaur, V. Gupta, Covid-19 vaccine: a comprehensive status report, *Virus Res.*, **288** (2020), 198114. https://doi.org/10.1016/j.virusres.2020.198114
10. M. Ahmad, M. Hirz, H. Pichler, H. Schwab, Protein expression in pichia pastoris: recent achievements and perspectives for heterologous protein production, *Appl. Microbiol. Biotechnol.*, **98** (2014), 5301–5317. https://doi.org/10.1007/s00253-014-5732-5

11. D. Fouque, K. Kalantar-Zadeh, J. Kopple, N. Cano, P. Chauveau, L. Cuppari, et al., A proposed nomenclature and diagnostic criteria for protein–energy wasting in acute and chronic kidney disease, *Kidney Int.*, **73** (2008), 391–398. https://doi.org/10.1038/sj.ki.5002585

12. J. Dehghani, A. Movafeghi, E. Mathieu-Rivet, N. Mati-Baouche, S. Calbo, P. Lerouge, et al., Microalgae as an efficient vehicle for the production and targeted delivery of therapeutic glycoproteins against sars-cov-2 variants, *Mar. Drugs*, **20** (2022), 657. https://doi.org/10.3390/md20110657

13. S. Huleani, M. R. Roberts, L. Beales, E. H. Papaioannou, Escherichia coli as an antibody expression host for the production of diagnostic proteins: significance and expression, *Crit. Rev. Biotechnol.*, **42** (2022), 756–773. https://doi.org/10.1080/07388551.2021.1967871

14. P. Gu, F. Yang, T. Su, Q. Wang, Q. Liang, Q. Qi, A rapid and reliable strategy for chromosomal integration of gene(s) with multiple copies, *Sci. Rep.*, **5** (2015), 9684. https://doi.org/10.1038/srep09684

15. C. A. Scorer, J. J. Clare, W. R. McCombie, M. A. Romanos, K. Sreekrishna, Rapid selection using g418 of high copy number transformants of pichia pastoris for high–level foreign gene expression, *Nat. Biotechnol.*, **12** (1994), 181–184. https://doi.org/10.1038/nbt0294-181

16. K. Tyo, P. K. Ajikumar, G. Stephanopoulos, Stabilized gene duplication enables long-term selection-free heterologous pathway expression, *Nat. Biotechnol.*, **27** (2009), 760–765. https://doi.org/10.1038/nbt.1555

17. R. Aw, K. M. Polizzi, Can too many copies spoil the broth? *Microb. Cell Fact.*, **12** (2013), 128. https://doi.org/10.1186/1475-2859-12-128

18. J. Buerstedde, N. Lowndes, D. G. Schatz, Induction of homologous recombination between sequence repeats by the activation induced cytidine deaminase (aid) protein, *Elife*, **3** (2014), e03110. https://doi.org/10.7554/eLife.03110

19. G. Terai, S. Kamegai, A. Taneda, K. Asai, Evolutionary design of multiple genes encoding the same protein, *Bioinformatics*, **33** (2017), 1613–1620. https://doi.org/10.1093/bioinformatics/btx030

20. S. T. Parvathy, V. Udayasuriyan, V. Bhadana, Codon usage bias, *Mol. Biol. Rep.*, **49** (2022), 539–565. https://doi.org/10.1007/s11033-021-06749-4

21. J. Athey, A. Alexaki, E. Osipova, A. Rostovtsev, L. V. Santana-Quintero, U. Katneni, et al., A new and updated resource for codon usage tables, *BMC Bioinf.*, **18** (2017), 391. https://doi.org/10.1186/s12859-017-1793-7

22. J. M. Comeron, M. Aguadé, An evaluation of measures of synonymous codon usage bias, *J. Mol. Evol.*, **47** (1998), 268–274. https://doi.org/10.1007/PL00006384

23. M. Gouy, C. Gautier, Codon usage in bacteria: correlation with gene expressivity, *Nucleic Acids Res.*, **10** (1982), 7055–7074. https://doi.org/10.1093/nar/10.22.7055

24. P. M. Sharp, W. Li, The codon adaptation index-a measure of directional synonymous codon usage bias, and its potential applications, *Nucleic Acids Res.*, **15** (1987), 1281–1295. https://doi.org/10.1093/nar/15.3.1281

25. G. A. Gutman, G. W. Hatfield, Nonrandom utilization of codon pairs in escherichia coli, *PNAS*, **86** (1989), 3699–3703. https://doi.org/10.1073/pnas.86.10.369

26. A. Tats, T. Tenson, M. Remm, Preferred and avoided codon pairs in three domains of life, *BMC Genomics*, **9** (2008), 463. https://doi.org/10.1186/1471-2164-9-463

27. M. Baeza, J. Alcaíno, S. Barahona, D. Sepúlveda, V. Cifuentes, Codon usage and codon context bias in xanthophyllomyces dendrorhous, *BMC Genomics*, **16** (2015), 293. https://doi.org/10.1186/s12864-015-1493-5

28. R. Prabha, D. P. Singh, S. Sinha, K. Ahmad, A. Rai, Genome-wide comparative analysis of codon usage bias and codon context patterns among cyanobacterial genomes, *Mar. Geonomics*, **32** (2017), 31–39. https://doi.org/10.1016/j.margen.2016.10.001

29. J. R. Coleman, D. Papamichail, S. Skiena, B. Futcher, E. Wimmer, S. Mueller, Virus attenuation by genome-scale changes in codon pair bias, *Science*, **320** (2008), 1784–1787. https://doi.org/10.1126/science.1155761

30. H. Seligmann, Cost minimization of ribosomal frameshifts, *J. Theor. Biol.*, **249** (2007), 162–167. https://doi.org/10.1016/j.jtbi.2007.07.007

31. H. Seligmann, D. D. Pollock, The ambush hypothesis: hidden stop codons prevent off-frame gene reading, *DNA Cell Biol.*, **23** (2004), 701–705. https://doi.org/10.1089/dna.2004.23.701

32. A. Gupta, T. R. Singh, Shift: server for hidden stops analysis in frame-shifted translation, *BMC Res. Notes*, **6** (2013), 68. https://doi.org/10.1186/1756-0500-6-68

33. P. Svoboda, A. D. Cara, Hairpin rna: a secondary structure of primary importance, *Cell. Mol. Life Sci.*, **63** (2006), 901–908. https://doi.org/10.1007/s00018-005-5558-5

34. C. Bao, S. Loerch, C. Ling, A. A. Korostelev, N. Grigorieff, D. N. Ermolenko, mRNA stem-loops can pause the ribosome by hindering a-site trna binding, *Elife*, **9** (2020), e55799. https://doi.org/10.7554/eLife.55799

35. M. V. Díaz-Galián, M. A. Vega-Rodríguez, Many-objective approach based on problem-aware mutation operators for protein encoding, *Inf. Sci.*, **613** (2022), 376–400. https://doi.org/10.1016/j.ins.2022.09.048

36. A. Watts, S. Sankaranarayanan, A. Watts, R. K. Raipuria, Optimizing protein expression in heterologous system: strategies and tools, *Meta Gene*, **29** (2021), 100899. https://doi.org/10.1016/j.mgene.2021.100899

37. B. Gonzalez-Sanchez, M. A. Vega-Rodríguez, S. Santander-Jiménez, A multi-objective butterfly optimization algorithm for protein encoding, *Appl. Soft Comput.*, **139** (2023), 110269. https://doi.org/10.1016/j.asoc.2023.110269

38. K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, *IEEE Trans. Evol. Comput.*, **18** (2013), 577–601. https://doi.org/10.1109/TEVC.2013.2281535

39. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evol. Comput.*, **6** (2002), 182–197. https://doi.org/10.1109/4235.996017

40. A. Razmi, M. Rahbar, M. Bemanian, Pca-ann integrated nsga-iii framework for dormitory building design optimization: energy efficiency, daylight, and thermal comfort, *Appl. Energy*, **305** (2022), 117828. https://doi.org/10.1016/j.apenergy.2021.117828

41. I. Khettabi, M. A. Boutiche, L. Benyoucef, NSGA-II vs NSGA-III for the sustainable multi-objective process plan generation in a reconfigurable manufacturing environment, *IFAC-PapersOnLine*, **54** (2021), 683–688. https://doi.org/10.1016/j.ifacol.2021.08.180

42. X. Li, H. Lv, D. Zeng, Q. Zhang, An improved multi-objective trajectory planning algorithm for kiwifruit harvesting manipulator, *IEEE Access*, **11** (2023), 65689–65699. https://doi.org/10.1109/ACCESS.2023.3289207

43. H. Ishibuchi, R. Imada, Y. Setoguchi, Y. Nojima, Performance comparison of nsga-ii and nsga-iii on various many-objective test problems, in *2016 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, (2016), 3045–3052. https://doi.org/10.1109/CEC.2016.7744174

44. S. Wang, Y. Wang, Y. Wang, Z. Wang, Comparison of multi-objective evolutionary algorithms applied to watershed management problem, *J. Environ. Manage.*, **324** (2022), 116255. https://doi.org/10.1016/j.jenvman.2022.116255

45. J. Blank, K. Deb, Pymoo: multi-objective optimization in python, *IEEE Access*, **8** (2020), 89497–89509. https://doi.org/10.1109/ACCESS.2020.2990567

46. B. Gonzalez-Sanchez, M. A. Vega-Rodríguez, S. Santander-Jiménez, J. M. Granado-Criado, Multi-objective artificial bee colony for designing multiple genes encoding the same protein, *Appl. Soft Comput.*, **74** (2019), 90–98. https://doi.org/10.1016/j.asoc.2018.10.023

47. B. Gonzalez-Sanchez, M. A. Vega-Rodríguez, S. Santander-Jiménez, Parallel multi-objective optimization approaches for protein encoding, *J. Supercomput.*, **78** (2022), 5118–5148. https://doi.org/10.1007/s11227-021-04073-z

48. B. Gonzalez-Sanchez, M. A. Vega-Rodriguez, S. Santander-Jimenez, Multi-objective protein encoding: redefinition of the problem, new problem-aware operators, and approach based on variable neighborhood search, *Inf. Sci.*, **500** (2019), 173–189. https://doi.org/10.1016/j.ins.2019.05.088

49. B. Gonzalez-Sanchez, M. A. Vega-Rodriguez, S. Santander-Jimenez, Multi-objective memetic meta-heuristic algorithm for encoding the same protein with multiple genes, *Expert Syst. Appl.*, **136** (2019), 83–93. https://doi.org/10.1016/j.eswa.2019.06.031

50. D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm, *J. Global Optim.*, **39** (2007), 459–471. https://doi.org/10.1007/s10898-007-9149-x

51. P. Hansen, N. Mladenović, J. A. Moreno Perez, Variable neighbourhood search: methods and applications, *Ann. Oper. Res.*, **175** (2010), 367–407. https://doi.org/10.1007/s10479-009-0657-6

52. N. Mladenović, P. Hansen, Variable neighborhood search, *Comput. Oper. Res.*, **24** (1997), 1097–1100. https://doi.org/10.1016/S0305-0548(97)00031-2

53. E. Elbeltagi, T. Hegazy, D. Grierson, A modified shuffled frog-leaping optimization algorithm: applications to project management, *Struct. Infrastruct. Eng.*, **3** (2007), 53–60. https://doi.org/10.1080/15732470500254535

54. I. Das, J. E. Dennis, Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems, *SIAM J. Optim.*, **8** (1998), 631–657. https://doi.org/10.1137/S1052623496307510

55. S. Arora, S. Singh, Butterfly optimization algorithm: a novel approach for global optimization, *Soft Comput.*, **23** (2019), 715–734. https://doi.org/10.1007/s00500-018-3102-4

56. K. Deb, E. Goodman, C. A. C. Coello, K. Klamroth, K. Miettinen, S. Mostaghim, et al., *Evolutionary Multi-Criterion Optimization: 10th International Conference, EMO 2019, East Lansing, MI, USA, March 10-13, 2019, Proceedings*, Springer, **11411** (2019).

57. D. D. Holcomb, A. Alexaki, U. Katneni, C. Kimchi-Sarfaty, The kazusa codon usage database, cocoputs, and the value of up-to-date codon usage statistics, *Infect., Genet. Evol.*, **73** (2019), 266–268. https://doi.org/10.1016/j.meegid.2019.05.010

58. *Kazusa DNA Research Institute*, Saccharomyces cerevisiae gc contents, 2023. Available from: https://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=4932.

59. The UniProt Consortium, UniProt: the Universal Protein Knowledgebase in 2023, *Nucleic Acids Res.*, **51** (2023), D523–D531. https://doi.org/10.1093/nar/gkac1052

60. D. Kim, J. Kim, Optimization of designing multiple genes encoding the same protein based on NSGA-II for efficient execution on GPUs, *Electron. Res. Arch.*, **31** (2023), 5313–5339. https://doi.org/10.3934/era.2023270

61. G. Tzeng, J. Huang, *Multiple Attribute Decision Making: Methods and Applications*, CRC press, 2011.

62. D. L. Church, L. Cerutti, A. Gürtler, T. Griener, A. Zelazny, S. Emler, Performance and application of 16s rrna gene cycle sequencing for routine identification of bacteria in the clinical microbiology laboratory, *Clin. Microbiol. Rev.*, **33** (2020). https://doi.org/10.1128/CMR.00053-19

63. B. Xue, M. Zhang, W. N. Browne, X. Yao, A survey on evolutionary computation approaches to feature selection, *IEEE Trans. Evol. Comput.*, **20** (2015), 606–626. https://doi.org/10.1109/TEVC.2015.2504420

64. M. A. Dulebenets, An adaptive polyploid memetic algorithm for scheduling trucks at a cross-docking terminal, *Inf. Sci.*, **565** (2021), 390–421. https://doi.org/10.1016/j.ins.2021.02.039

65. B. Song, Z. Wang, L. Zou, An improved pso algorithm for smooth path planning of mobile robots using continuous high-degree bezier curve, *Appl. Soft Comput.*, **100** (2021), 106960. https://doi.org/10.1016/j.asoc.2020.106960