GOM-Based Compatible Substitutions Optimization for Variable-Length Representation Gray-Box Problems*

Maciej Komosinski Konrad Miazga

Poznan University of Technology
Institute of Computing Science
Poznan, Poland
maciej.komosinski@cs.put.poznan.pl

Abstract

Effective recombination operators utilizing interdependence of genes ensure that specific arrangements or combinations of genes are preserved, allowing offspring to inherit beneficial traits from both parents without disrupting important gene interactions. However, such operators are easiest to implement for fixed-length genetic representations such as vectors of genes. In this work, we show that for some problems with variable-length representations, it is possible to design an algorithm that employs the GOM (Gene-pool Optimal Mixing) operator without the need to learn dependencies between specific genes. Instead, our approach – Compatible Substitutions Optimization (CoSO) – leverages expert-driven models of compatible substitutions that take advantage of the characteristics of the representation. Our experiments indicate that the proposed method performs better than standard evolutionary algorithms on a problem of evolving tall 3D structures, while also providing significant potential for further enhancements.

1 Introduction

The importance of the concept of *building blocks* for evolutionary algorithms has been recognized long ago by John Holland [3]. Building blocks are components of solutions that improve their fitness when included. To be effective, an optimization algorithm must both preserve and increase the prevalence of existing building blocks within the population, while also seeking out new ones.

Although all evolutionary algorithms do that in an implicit way, there are some algorithms that try to perform this task more explicitly, recognizing these blocks when they appear in the population and proliferating them without disruptions. One of such algorithms is Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [1], which learns the epistatic structure of the problem. This allows for the optimal mixing of the building blocks present in the population, which in turn helps quickly find high-quality solutions.

While GOMEA has shown remarkable performance on certain problems, it is constrained by its reliance on positional discovery of building blocks. This approach necessitates that these blocks remain fixed in position and requires a representation of fixed length. In this paper, we propose a new optimization algorithm called Compatible Substitutions Optimization (CoSO), which utilizes the GOM operator in variable-length representations. CoSO allows for proliferating building blocks found during the search without assigning them to fixed positions in the genotype. We show that its performance is similar or better than that of standard evolutionary algorithms, while leaving a lot of space for further improvements.

*The final version of this paper appeared in Genetic and Evolutionary Computation Conference (GECCO '25) Companion, Malaga, Spain, 2025. http://dx.doi.org/10.1145/3712255.3726717.

2 The CoSO Algorithm

CoSO is inspired by the GOM (Gene-pool Optimal Mixing) operator – the only operator used in GOMEA [1]. It takes one solution as its input (which we will call the *recipient*) and produces one solution as its output. Within the GOM operator, GOMEA goes through all the elements of the Family of Subsets (FOS). GOMEA swaps the values of recipient's genes on positions from the currently processed FOS element for the values of the same genes, taken from a random *donor* solution in the current population. A substitution is accepted only if it does not decrease the fitness of the recipient.

FOS is an important part of GOMEA, as it contains the information about the epistatic linkage between genes, and it is used to decide what types of donations between two solutions are worth testing, reducing their number and improving quality.

In its original formulation, GOMEA is limited to fixed-length representations of solutions, where the value of each gene can be varied independently of the others. This limitation facilitates learning the FOS model. However, in problems with variable-length representations, the exact positions of sub-solutions within the genotype usually matter less than the context in which they appear. A sequence appearing at the start of one genotype could be beneficially used in the middle of another.

In the CoSO (*Compatible Substitutions Optimization*) algorithm introduced here, the population is repeatedly improved using the GOM operator, while eliminating the need to learn the FOS model in favor of generating a pool of matching substitutions based on a Substitution Compatibility Function (SCF). This function represents a form of expert knowledge about which genes should be exchanged together to minimize their influence on the fitness contribution of the rest of the solution. Since CoSO requires this knowledge, the optimization problems it can address are categorized as gray-box problems. Examples of such problems can be found in genetic programming and in evolutionary design.

The SCF takes two sub-solutions as arguments and determines how well a part of the donor solution can substitute a part of the recipient solution. The lower the value of the SCF, the better the substitution.

There are two ways in which the SCF may decide if the substitution is good. The first way is to verify the impact of the substitution on the way the rest of the solution is going to be projected into a phenotype and, therefore, on its impact on the contribution of the rest of the solution to the final fitness of a solution – a low SCF value will indicate a weak epistatic link to the rest of the genotype. The second way is to verify if a substitution serves the same purpose. For problems in the area of automated design or evolutionary robotics, an example would be substituting a "leg" of a structure for another realization of a "leg". Ideally, a good SCF should weigh both of these aspects of a good substitution, as they both align with the recommendation that genetic operators should be designed to maximize fitness-distance correlation (FDC) [4].

In CoSO, the GOM operator takes as arguments the recipient genotype and the whole population. All sub-solutions available in the population (including the recipient itself) constitute a list of substitution candidates. The form of a sub-solution may depend on the representation used in the problem. For example, for sequences, a sub-solution will be a subsequence, for trees – a subtree, etc.

For each sub-solution present in the recipient, we evaluate the value of the SCF for that sub-solution and all sub-solutions in the list of candidates. Then, we match all fragments of the recipient solution with all sub-solutions from the list of candidates, for which the value of the SCF is less than or equal to a predefined threshold θ. All matching substitutions are then tested in a random order, and only the substitutions that improve the fitness of the recipient are accepted. Accepting a substitution typically invalidates some matches, as the modified solution fragment overlaps with some of the remaining fragments. Such overlapping matches are discarded.

To ensure a diverse collection of building blocks to be used by the algorithm, the initial population is filled with randomly generated solutions of maximum allowed length.

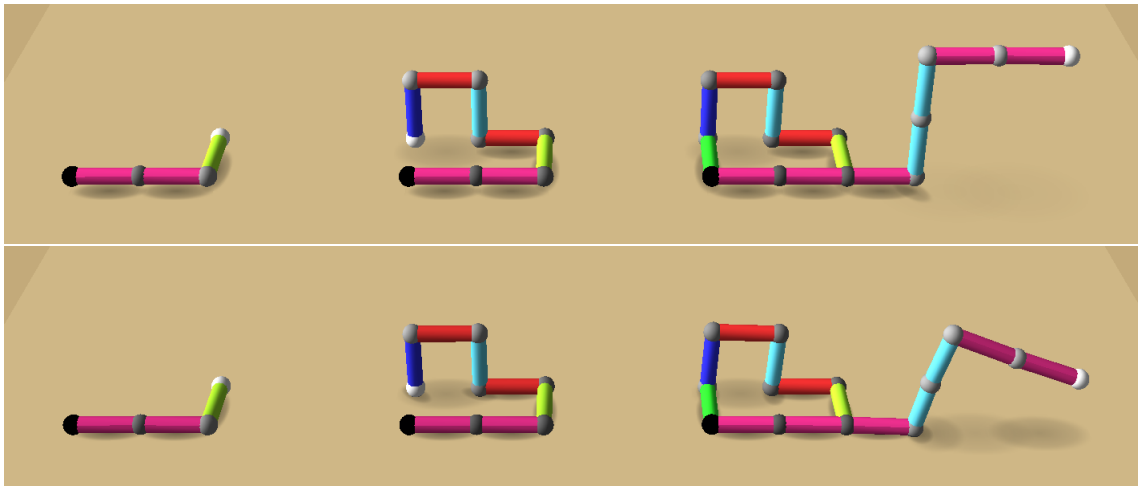

Figure 1: Top: phenotypes resulting from three sample genotypes. Bottom: the same phenotypes in simulation, after stabilization. From left to right: RRF (*vertpos* = −0.01), RRFLULD (*vertpos* = 0.25), and RRFLULDBRRRUURR (*vertpos* = 0.42). The first created part is shaded black, and the last part is shaded white.

3 The benchmark

3.1 The optimization problem

To validate the approach described above, we apply it to a simple automated design optimization problem. The problem involves designing of the tallest three-dimensional structure built from elastic segments of the same length, aligned with the three axes (X, Y, Z).

The height of a 3D structure is measured not by the elevation of its highest element, but by the vertical position of the center of gravity, hence we further call this optimization criterion *vertpos*.

The genetic representation of each solution is very simple and consists of a sequence of six letters that indicate the direction of movement of a "3D turtle" as in simple 3D turtle graphics: L (left), R (right), B (back), F (forth), D (down), U (up). This sequence is interpreted as a string of characters, each of which moves the position of the turtle in a given direction in 3D, creating a body (i.e., phenotype) segment.

Fig. 1 illustrates how 3D structures are constructed from three sample genotypes. Note that the movement along each axis is visualized in different colors (reddish for the X-axis, greenish for the Y-axis, blueish for the Z-axis). The left and middle structures are straightforward in their direct mapping from individual genes to the phenotype. The genotype on the right illustrates what happens when the path overlaps: the turtle moves along the existing segments that were created earlier, no additional parts are created, and the existing structure is not modified.

To avoid perfectly horizontal and vertical segments, small deterministic noise is added to each part based on the number of the part in the created sequence. This helps prevent the formation of unrealistic or perfectly symmetrical structures, requiring them to be more robust in simulation. For example, it becomes impossible to create a stable, perfectly vertical pole.

To calculate the fitness of a genotype, the resulting phenotype is simulated in the Framsticks environment [5, 6]. The phenotype first undergoes the stabilization period, when its center of gravity needs to move less than the set threshold (0.005) for 100 simulation steps. After this is fulfilled, the structure is simulated for another 10 000 steps and, in each step, the vertical coordinate of the center of gravity is recorded. The fitness (the *vertpos* value) becomes the average of these coordinates. Fitness values are provided for sample structures in the caption of Fig. 1.

For the purpose of the experiments conducted in this paper, we limit the maximum length of the genotypes to 20 characters.

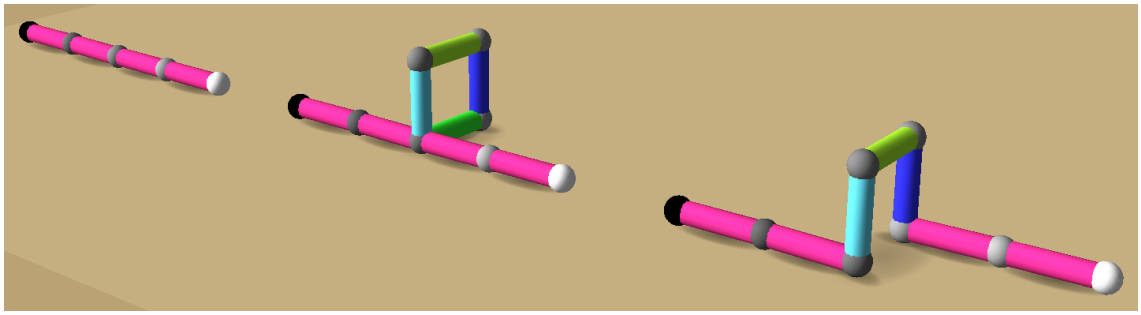

Figure 2: Effects of different types of substitutions on the phenotype of a solution. From left to right: the original genotype (RRRR), the genotype after a delta-preserving substitution (RRUFDBRR), and the genotype after a delta-changing substitution (RRUFDRR).

3.2 The SCF for the optimization problem

Although the formulation of the SCF for the *vertpos* benchmark is specific to the genetic representation considered, it demonstrates how such a function can be constructed. In this case, we will not use specific knowledge of the problem itself, limiting ourselves instead to the knowledge of the genetic representation. Therefore, the same SCF could be used to solve other problems using the same genetic representation.

As the genetic representation uses turtle-like movement to translate genotypes to phenotypes (physical structures), changing a single character in a genotype can affect the placement of all the following segments (sticks) in relation to the preceding sticks. Therefore, we can expect strong epistatic links between a given gene and all subsequent genes. In order to minimize this effect, changes applied to a genotype at some position should not shift the following sticks in relation to the preceding sticks, as illustrated in Fig. 2. This can be achieved by performing substitutions of equal *deltas*. We understand delta $\delta(s)$ of a sequence s to be a vector pointing from the initial point of the path drawn by a sequence to its terminal point. Since swapping two sequences of equal deltas will not affect the relation between the preceding and the following sticks, it should have – on average – a lower impact on the fitness contribution of the rest of the genotype than arbitrary substitutions.

The SCF for the considered genetic representation is calculated as the Euclidean distance between the deltas for arguments of the function, $SCF(s_1, s_2) = \|\delta(s_1) - \delta(s_2)\|$, with the threshold $\theta = 0$.

3.3 Details of the CoSO implementation

Empty sequences and loops Some sequences of gene symbols result in a loop in the 3D structure. Notice that empty sequences also have the delta of zero: $\delta(\text{“ ”}) = (0, 0, 0)$. Empty sequences will always match with loops, which in practice allows for insertion of loops in any place in a sequence, or – conversely – deletion of any loop existing in a genotype. To verify the effect of including this type of substitutions, in this paper we test both scenarios: including the empty sequences as proper sub-solutions and excluding them.

Single-position deletions One of the weaknesses of the proposed SCF function is its struggle to modify the distance between two substructures in a structure, and its inability to change the parity of the length of the genotype. To reduce the impact of these flaws, in this work we test a scenario where substitutions constituting single-character deletions are also included in the pool of matching substitutions, even though $SCF(s, \text{“ ”}) = 1$ for all s where $len(s) = 1$.

Substitution length limit We add a limitation to the set of subsequences taken from donors, omitting those with lengths greater than the full donor genotype length minus two. This is done

to avoid situations where a solution can be fully replaced by another, better solution in situations where $\delta(recipient) = \delta(donor)$.

Elite archive and multiple restarts Once there are no matching substitutions left in the population that could improve the quality of a solution, all the solutions from the population are moved to a second population, called elite archive. Then, the elite archive is trimmed to its maximum size (which is a parameter of the algorithm) by removing the worst-performing solutions. The population is then filled with as many new random solutions as it initially contained. The solutions contained in the elite archive do not undergo optimization, but they are still used by the GOM operator, along with solutions from the current population, to improve recipients.

Full deltas learning The proposed SCF cannot change the delta of the entire genotype. To avoid wasting computational time optimizing unpromising solutions, we estimate the normal distribution of deltas of full solutions in the elite archive, and then draw new random solutions according to that distribution. To prevent new solution deltas from converging when the best solution deltas do, the standard deviations of their distributions are artificially increased.

4 Experimental setup

We perform a comparison of the results obtained by CoSO with two experiments, i.e., two different grids of parameters N (the population size), E (the elite archive size), and L (the level of scope of matching substitutions considered by the GOM operator). For $L = 0$, we omit empty sequences, so loops cannot be added nor removed from the genotypes. For $L = 1$, we allow the addition and removal of loops. For $L = 2$, we also allow single-position deletions.

For the first experiment, $N \in \{1, 5, 10\}$, $E \in \{10, 20, 30\}$, $L \in \{0, 1, 2\}$. For the second experiment, $N = 1$, $E \in \{1, 2, 5, 10, 20, 30\}$, and $L = 2$.

In addition to CoSO, we also perform experiments using a standard generational evolutionary algorithm (EA) with tournament selection, which will serve as a reference point for the performance of CoSO. The algorithm is implemented using the DEAP framework [2]. After initial broad grid search experiments, we focused on the parameter values that performed especially well on the *vertpos* benchmark: population size $N \in \{100, 200, 500\}$, tournament size $k \in \{5, 10\}$, mutation probability $p_{mut} = 0.9$, crossover probability $p_{xov} = 0.1$, elitism (one best solution preserved) *elitism* $\in \{$*off*, *on*$\}$.

For each parameterization of the algorithms, 30 independent runs were performed. Each run consisted of 300 000 evaluations of unique solutions.

5 Results

The first experiment The series for $L = 1$ were omitted from Fig. 3 to improve its readability. The best results can be seen for a population consisting of a single solution ($N = 1$). Under such a parametrization, the singular solution present in the population can be viewed as a randomly shaped vessel, which the GOM operator attempts to fill with the already discovered building blocks in new configurations, while being constrained by the particular shape of that vessel. In that way, CoSO resembles a local search algorithm with multiple restarts, where the neighborhood is defined by the accumulated knowledge of all previous solutions.

In the long run, the best results are found for a single-solution population with a fully expanded pool of matching substitutions and a small archive of elites ($E \leq 20$).

Results get worse as the size of the population increases. This may mean that the matching substitutions provided by the solutions in the population may be hindering the search process more than they are helping to explore the solution space.

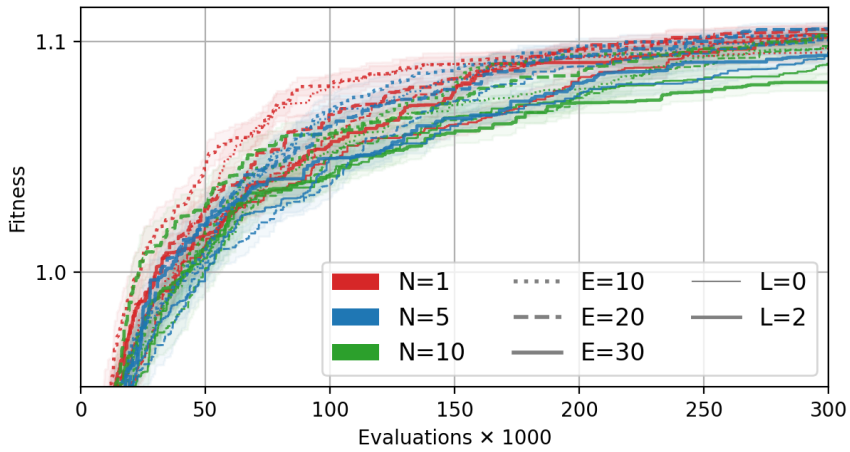

Figure 3: Fitness of the best solution found so far for several parameterizations of the CoSO algorithm using **elite-archive** approach on the *vertpos* task. Each series is averaged over 30 independent run of the algorithm. Bands around the series indicate 20% of the standard deviation.

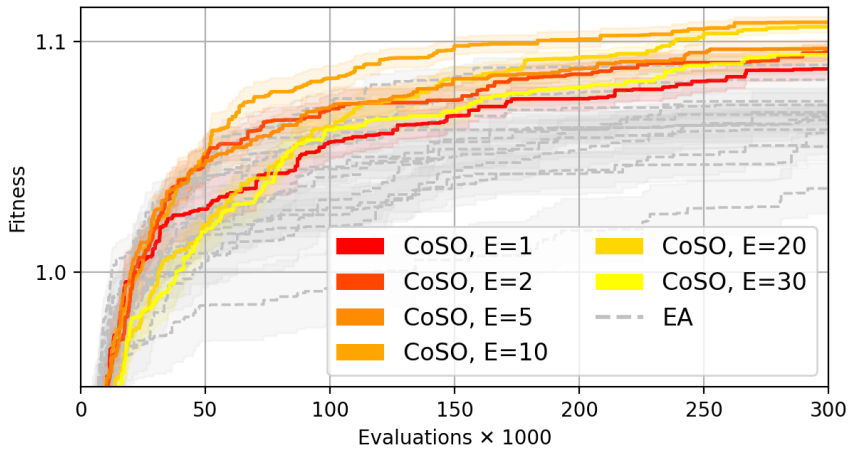

Figure 4: Fitness of the best solution found so far for CoSO (solid lines in color; $N = 1$, $L = 2$) algorithm using **elite-archive** approach, and standard EA (gray dashed lines) on the *vertpos* task. Each series is averaged over 30 independent runs of the algorithm. Bands around the series indicate 20% of the standard deviation.

For small populations such as $N = 1$, having an expanded pool of matching substitutions is preferred over a smaller set. On the other hand, for larger populations, the opposite can be observed. A possible explanation is that, in larger populations, the number of possible substitutions is already sufficiently large (since more solutions contribute their sub-solutions), and adding more options does not improve the solutions but instead slows down the search.

CoSO seems to benefit from having a smaller archive of elites. This may indicate that adding more solutions to the archive yields diminishing returns while increasing the size of the pool of matching substitutions linearly. In other words, it slows down the search process more than it speeds it up.

The second experiment Here we consider the best parametrization found so far ($N = 1$, $L = 2$) while varying the size of the elite archive E. Additionally, we compare the performance of CoSO under this parameterization to the performance of a standard EA over a range of parameterizations. The results of this comparison are shown in Fig. 4.

In the initial phase of the search process, CoSO performance for different sizes of elite archives are comparable. In the long run, a moderately sized archive ($E = 10$) performs best, striking a balance between the advantage of a larger collection of building blocks and the increased effort required to search through it.

The gray, dashed lines in the background of Fig. 4 correspond to different parametrizations of a standard EA. None of the tested EA parametrizations surpass the performance of CoSO. However, CoSO, when utilizing a single-solution population combined with an expanded pool of matching substitutions and an elite archive of size 10, appears to outperform EA over longer time frames.

6 Conclusions and future work

This paper lays the groundwork for research in Compatible Substitution Optimization (CoSO). We introduce a novel optimization algorithm designed for gray-box problems that involve variable-length representations, such as genetic programming and evolutionary design. The underlying logic and methodology of the algorithm were explored, and its application was demonstrated using a simple evolutionary design problem.

In its current form, the proposed algorithm performs better than well-parameterized EAs, and represents an initial step toward developing efficient, building-block-preserving algorithms tailored to gray-box problems with variable-length representations. Further improvements could be achieved by training the SCF function based on the data gathered during the search. Another idea would be to use an elite *sub-solution* archive instead of the archive of elite solutions, which could be constructed to contain a list of all substitutions encountered during the run of the algorithm, divided into buckets based on the characteristics of sub-solutions – in the case of the problem considered in this paper, delta values. Then, each bucket in the archive could be trimmed independently based on the fitness of the best solution from which it was taken.

Acknowledgements

This research was supported by Poznan University of Technology grant no. 0311/SBAD/0752.

References

[1] Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A. N. Bosman. Parameterless gene-pool optimal mixing evolutionary algorithms. *Evolutionary computation*, 32(4):371–397, 2024.

[2] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

[3] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[4] Terry Jones, Stephanie Forrest, et al. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *ICGA*, volume 95, pages 184–192, 1995.

[5] Maciej Komosinski and Szymon Ulatowski. Framsticks: Creating and understanding complexity of life. In Maciej Komosinski and Andrew Adamatzky, editors, *Artificial Life Models in Software*, chapter 5, pages 107–148. Springer, London, 2nd edition, 2009. URL: `http://www.springer.com/978-1-84882-284-9`.

[6] Maciej Komosinski and Szymon Ulatowski. Framsticks website, 2025. URL: `http://www.framsticks.com`.