Genet Program Evolvable Mach (2017) 18:247–277
DOI 10.1007/s10710-016-9282-8

Solving metameric variable-length optimization problems using genetic algorithms

Matthew L. Ryerkerk[1] · Ronald C. Averill[1] · Kalyanmoy Deb[1] · Erik D. Goodman[1]

Received: 23 March 2016 / Revised: 22 July 2016 / Published online: 26 September 2016

Abstract In many optimization problems, one of the goals is to determine the optimal number of analogous components to include in the system. Examples include the number of sensors in a sensor coverage problem, the number of turbines in a wind farm problem, and the number of plies in a laminate stacking problem. Using standard approaches to solve these problems requires assuming a fixed number of sensors, turbines, or plies. However, if the optimal number is not known a priori this will likely lead to a sub-optimal solution. A better method is to allow the number of components to vary. As the number of components varies, so does the dimensionality of the search space, making the use of gradient-based methods difficult. A metameric genetic algorithm (MGA), which uses a segmented variable-length genome, is proposed. Traditional genetic algorithm (GA) operators, designed to work with fixed-length genomes, are no longer valid. This paper discusses the modifications required for an effective MGA, which is then demonstrated on the aforementioned problems. This includes the representation of the solution in the genome and the recombination, mutation, and selection operators. With these modifications the MGA is able to outperform the fixed-length GA on the selected problems, even if the optimal number of components is assumed to be known a priori.

Keywords Variable-length genetic algorithms · Metameric variable-length problems · Metameric genetic algorithm · Sensor coverage · Windfarm layout · Composite laminate design

✉ Matthew L. Ryerkerk
matt.ryerkerk@gmail.com

[1] Michigan State University, East Lansing, MI 48823, USA

1 Introduction

Most optimization algorithms use a fixed-length vector of variables to represent a solution. Optimality theory is largely based on the fixed-length assumption, as there is no clear definition of a gradient vector for a variable-length search space. However, there exist many examples of variable-length problems where the number of variables is not necessarily fixed.

Standard optimization algorithms can be employed by assuming a fixed number of variables, but a suboptimal length will lead to a suboptimal solution. The algorithm could be performed iteratively, changing the prescribed length until the optimal solution is found. This is inefficient and impractical if the range of possible lengths is large. A better approach is to use an algorithm whose solution vectors can vary in length, known as variable-length algorithms.

This paper focuses on a subset of variable-length problems: those whose solutions are formed by a set of analogous components. Examples include wireless sensor network (WSN), wind farm layout, and composite laminate stacking problems. Solutions to such problems are valid regardless of the number of components (e.g. sensors, turbines, or plies) used. Optimality may require a particular number of components, which may not be known a priori. Each component will be defined using the same set of variables (e.g. x- and y-position, or ply angle), but these values will most likely vary among components.

We propose to classify such problems as "metameric variable-length problems". Metameric is a term used in biology to describe a body that is divided into a series of structurally similar, but not necessarily identical, segments. Examples include the body segments of a centipede, or the vertebrae in a spinal column. The genome used for metameric variable-length problems will also be composed of a series of segments. Each segment shares the same structure, containing the variables necessary to define one component (e.g. sensor, turbine, or ply) in the solution. This paper proposes the term "metavariable" to describe an individual segment of the genome.

In biology the number of segments is usually fixed for an individual species. However, in some cases that number can vary for a single family of species. For example, the *Mecistocephalidae* family of centipedes varies between having 47 and 101 leg-bearing segments [1]. There are also examples of the number of segments varying within a single species, such as a population of *Strigamia maritima* centipedes [2]. The number of segments in metameric structures may also change when observing the macroevolution of a species.

Other existing variable-length methods, such as Genetic Programming in its many forms [3], typically use representations that are dramatically different from those required by metameric variable-length problems. More general forms of these algorithms could eventually give rise to the required solution structure, however this is inefficient.

A better solution would be to adapt a genetic algorithm (GA) [4–6] to utilize the metameric structure. While GA has been applied extensively to a wide range of applications it has typically focused primarily on problems that use a fixed-length

genome. Classical GAs have sometimes been modified for variable-length problems by using a fixed-length genome and allowing some alleles to remain unexpressed in the evaluated solution [7–18]. This paper refers to this as the hidden-metavariable representation and will compare its performance against other methods using variable-length genomes.

The proposed algorithm will be referred as the metameric genetic algorithm (MGA), in which the genome is formed by a series of metavariables. The genome cannot contain any partially defined metavariables. In this paper the genome will typically be variable-length, with the exception of an implementation using the hidden-metavariable representation. MGAs using either representation are considered variable-length algorithms and can be used to solve metameric variable-length problems. It is feasible that some problems also call for a fixed number of "global" variables, in addition to a set of metavariables. Such a problem is not explored in this paper, but the findings in this paper could be applied to the variable-length portion of the genome while applying traditional GA operators to the fixed-length portion.

Since the classical GA operators were designed for fixed-length genomes, they must be modified to handle metameric variable-length genomes. It is not clear, for example, how the recombination operator should produce children from two parents of different lengths. A key to the success of a GA is its ability to form building blocks, short subsequences (or schemata) of the genome that have a positive influence on the fitness of an individual. This idea still holds true for MGA and needs to be considered when modifying the GA operators. The recombination operator must be able to perform "respectful" recombination [19] and to minimize the risk of disruption [20]. A respectful recombination operator is one that produces children containing any schemata shared by both parents. Disruption occurs when a building block is not fully inherited from a parent by a child.

To enable effective search of the design space, a population must also maintain a level of diversity. The mutation operator is a key source of such diversity. Since the solution length is no longer fixed, the selection operator should ensure a level of length-based diversity from one generation to another.

As a continuation of previous work on these topics [21], this paper focuses on the how GA operators may be effectively modified for use in MGA with an emphasis on the recombination and selection operators. The performance of several potential operators, as well as the hidden-metavariable representation, are compared. MGA is then also compared to a fixed-length GA—one that requires that the optimal number of components be known a priori. Sensor coverage, wind farm, and composite laminate stacking problems are used to evaluate the performance of each algorithm.

The remainder of this paper is organized as follows. Section 2 contains a short review and formulation of each benchmark problem. Section 3 presents the modifications made to the GA operators for use in MGA. Section 4 presents the study results, with a discussion of important findings in Sect. 5 followed by conclusions in Sect. 6.

2 Benchmark problem formulation

Each of the following subsections contains a brief literature review and a description of the objective function for the benchmark problems considered herein. Section 2.1 describes a sensor coverage problem, Sect. 2.2 a wind farm layout problem, and Sect. 2.3 a laminate stacking problem. It is important to note that each of these problems is used only as a test case for the MGA. For this reason, no problem-specific heuristics are included in the optimization process.

2.1 Sensor coverage problem

Wireless sensor networks have been studied extensively in the literature with a great range of considerations and objectives. An ideal sensor model provides 100 % coverage inside a circular domain. Realistic considerations include directional sensors [22], sensors with probabilistic detection rates [23], irregularly shaped sensing ranges [24, 25], or obstructions in the domain [26]. Network connectivity, longevity, and fidelity [27] are also considered frequently as objectives. Some studies assume a random deployment of sensors—for example, dropped from the air—and focus on determining a subset of sensors that achieves the most efficient coverage [28].

This paper seeks to optimize the individual placement and sensing ranges of a set of sensors such that the coverage of a given domain is maximized in a cost-efficient manner. Since the network-based objectives are not considered here, the problem will be referred to as simply the "sensor coverage" problem. This provides an easily interpreted and visualized problem, but one that remains considerably complex. If maximal coverage efficiency of an unbounded region using circles is desired, the optimal placement is known to be on the nodes of an equilateral triangle lattice [29]. This type of grid placement and related ones are often used as a starting point when forming sensor deployment patterns with additional considerations, such as connectivity [30]. However, if the region is bounded, this solution may not be optimal due to the amount of wasteful coverage that extends past the bounded region.

Many techniques have been used to optimize the placement of wireless sensor networks or equivalent problems. A quasi-Newton method was used to determine the minimal radius required to cover a square with up to 30 circles [31]. Computational geometry techniques, commonly Delaunay triangulation and Voronoi diagrams, are employed to improve sensor placement by identifying regions with the worst coverage [32, 33]. Other examples of techniques used include particle swarm optimization [34], and greedy heuristics [23].

GAs have also been used to solve wireless sensor network problems. Ting et al. [35] used variable-length genomes to allow the number of sensors to vary. A mix of cut-and-splice crossovers (allowing the number of sensors to vary) and uniform crossovers (to recombine individual sensors) were utilized. Jourdan and Weck [36] used GAs to optimize the coverage and lifetime of a network; however, they assume a fixed number of sensors. Marks [37] provides a survey of several studies focused

on wireless sensor network design using a multi-objective GA. Chan et al. [7] use a jumping-gene genetic algorithm with a hidden-metavariable representation to determine base station placement in a wireless network. Tonda et al. [38] use a cooperative co-evolution algorithm to solve a functionally similar coverage problem, using a set of lamps to illuminate a room.

In the current study, sensors are placed in a bounded domain such that the coverage is maximized while minimizing the cost. Each sensor is defined by an x- and y-position, as well as a sensing radius. Sensors with a larger sensing radius are more expensive, but also more cost efficient in terms of area covered. The coverage and cost objectives are combined into a single weighted sum objective function, given in Eq. (1). The first term is a penalty that scales linearly with the uncovered area in the domain. The second term is the cumulative cost of all sensors; each sensor has a base cost plus a cost that scales with the sensing area. The relative weighting of these two terms affect which solutions will be considered optimal. This paper places a higher relative weight on the penalty for uncovered areas; this results in optimal solutions that achieve an almost complete (>99 %) coverage of the field.

$$\begin{aligned} \text{Minimize } f(x) = & 1000 * \left(1 - \frac{\boldsymbol{A} \cap \bigcup_{i=1}^{n} \pi r_i^2(x_i, y_i)}{\boldsymbol{A}} \right) + \sum_{i=1}^{N} \left(1 + 10 * r_i^2\right) \\ & -1 < x_i < 1 \quad i = 1, 2, \ldots N \\ & -1 < y_i < 1 \quad i = 1, 2, \ldots N \\ & 0.10 < r_i < 0.25 \quad i = 1, 2, \ldots N \end{aligned} \tag{1}$$

For sensor i, the position is defined by x_i and y_i, and the sensing radius is r_i. The solution uses a total of N sensors. The domain to be covered is denoted by **A**, which represents a 2 × 2 square. In the above problem description, the objective function $f(x)$ and the variables x_i, y_i, r_i are assumed to be dimensionless.

2.2 Wind farm layout

In a wind farm each turbine produces power based on the local wind speed, however each turbine also creates a downstream wake. Turbines caught in this wake will experience a lower effective wind speed and produce less power. In the wind farm layout problem the positioning of turbines in a domain is optimized such that these wake effects are minimized, resulting in a greater overall power production.

A constraint universally applied in wind farm layouts is a minimum spacing between turbines. This distance varies but is typically a multiple of the rotor diameter. This paper assumes a rotor diameter D of 40 m, and a minimum turbine spacing of 5D, or 200 m.

A popular method for solving the wind farm is to discretize the available domain into a grid. Solutions are usually represented by a binary string, where the length of the string is equal to the number of elements in the grid [39–44]. Each binary value indicates whether or not a turbine is present at the center of the associated element.

The size of the elements is chosen such that the turbine spacing constraint is automatically satisfied. This representation allows the number of turbines to vary, but limits the flexibility in turbine positioning.

Some studies use a variable-length genome, however the positioning of the turbines is still limited to a set of discrete locations [45–47]. Other algorithms consider a continuous domain for placing turbines [48–51], however the cited studies assume a fixed number of turbines. Additional considerations often include the cost of infrastructure for the wind farm [46, 47]. The turbine wake and power curve models vary among studies. Detailed reviews of the various models, additional problem considerations, and algorithms used to solve the wind farm problem are available in recent articles by Herbert-Acero et al. [52] and González et al. [53].

This paper will optimize both the number and locations of the turbines on a continuous domain. A domain size of 2000 m $\times$ 2000 m is assumed, however the placement of turbines is limited such that there is a buffer of 100 m between the turbine and domain boundary. The constraint that enforces the minimum turbine spacing is given in Eq. (3), where x_i and y_i denote the position of the i th turbine.

The turbine data and wake model used here is the same as that employed by Mosetti et al. [39] and Grady et al. [40], which is based on a wake decay model developed by Jenson [54, 55]. The environmental wind data is the same as the second case used by Grady et al., which assumes a 12 m/s wind speed occurring at an equal rate from all directions. Using these models and data the total power produced by a particular wind farm can be determined.

The objective is to minimize the cost per unit power produced. The cost model, given in Eq. (4), gives a non-dimensionalized cost/year for the wind farm. This model includes a cost reduction term based on the total number of turbines, denoted by N, and was originally proposed by Mosetti et al. [39]. The full optimization statement can be seen in Eq. (2) below.

$$\begin{aligned} &\text{Minimize } f(x) = \frac{cost}{powerproduced} \\ &\quad \text{Subject to } g(x) \leq 0 \\ &100 < x_i < 1900 \quad i = 1, 2, \ldots N \\ &100 < y_i < 1900 \quad i = 1, 2, \ldots N \end{aligned} \tag{2}$$

where

$$\begin{aligned} g(x) &= \sum_{i=1}^{N} \sum_{j=i+1}^{N} Ramp\left(200 - \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}\right) \\ Ramp(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \end{aligned} \tag{3}$$

$$cost = N\left(\frac{2}{3} + \frac{1}{3}e^{-0.00174N^2}\right) \tag{4}$$

2.3 Composite laminate stacking problem

Typical composite laminate optimization problems consider the optimization of ply orientation, thickness and/or layup sequence with the goal of minimizing weight or cost, as shown in Fig. 1.

One method of solving this is by treating the ply orientation and/or thickness as continuous variables [8, 56, 57]. However, the orientation and thickness of the plies are normally limited to several predefined values, dependent on manufacturing capabilities. Additionally, the multimodality of the laminate composite design problem hinders the ability of traditional gradient-based methods [8, 56]. Genetic algorithms have been found to be a suitable alternative.

The optimal number of plies is unlikely to be known a priori. A popular method allowing for a variable number of plies is to use a fixed-length genome but allow for empty plies [8–11]. This is one variation of the hidden-metavariable representation, which will be discussed in Sect. 3.1. Another option is to define a fixed number of layers where each layer consists of a variable number of identical plies [58].

In the sensor coverage and wind farm problems described above, the performance of a solution is independent of the order of metavariables in the genome. This is not true for the laminate stacking problem; the order of the genome affects the layup order of the plies, which affects performance. Specifically, the layup order affects the bending and twisting stiffness of the laminate.

In the current study the thickness of each ply is assumed to be fixed, and the orientation of each ply is restricted to angles corresponding to 15° increments, starting with 0°. Integer values are assigned to each possible ply orientation; a candidate laminate design is encoded as a string of integers. To ensure a balanced design, each integer is expressed as a $+\theta/-\theta$ pair of plies in the candidate solution.

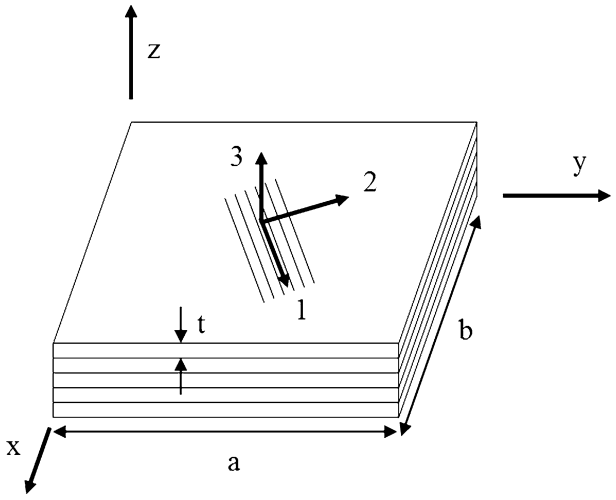

Fig. 1 Simply supported laminate. (x, y, z) denote global coordinates, (1, 2, 3) denote material coordinates

Specifically, the values {1, 2, 3, 4, 5, 6, 7} are interpreted as {$0°_2$, ±15°, ±30°, ±45°, ±60°, ±75°, $90°_2$} ply stacks in the solution. To ensure symmetry, only the top half of the laminate is encoded; the bottom half is formed by mirroring the top half over the mid-plane. For example, the laminate $[\pm 75°, \pm 30°, 0°_2]_s$ is encoded as the integer string [6 3 1]. As a result, each metavariable effectively encodes 4 plies.

Analysis of each laminate design is performed using Classical Laminate Theory [59]. The laminate is considered to be simply supported on all four edges. The global coordinate system is denoted by (x, y, z), and the material coordinates by (1, 2, 3), as shown in Fig. 1.

The relationship between the force/moment resultants and laminate strain/curvature responses are given in Eq. (5).

$$\begin{bmatrix} N \\ M \end{bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix} \begin{bmatrix} \varepsilon^0 \\ \kappa \end{bmatrix} \tag{5}$$

The vector **N** contains the normal and shear forces per unit length (N_x, N_y, $N\text{-}_{xy}$), and the vector **M** contains the bending and twisting moments per unit length (M_x, M_y, M_{xy}). Only in-plane normal and shear force loads are considered in this paper. **A**, **B**, and **D** are each 3 × 3 matrices containing the extensional, coupling, and bending stiffnesses, respectively. As our laminates in this paper are symmetric, the coupling stiffness matrix **B** will always be zero. The vector ε^0 contains the in-plane mid-surface strains (ε_x^0, ε_y^0, γ_{xy}^0) and the vector κ contains the curvatures of the laminate (κ_x, κ_y, κ_{xy}). The material properties and loading conditions used in this paper are shown in Table 1.

The ability of a laminate to support a load is measured by its load factor λ. For a baseline compressive in-plane loading (N_x, N_y, N_{xy}), the load factor is the largest scaling factor that can be applied to the loads (λN_x, λN_y, λN_{xy}) without the laminate failing. Therefore a load factor greater than 1 indicates the laminate can support the baseline load. A laminate can fail either through compressive material failure or buckling. Each mode of failure has its own load factor, λ_b for failure by buckling

Table 1 Composite laminate plate properties and problem setup

Plate dimensions		Material properties	
a	508 mm	E_{11}	127.5 GPa
b	127 mm	E_{22}	13.0 GPa
t (per ply)	0.127 mm	G_{12}	6.41 GPa
		v_{12}	0.3
Baseline loads		**Material strengths**	
N_x	900 kN/m	ε_1^a	0.008
N_y	450 kN/m	ε_2^a	0.029
N_{xy}	0	γ_{12}^a	0.015
M_x	0		
M_y	0		
M_{xy}	0		

and λ_m for material failure. The load factor for the laminate is the minimum of these two values.

To determine λ_m the strain response determined by Eq. (5) is used to find the in-plane material strains (ε_1,ε_2, γ_{12}) for each ply. The material load factor λ_m is the smallest value that results in at least one ply failing in at least one direction. The allowable strains ($\varepsilon_1^a, \varepsilon_2^a, \gamma_{12}^a$) are listed in Table 1. The material properties, strengths, and plate dimensions are the same as those used by Le Riche and Haftka [10].

The buckling load factor λ_b is approximated by Eq. (6) [60, 61]. When the laminate buckles it will do so in m and n half-waves in the x and y directions, respectively. The buckling load factor is determined by the combination of m and n half-waves that produces the minimum factor.

$$\lambda_b = \min_{m,n}\left(\pi^2 \frac{D_{xx}\left(\frac{m}{a}\right)^4 + 2\left(D_{xy}+2D_{ss}\right)\left(\frac{m}{a}\right)^2\left(\frac{n}{b}\right)^2 + D_{yy}\left(\frac{n}{b}\right)^4}{N_x\left(\frac{m}{a}\right)^2 + N_y\left(\frac{n}{b}\right)^2 + N_{xy}\left(\frac{mn}{ab}\right)}\right) \tag{6}$$

This paper seeks to find the thinnest composite laminate that provides a load factor of at least 3. The optimization statement is given in Eq. (7), where p_i is an integer value of the ith metavariable that will be expressed as a set of plies with a specific orientation as described earlier.

$$\begin{aligned} &\text{Maximize} f(x) = \lambda \\ &\text{Subject to } n - n_{\min} = 0 \\ &p_i \in \{1,2,3,4,5,6,7\} \quad i = 1,2,\ldots n \end{aligned} \tag{7}$$

The laminate will have a total of 4n plies. The value n_{min} is the minimum number of metavariables required to achieve $\lambda \geq 3$ as identified by the algorithm. The interaction of this constraint with the selection operator is explained in further detail in Sect. 3.4. Problem data is given in Table 1.

3 Metameric genetic algorithms

Metameric genetic algorithms (MGA) are characterized by their genome, consisting of a series of metavariables. An example is shown in Fig. 2. Each metavariable contains the design variables needed to fully define one component (a sensor, wind turbine, or ply) in the solution. Genomes cannot contain partially defined metavariables.

Fig. 2 Each genome used in MGA is comprised of a series of metavariables (M1, M2,…MN). Each metavariable contains its own set of design variables

In this paper the MGA genomes will typically be variable-length. An alternate representation is also presented here, the hidden-metavariable representation. This uses a fixed-length genome but is still considered to be a variable-length algorithm. This is achieved by allowing some metavariables to remain unexpressed in the evaluated solution. This is detailed further in Sect. 3.1. The metameric variable-length genome, along with several potential recombination operators, is described in Sect. 3.2. Both representations will use the same mutation and selection operators, discussed in Sects. 3.3 and 3.4, respectively.

The results of these algorithms will be compared to a fixed-length GA that assumes the optimal number of metavariables is known a priori. The parameters for the fixed-length and variable-length algorithms can be found in Table 2. The same parameters are used for all problems with the exception of the initial number of metavariables.

3.1 Hidden-metavariable representation

The hidden-metavariable representation, as it will be referred to in this paper, is one of the most common approaches in the literature for solving metameric variable-length problems. Examples of its use include application to sensor coverage [7], composite laminate [8–11], multilayer optical coating [12], space trajectory [13], clustering [14–17], and electronic circuit [18] problems. The commonly used representation in wind farm problems, where the genome indicates whether or not a turbine is present at a set of predetermined locations, closely resembles the hidden-metavariable representation. Despite its frequent use there is not a commonly agreed upon methodology or terminology. Observed terms include "hidden gene" [13], "active/inactive gene" [18], "hierarchical chromosome" [7], "filtering mask" [17], "activation threshold" [15, 16], "activation mask" [18], "control gene" [7, 16], and "don't care symbol" [14].

Every approach fixes the length of the genome, but introduces some sort of mechanism that determines whether or not each metavariable is expressed in the evaluated solution. This allows the genome to have a *phenotypic* (or *expressed*, or *effective*) length of anywhere from zero metavariables to an upper bound determined

Table 2 Parameters used by the genetic algorithms

Recombination rate	0.8
Mutation rate	(Total # variables)$^{-1}$
Mutation magnitude	0.05
Component insertion rate[a]	0.05
Component deletion rate[a]	0.05
Initial # of metavariables[b]	
Sensor coverage	10–50
Windfarm layout	10–50
Laminate stacking	10–20
Population size	20
Number of generations	2500

[a] The fixed-length genetic algorithm has a metavariable insertion and deletion rate of 0

[b] The number of metavariables in the fixed-length genetic algorithm will be defined a priori

by the length of the genome. As such this is considered to be a variable-length algorithm, despite the fixed-length genome.

The primary benefit of this is that by using a fixed-length genome the traditional mutation and recombination operators can be used. In this paper a standard 2-point crossover is used. A similar approach is used by structured GA [62], solutions are represented by a hierarchical genomic structure where high level genes are able to activate or deactivate lower level genes.

One common implementation of the hidden-metavariable representation is used when a metavariable already includes a discrete or integer variable. An additional 'empty' value is added to the list of potential values, any metavariable that contains this value would be removed from the expressed solution [8–12]. A second option is to extend a variable's value beyond the valid bounds of its domain; metavariables that contain invalid values for these variables would remain unexpressed.

Another option is to introduce additional 'flag' variables for each metavariable. Some implementations use a binary flag to control whether or not a metavariable is expressed [7, 17, 18], while others use a continuous flag with a predefined activation threshold [15, 16]. This method is used in this paper as it is easily implementable and non-problem-specific. An additional binary flag is added to each metavariable. Metavariables with a flag value of '0' are not expressed in the evaluated solution, as illustrated in Fig. 3. A genome length of 100 metavariables will be used for the sensor coverage and wind farm problems, and 40 metavariables for the laminate stacking problem. These form the upper bound solution size and are well over the expected optimal lengths.

The chosen length of the genome can affect algorithm performance. If the genome length is shorter than the optimal solution length it will be impossible to reach that optimal length. Using a genome length that is exceedingly large may also affect the mutation and recombination operators. Depending on the implementation of the mutation operator using longer genomes could result in more variance in the resulting genome length due to changes to the flag variables. In this study the mutation operator is implemented in such a way that it is insensitive to genome length, as discussed in Sect. 3.3. A longer genome would also result in more potential crossover points for the recombination operator. Depending on the distribution of expressed metavariables in the genotype this may limit the effectiveness of the operator. If many metavariables were located in a small portion of the genotype it would be very difficult to separate these metavariables during recombination.

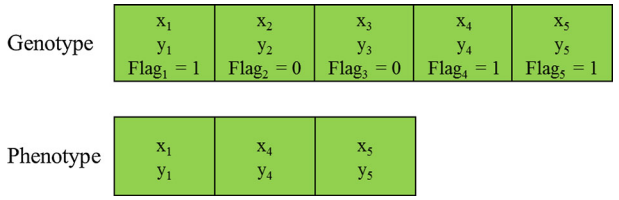

Fig. 3 Illustration of hidden-metavariable representation. An additional flag variable is included in each metavariable. The value of the flag determines if the metavariable is included in the evaluated phenotype

3.2 Metameric variable-length genomes

A metameric variable-length genome is one whose length can vary, but can contain only fully defined metavariables. Metavariables can be added to or removed from the genome through the recombination or mutation operators. This means the traditional genetic algorithm operators cannot be applied.

The modifications to the mutation operator are straightforward and described in Sect. 3.3. It is less clear how to modify the recombination operator to function with metameric variable-length genomes. Several existing, and one newly proposed, recombination operators are described in the following subsections.

3.2.1 Cut-and-splice recombination

The cut-and-splice recombination operator is similar to the 2-point crossover with the exception that the crossover points in the two parents do not have to match. As a result, the number of metavariables in each child may be different than the number of metavariables in either parent, as illustrated in Fig. 4.

3.2.2 Spatial recombination

Spatial recombination, demonstrated in Fig. 5, operates in the phenotypic space of the solution. In the sensor coverage wind farm problems, this involves splitting the square domain into two parts using a random dividing line. The set of metavariables that lie on either side of this line are exchanged to form children. When used for the laminate stacking problem, this operator is more similar to a one-point crossover. The dividing line is chosen at a random ply interface in the shorter parent. All plies between the dividing line and mid-plane of the ply are then exchanged.

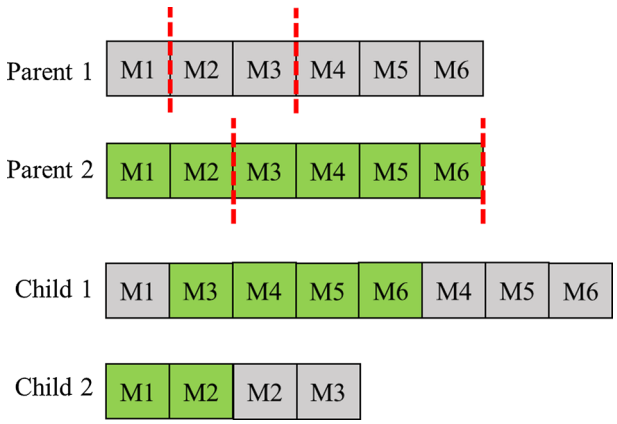

Fig. 4 Cut-and-splice recombination. Two crossover points are used, however their positions do not have to be the same in both parents

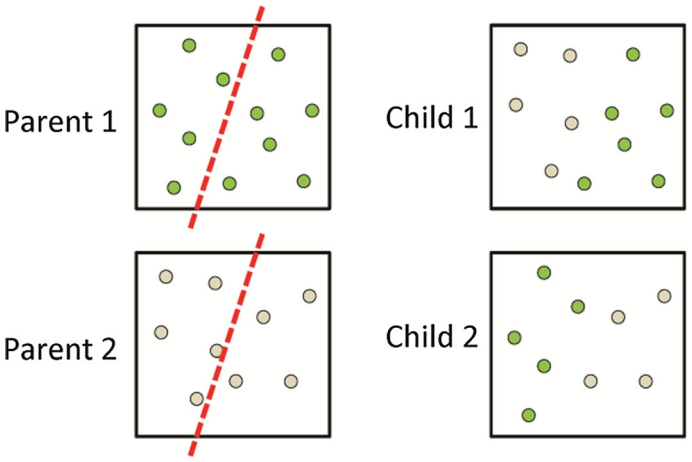

Fig. 5 Spatial recombination. When used for sensor placement and wind farm layout, the domain is split into two parts by a randomly-placed and randomly-oriented *line*. Components, or metavariables, on opposite sides of the *line* are exchanged to form children

Since this recombination operator preserves the spatial relationships of a large number of metavariables, with the disruption being limited to the dividing line, it is considered less disruptive than an N-point crossover [63].

3.2.3 Synapsing variable length crossover (SVLC)

A method proposed by Hutt and Warwick [64], the synapsing variable-length crossover (SVLC), attempts to preserve common subsequences in the genomes of both parents. Substrings of genes, or in this case metavariables, common to the parents are identified and then glued, or 'synapsed', together. Crossover may then occur only within the synapsed regions. This preserves the common subsequences in both parents while exchanging the differences between the synapsed regions. An illustration of SVLC is shown in Fig. 6. This recombination preserves the relative order of components in the genome. For this reason it is used only for the laminate stacking problem.

3.2.4 Similar-metavariable recombination

The similar-metavariable recombination proposed herein was inspired by SVLC and its effort to preserve common substrings in the genomes of both parents. However, rather than identifying common substrings, this operator identifies similar metavariables between the parents. Unlike the SVLC operator, this one does not preserve the relative ordering of the genomes. To accomplish this, a measure of dissimilarity between two metavariables, shown in Eq. 8, is used. M_{1i} and M_{2i} represent the *i*th design variable of two metavariables, one from each parent, where each metavariable contains n design variables. L_i represents the length of the domain of the *i*th design variable. A dissimilarity of 0 would indicate two identical

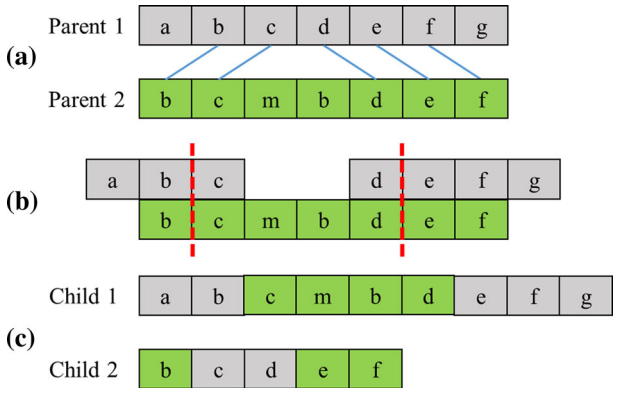

Fig. 6 SVLC. **a** Common subsequences identified between parents, shown as *thin lines*. **b** Common subsequences are synapsed together and crossover points are chosen within synapsed regions. **c** Resulting child chromosomes

metavariables, while a value of 1 indicates two metavariables that are as dissimilar as possible for the given variable bounds.

$$Dissimilarity = \frac{1}{n}\sum_{i=1}^{n}\frac{|M_{1i} - M_{2i}|}{L_i} \tag{8}$$

The dissimilarity of every metavariable pair between the two parents is measured. Each metavariable is then 'linked' with its most similar, or least dissimilar, counterpart in the other parent. This is illustrated in Fig. 7. Oftentimes two metavariables will form a mutual linkage, however this is not always the case. A metavariable in parent 1 may link itself to a metavariable in parent 2, but there may exist a better match for that metavariable. Examples of this are shown in Fig. 7. When this happens all metavariables in a parent that are connected through a pathway of linkages will form a subset, or building block, of metavariables.

The parent genomes will segment themselves into paired subsets of one or more metavariables based on the linkages that have formed. The number of subsets in each parent will match, but the number of metavariables in each subset do not have to match. Some of these subsets are then randomly exchanged between the parents to form child solutions. A random integer value N, chosen between 1 and (total number of subsets—1), determines the number of subsets that will be exchanged.

3.3 Mutation

There are two types of mutation: design variable mutation and metavariable insertion or deletion. The rate of design variable mutation is inversely proportional to the total number of design variables contained in the genome. On average only one design variable will be mutated per operator call. When using the hidden-metavariable representation the unexpressed metavariables are not included when determining the mutation rate, and the 'flag' variable is not subject to design variable mutation. The magnitude of the mutation is determined by a random

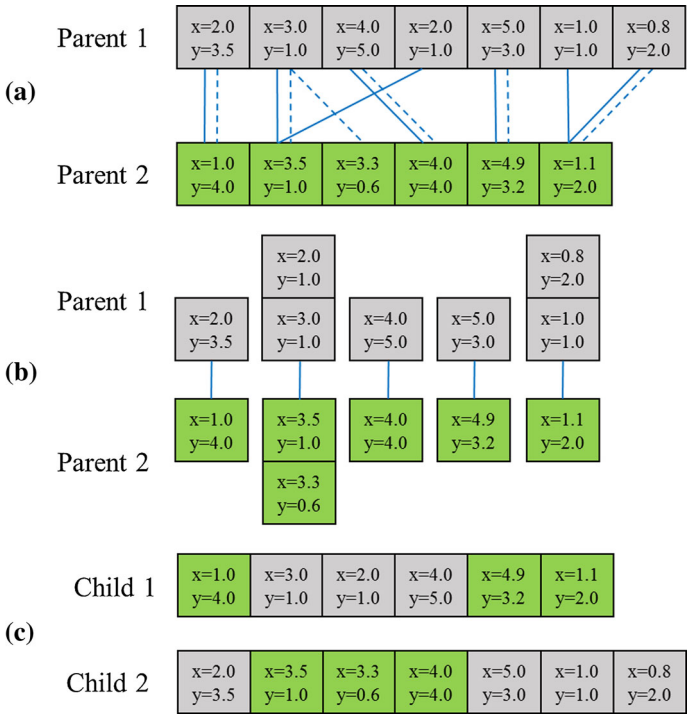

Fig. 7 Similar-metavariable recombination. **a** Each metavariable identifies its most similar counterpart in the other parent and forms a link. *Solid lines* show linkages formed by the metavariables in parent 1, and *dashed lines* for those in parent 2. **b** All metavariables connected by these linkages are grouped together to form a subset. **c** A random number of these groups are exchanged to form child solutions

number from a normal distribution with a standard deviation equal to 5 % of the domain length of the design variable being mutated.

The metavariable insertion mutation will insert a randomly generated metavariable at a random position in the genome. The metavariable deletion mutation removes a randomly chosen metavariable from the genome. For the hidden-metavariable representation the insertion operation can only activate an unexpressed metavariable, in which case the design variables will also be replaced with new random values. The deletion operation will set the 'flag' variable of an expressed metavariable to 'off'. The insertion and deletion operations are not used for the fixed-length GA.

3.4 Selection and the helper objective

Selection is based on NSGA-II and uses multi-objective non-dominated sorting [65]. The standard crowding and niching methods are not employed here. Instead, a helper objective is used to maintain diversity based on solution length. Without this diversity it is more likely that the population will converge to a non-optimal solution length. This is because adding or removing metavariables tends to be destructive to

solution fitness. In the sensor coverage problem removing a sensor is likely to leave some portion of the domain uncovered, or adding a turbine in the wind farm problem is likely to place existing turbines in its wake. Even if the new solution length is more optimal, the destructive nature of these changes makes it unlikely for the solution to survive if the helper objective is not employed.

The helper objective either minimizes or maximizes the number of metavariables, as decided by the user. This choice requires some intuition regarding the problem and depends on whether adding or removing metavariables is expected to be more destructive. In the sensor coverage problem, exposing a portion of the domain is more costly than adding a redundant sensor. Based on this, the helper objective is chosen to minimize the number of metavariables. Using similar reasoning, it was decided to maximize the number of metavariables in the wind farm problem, and minimize the number of metavariables in the laminate stacking problem.

The performance of an algorithm depends heavily on its efficient use of available computational resources. When using multi-objective non-dominated sorting, the Pareto set, and with it the range of the number of metavariables in solutions, can grow very large. Populations could arise where every individual contains a different number of metavariables; this spreads the computational resources too thin and is detrimental to the overall performance.

To avoid this, a moving constraint window is used. First, the current best solution is chosen based on the objective function and constraints of the problem. The number of metavariables in this solution, also referred to as the "baseline" length, forms the center of a constraint window. The constraints and are formed such that only solutions with lengths within ± 2 metavariables of the current best solution are considered feasible. This allows solutions that undergo destructive length changing operations to survive such that they might eventually form a new best solution. As the algorithm progresses, the constraint window shifts so that it is always centered on the number of metavariables in the current best solution.

The methodology for the laminate stacking problem is slightly different. This problem seeks to find the thinnest laminate possible with a load factor greater than 3. Rather than placing a constraint on the load factor the selection operator will be used to select individuals that meet or are close to meeting this criterion. To do this, the constraint window is centered on the length of the shortest solution that has a load factor of at least 3. Since each metavariable encodes for 4 plies, the constraint window includes solutions within ± 8 plies of the baseline solution. This allows solutions that use fewer plies to survive and possibly improve, through recombination and mutation, such that they eventually meet the minimum load factor. If the constraint were employed in a traditional manner, this would not be possible, and any solution that did not initially meet the load factor requirement would most likely not survive the selection operator.

A full list of objectives and constraints for each problem can be found in Table 3. For comparative purposes, a single-objective MGA—one that does not use the helper objective or the constraint window—is also used and results are compared with other methods.

Table 3 Objectives and constraints used for each optimization problem, including the helper objective and window constraints used by the selection operator

Optimization problem	Objectives	Constraints
Sensor coverage	Minimize objective function value	Number of sensors $\leq$ baseline + 2
	Minimize number of sensors	Number of sensors $\geq$ baseline − 2
Wind farm	Minimize cost per unit power	Proximity violations ≤ 0
	Maximize number of turbines	Number of turbines $\leq$ baseline + 2
		Number of turbines $\geq$ baseline + 2
Laminate stacking	Maximize load factor	Number of plies $\leq$ baseline + 8
	Minimize number of plies	Number of plies $\geq$ baseline − 8

4 Results

This section presents the results for each optimization problem test case. Each algorithm was run for 100 trials. The performance of each algorithm is measured as the mean of the fitness of the best individual from each trial at a given generation.

4.1 Sensor coverage

The sensor coverage problem minimizes an objective function that combines both the cost of the sensors and a penalty that scales with the uncovered portion of the field. A sample solution with a very good fitness is shown in Fig. 8. The average fitness and solution length of each tested algorithm over the 100 studies performed are given in Table 4.

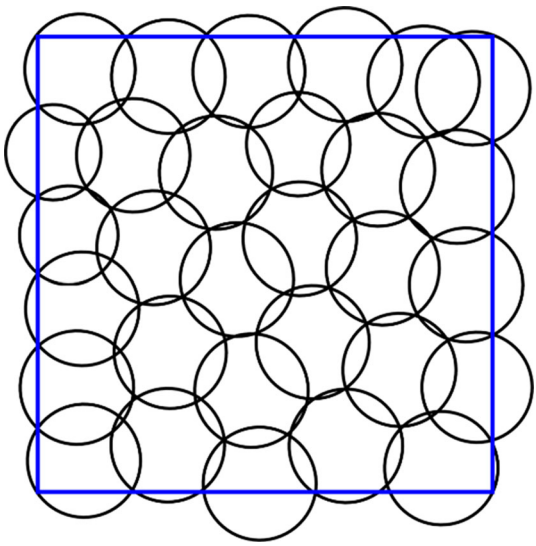

Fig. 8 A sensor coverage sample solution using 30 sensors to obtain a fitness of 48.68

Table 4 The mean and standard deviation of the best fitness achieved over 100 studies by each algorithm for the sensor coverage problem

	Evaluations		
Method/recombination operator	5000	20,000	50,000
Cut and splice	137.29 ± 10.70	86.86 ± 4.81	67.33 ± 3.30
	70.94 ± 6.31 Sensors	55.57 ± 3.97 Sensors	44.28 ± 3.02 Sensors
Hidden-metavariable	74.82 ± 3.63	55.80 ± 2.13	50.55 ± 1.03
	46.33 ± 3.01 Sensors	35.56 ± 1.95 Sensors	30.92 ± 1.11 Sensors
Similar-metavariable	74.28 ± 3.69	55.90 ± 2.25	50.27 ± 1.02
	46.58 ± 3.41 Sensors	35.79 ± 2.07 Sensors	30.76 ± 1.19 Sensors
Spatial	73.98 ± 3.58	55.23 ± 2.19	50.03 ± 0.98
	46.41 ± 3.10 Sensors	35.23 ± 2.09 Sensors	30.54 ± 1.10 Sensors
Spatial—single objective	72.74 ± 3.31	66.85 ± 2.86	66.26 ± 2.78
	49.47 ± 2.95 Sensors	47.65 ± 2.67 Sensors	47.58 ± 2.64 Sensors
Mutation only	90.51 ± 5.86	60.62 ± 2.36	51.61 ± 1.22
	52.84 ± 3.91 Sensors	39.64 ± 2.19 Sensors	31.71 ± 1.19 Sensors
Fixed-length (27 sensors)	125.80 ± 17.85	60.85 ± 5.39	56.39 ± 4.13
Fixed-length (28 sensors)	120.49 ± 16.01	58.96 ± 5.27	54.21 ± 3.65
Fixed-length (29 sensors)	111.39 ± 12.08	55.60 ± 3.90	52.36 ± 2.75
Fixed-length (30 sensors)	106.39 ± 14.99	54.77 ± 3.39	52.17 ± 1.98
Fixed-length (31 sensors)	99.15 ± 13.12	53.54 ± 2.75	51.80 ± 2.04
Fixed-length (32 sensors)	97.92 ± 14.23	53.96 ± 2.45	52.30 ± 1.64
Fixed-length (33 sensors)	94.03 ± 13.00	54.31 ± 2.36	52.95 ± 1.63

The average solution length is also given for variable-length algorithms

Figure 9 shows a plot of the average number of sensors used by each MGA. Each method has a tendency to use many sensors in early generations; this is a result of the penalty for uncovered regions being very high relative to the cost of individual sensors. As the algorithms progress, the solutions are refined to use a more optimal number. Table 5 shows the best known fitness for various numbers of sensors, and while these are not globally optimal solutions, they give insight toward the most optimal solution lengths. The best known solutions use 28–30 sensors, and there is a decline in fitness outside of this range. Several fixed-length algorithms were performed using the optimal or near-optimal number of sensors. At 20,000 evaluations, some of the better fixed-length algorithms have better solutions than the variable-length algorithms, which have still not reached optimal solution lengths. By 50,000 evaluations, the variable-length algorithms have outperformed the fixed-length algorithms.

The MGA using the spatial recombination operator produces the best fitness values on average, followed by the similar-metavariable recombination operator and the hidden-metavariable representation. The difference in performance between

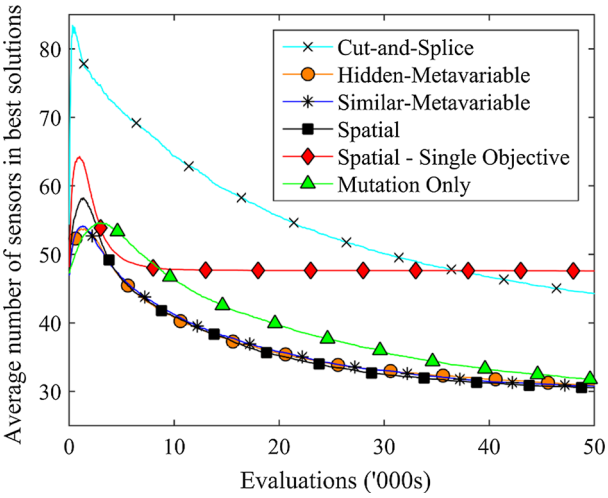

Fig. 9 Average number of sensors in best solutions for the MGAs, averaged over 100 studies at each iteration

Table 5 Best known fitness using different numbers of sensors

Number of sensors	Best fitness achieved
27	48.96
28	48.24
29	**47.63**
30	48.03
31	49.16
32	50.08

each of these three operators and all other algorithms is statistically significant at 50,000 evaluations ($p < 0.0001$). The difference between the spatial and similar-metavariable operators is not quite statistically significant ($p = 0.1008$), but the difference between the spatial operator and hidden-metavariable representation is statistically significant ($p = 0.0004$).

4.2 Wind farm

The wind farm problem seeks to minimize the cost per unit power generated, with results given in Table 6. A sample solution is shown in Fig. 10. The optimal solutions commonly position turbines along the outer edge of the domain; this helps maximize the distance between turbines and minimize the effective wake. The average solution lengths for the MGAs are shown in Fig. 11. The algorithms prefer to use few turbines in early generations due to the constraint that enforces a minimum distance between turbines, which is more easily satisfied with fewer turbines. By the end, the algorithms use approximately 40 turbines on average. The

Table 6 The mean and standard deviation of the best fitness achieved over 100 studies by each algorithm for the wind farm problem

	Evaluations		
Method/recombination operator	5000	20,000	50,000
Cut and splice	1.6723 ± 0.0264	1.5535 ± 0.0062	1.5211 ± 0.0048
	25.69 ± 2.03 Turbines	36.91 ± 1.02 Turbines	38.99 ± 0.83 Turbines
Hidden-metavariable	1.5386 ± 0.0063	1.5020 ± 0.0041	1.4917 ± 0.0042
	38.52 ± 0.92 Turbines	39.81 ± 0.66 Turbines	40.09 ± 0.55 Turbines
Similar-metavariable	1.5336 ± 0.0050	1.5010 ± 0.0040	1.4920 ± 0.0037
	38.99 ± 0.92 Turbines	39.76 ± 0.57 Turbines	40.10 ± 0.54 Turbines
Spatial recombination	1.5347 ± 0.0057	1.5011 ± 0.0040	1.4919 ± 0.0040
	38.80 ± 0.91 Turbines	39.95 ± 0.54 Turbines	40.17 ± 0.55 Turbines
Spatial recombination—single-objective	1.5314 ± 0.0055	1.5016 ± 0.0041	1.4928 ± 0.0041
	37.79 ± 0.81 Turbines	38.85 ± 0.54 Turbines	39.34 ± 0.50 Turbines
Mutation only	1.5548 ± 0.0058	1.5081 ± 0.0035	1.4936 ± 0.0040
	36.92 ± 1.00 Turbines	39.43 ± 0.61 Turbines	40.00 ± 0.57 Turbines
Fixed-length (37 turbines)	1.5435 ± 0.0085	1.5131 ± 0.0060	1.5040 ± 0.0049
Fixed-length (38 turbines)	1.5434 ± 0.0116	1.5116 ± 0.0057	1.5028 ± 0.0054
Fixed-length (39 turbines)	1.5426 ± 0.0091	1.5099 ± 0.0064	1.5003 ± 0.0052
Fixed-length (40 turbines)	1.5455 ± 0.0098	1.5097 ± 0.0056	1.4999 ± 0.0053
Fixed-length (41 turbines)	1.5468 ± 0.0114	1.5105 ± 0.0057	1.5000 ± 0.0048
Fixed-length (42 turbines)	1.5505 ± 0.0100	1.5125 ± 0.0053	1.5013 ± 0.0054
Fixed-length (43 turbines)	1.5538 ± 0.0135	1.5128 ± 0.0059	1.5024 ± 0.0056

The average solution length is also given for variable-length algorithms

best known fitness at various solution lengths, based on the results from this study, are given in Table 7.

The hidden-metavariable representation produces the best results for this problem, followed by the spatial recombination operator and the similar-metavariable operator. However, the performance difference between these three algorithms is minimal and not statistically significant. The difference between the hidden-metavariable representation and single-objective MGA is not quite statistically significant at 50,000 evaluations ($p = 0.0729$), but the difference between the hidden-metavariable representation and mutation only algorithm is statistically significant ($p = 0.0027$). All variable-length algorithms, with the exception of the

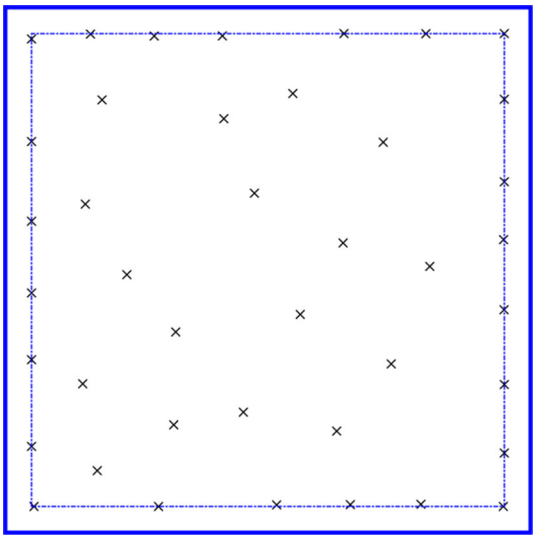

Fig. 10 A sample windfarm layout, using 41 turbines to achieve a fitness of 1.486 units cost per MW year. The *dashed line* represents the placement bounds of the turbines such that they are at least 100 m from the domain boundary

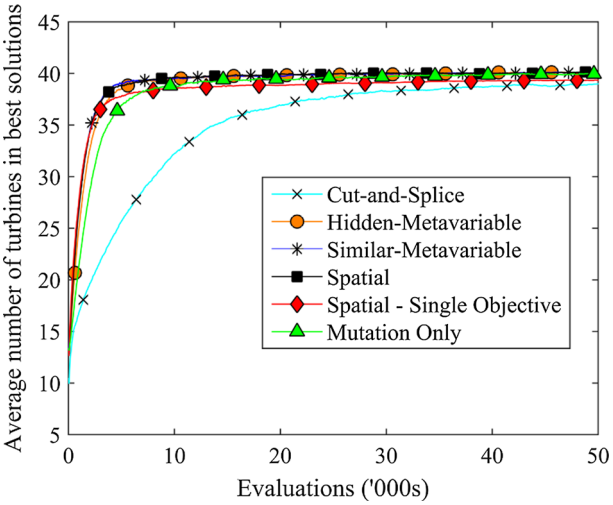

Fig. 11 Average number of turbines in best solutions for the MGAs, averaged over 100 studies at each iteration

cut-and-splice operator, outperform the fixed-length algorithms in a statistically significant manner ($p < 0.0001$).

4.3 Laminate stacking

The results for the laminate stacking problem are given in Table 8. The optimal solution length was found to be 14 metavariables (56 plies), as this is the minimum

Table 7 Best known fitness using different numbers of turbines

Number of turbines	Best fitness achieved
37	1.4948
38	1.4904
39	**1.4820**
40	1.4831
41	1.4830
42	1.4842
43	1.4861

Table 8 The mean and standard deviation of the best fitness achieved over 100 studies by each algorithm for the laminate stacking problem

	Evaluations		
Method/recombination operator	500	5000	50,000
Cut and splice	3.1636 ± 0.2056	3.2496 ± 0.0056	3.2578 ± 0.0011
	56.56 ± 1.39 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Hidden-metavariable	3.1553 ± 0.0385	3.2564 ± 0.0023	3.2585 ± 0.0002
	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Synapsing variable length crossover	3.1428 ± 0.0412	3.2561 ± 0.0023	3.2583 ± 0.0006
	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Spatial	3.1513 ± 0.0599	3.2528 ± 0.0045	3.2581 ± 0.0008
	56.04 ± 0.40 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Spatial—single objective	3.1798 ± 0.1270	3.2575 ± 0.0028	3.2586 ± 0.0002
	56.20 ± 0.88 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Mutation only	3.1523 ± 0.1414	3.2526 ± 0.0046	3.2581 ± 0.0010
	56.28 ± 1.03 Plies	56.00 ± 0.00 Plies	56.00 ± 0.00 Plies
Fixed-length (14 plies)	3.2094 ± 0.0212	3.2574 ± 0.0034	3.2585 ± 0.0015

The average solution length is also given for variable-length algorithms

number of plies required for a load factor of 3 or greater. The average solution lengths of the MGAs are shown in Fig. 12; by 1000 evaluations all algorithms have converged to the optimal length. No solutions were found that used fewer than 56 plies while achieving the required load factor. The best ten known stacking sequences are given in Table 9.

All algorithms produced near optimal results, however the single-objective MGA appears to produce the best results most consistently. At 5000 and 50,000 evaluations, the performance differences between the single-objective algorithm and all other algorithms, with the exception of the fixed-length algorithm, are statistically significant ($p < 0.001$). This is a result of the helper objective used in the other MGAs spreading the computational effort too thin. In this problem there is a well-defined optimal length of 14 metavariables, however algorithms that use the

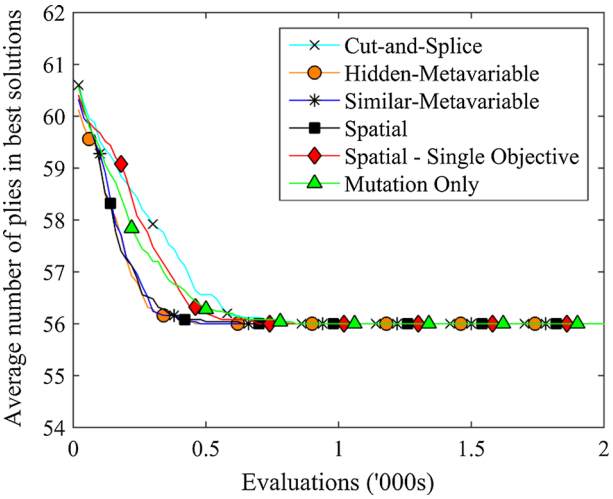

Fig. 12 Average number of plies in best solutions for the MGAs, averaged over 100 studies at each iteration. By approximately 1000 function evaluations all algorithms have converged to the optimal length

Table 9 The 10 best laminate stacking sequences determined over all trials

Stacking sequence	Buckling load factor	Material failure load factor
$[5\ 5\ 5\ 6\ 5\ 6\ 5\ 5\ 4\ 4\ 3\ 3\ 3\ 3]_s$	3.2589	3.3466
$[5\ 6\ 5\ 5\ 5\ 5\ 5\ 5\ 5\ 4\ 3\ 1\ 1\ 1]_s$	3.2588	3.2811
$[6\ 5\ 5\ 5\ 5\ 5\ 5\ 5\ 4\ 4\ 3\ 3\ 3\ 3]_s$	3.2587	3.3712
$[5\ 5\ 5\ 6\ 5\ 5\ 5\ 5\ 6\ 5\ 2\ 2\ 2\ 2]_s$	3.2584	3.2805
$[5\ 5\ 5\ 5\ 6\ 5\ 5\ 6\ 5\ 5\ 2\ 2\ 2\ 2]_s$	3.2582	3.2805
$[5\ 5\ 6\ 5\ 5\ 5\ 5\ 5\ 5\ 5\ 2\ 2\ 2\ 2]_s$	3.2581	3.3144
$[5\ 5\ 6\ 5\ 5\ 5\ 6\ 5\ 4\ 4\ 3\ 3\ 3\ 3]_s$	3.2577	3.3466
$[5\ 5\ 5\ 5\ 6\ 5\ 6\ 6\ 4\ 4\ 3\ 3\ 3\ 3]_s$	3.2576	3.3177
$[5\ 5\ 5\ 6\ 5\ 5\ 5\ 5\ 5\ 7\ 2\ 2\ 2\ 2]_s$	3.2575	3.2786
$[5\ 5\ 5\ 6\ 5\ 5\ 5\ 5\ 5\ 6\ 2\ 2\ 2\ 2]_s$	3.2574	3.2805

The integers {1, 2, 3, 4, 5, 6, 7} represent stacks of $\{0°_2, \pm15°, \pm30°, \pm45°, \pm60°, \pm75°, 90°_2\}$ respectively. The load factor is determined by the buckling load factor in all sequences shown

helper objective will have populations with a roughly even distribution of solutions with 12–16 metavariables. In this case it would be expected roughly 80 % of the children are of lengths other than 14 metavariables and will not bring the algorithm closer to more optimal solutions. In contrast, the single-objective and fixed-length algorithms will converge to populations containing only optimal-length solutions, giving them many more opportunities to obtain a more optimal solution.

5 Discussion

In the sensor coverage and wind farm problems the MGAs were found to outperform the fixed-length algorithms by a significant margin. The exception to this is the MGA using the cut-and-splice operator, which fails to reach optimal solutions in both problems, and the single-objective MGA, which fails to reach optimal solutions in the sensor coverage problem. The algorithm using the spatial recombination operator performed best for the sensor coverage problem. In the wind farm problem, the spatial operator, similar-metavariable operator, and hidden-metavariable representation produced the best results without significant performance differences between them.

In the laminate stacking problem all algorithms consistently produced optimal, or near optimal, solutions. The single-objective MGA and the fixed-length algorithms had a marginally better, but statistically significant, performance difference compared to the other algorithms. This problem is much simpler than the other two, requiring only 14 integer variables compared to 80–90 real-valued variables. Additionally, this problem has a well-defined optimal length, whereas the other problems had a range of solution lengths that produced near-optimal solutions.

The remainder of the discussion focuses on three important observations for solving metameric variable-length problems. Section 5.1 discusses why the MGAs frequently outperform the fixed-length algorithms, even if the optimal number of metavariables is known a priori. Section 5.2 discusses the importance of the helper objective, and why some algorithms may fail to find optimal solutions without it. Finally, Sect. 5.3 discusses the performance differences observed between the various MGA representations and recombination operators.

5.1 Fixed-length versus variable-length algorithms

In general, the MGAs produce results on par with, and frequently better than, the fixed-length algorithms even when the optimal solution length is known a priori. This is somewhat counter-intuitive since using a variable-length algorithm makes the search space significantly more complex compared to that for a fixed-length algorithm. However, there are two reasons that might explain the difference in performance.

First, the changing dimensionality of the search space provides new pathways to the optimal solutions. In each problem there is a clear trend of variable-length algorithms favoring solution lengths that provide the quickest increase in fitness or feasibility in the early generations. For example, in the sensor coverage problem there is a clear tendency to use many sensors in early generations, presumably to achieve a near complete coverage of the domain. In the wind farm problem very few turbines are used in the early generations, which is most likely a result of minimum turbine spacing constraint.

Once the variable-length algorithms have found these “simple” solutions, whether they use many or few metavariables, it becomes relatively easier for the search algorithm to move towards optimal solution lengths. In the sensor coverage

problem this means adjusting the position of existing sensors and removing extraneous ones. The length changes incrementally, an algorithm will not move from using 50 sensors to 30 sensors in a single step, but rather it will progress through many intermediate solution lengths. An optimal solution that uses 30 sensors would have most likely evolved from a near optimal solution that used 31 sensors, which itself came from a solution using 32 sensors. This is in contrast to the fixed-length algorithm which uses 30 sensors always and begins from a random location in the search space. It is very likely that these incremental changes in length are more likely to produce optimal solutions in comparison to starting at the assumed optimal length with a randomly generated solution.

This mechanism is notably used in the NeuroEvolution of Augmenting Topologies (NEAT) algorithm developed by Stanley and Miikkulainen [66]. Used to evolve neural networks, NEAT begins with a population of small, simple networks which are more easily optimized compared to larger networks. Over time new nodes and connections will be added to the already-functioning networks, allowing for more complex networks to emerge. This process allows the more complex networks to start in a high quality area of the search space, compared to starting from a random location, making it much more likely that a functioning solution will emerge. Stanley uses the biological term complexification to describe this process.

The second reason for the superior performance of variable-length algorithms is due to the diversity they provide. The populations in each algorithm will converge toward a general solution concept, or layout, examples of which are shown in Figs. 8 and 10. There are an extremely large number of high-quality layouts possible, even with a fixed number of metavariables. With the given mutation and recombination operators small adjustments may occur, but large changes to the general solution layout are extremely unlikely to occur in later generations. It is entirely possible that the fixed-length algorithms coverage toward a general layout that is not optimal for their given length. For example, an algorithm that assumes 30 sensors may converge toward a layout that is better suited for 28 or 29 sensors. If a variable-length algorithm converged toward this same layout it would continue to explore the shorter solution lengths, while the fixed-length algorithm must remain at 30 sensors. Additionally, when using a variable-length algorithm with the helper objective, a single study will produce high quality solutions at several different lengths, providing additional choice to the user.

5.2 Benefit of the helper objective

The helper objective is essential in order for the MGAs to solve the sensor coverage problem. Without it, as observed in Fig. 9, the single-objective algorithm converges to a non-optimal solution length and stagnates. Removing sensors, even if it results in a more optimal solution length, is very destructive to the fitness of a solution. Without the helper objective these solutions are unlikely to survive the selection operator, and the algorithm is unable to effectively explore the design space. With the helper objective such solutions are able to survive and refine themselves, and the algorithm is able to work its way toward the optimal solution length.

In the wind farm and laminate stacking problems the performance of the single-objective MGA is similar to that of the MGAs using the helper objective. The laminate stacking problem is simple enough that the algorithms are capable of reaching optimal solution lengths regardless of whether or not the helper objective is used. In the wind farm problem this is likely due to two reasons. First, it is possible that adding or removing a wind turbine is not particularly destructive to the solution's fitness. If this is the case, then the helper objective is not necessary to ensure such solutions survive the selection operator. Second, as evidenced in Table 7, there appears to be a large range of solution lengths that result in near-optimal fitness values. This makes it easier to achieve near-optimal fitness values, as there is not a specific solution length that needs to be reached. In fact, Table 6 shows that the single-objective MGA, on average, uses 1 fewer turbine when compared to MGAs that use the helper objective.

Even in problems where the helper objective is not essential, such as the wind farm or laminate stacking problems, it is shown to have similar, or better, performance than the single-objective algorithm. For this reason it is advised that every MGA use a helper objective or similar selection function that preserves length-based diversity in the population. The helper objective has the added benefit of ensuring the final population will contain solutions of several different lengths. This affords the user additional options when choosing the optimal solutions.

5.3 Variable-length recombination

Four variable-length recombination operators were tested for the MGA, plus a mutation-only algorithm. The difference in performance was most notable in the sensor coverage problem, in which the MGA using the spatial recombination operator performed best, followed by the similar-metavariable operator and hidden-metavariable representation. In the wind farm problem the performance differences were less notable, but the three aforementioned operators gave the best results. In the laminate stacking problem the differences were minimal.

The difference in performance, when it occurred, is explained by the level of disruption that occurs during recombination. Respectful recombination operators produce children that have schemata, or building blocks, common to both parents. Clusters of sensors that efficiently cover a portion of the domain, or a group of wind turbines that are positioned such they minimize the wake that they cast on each other, can be thought of as solution building blocks. Disrupting these blocks will reduce the quality of the children.

The cut-and-splice operator is highly disruptive. It effectively randomly distributes metavariables from both parents to the children, and it is unsurprising that this operator consistently produced the worst results.

The spatial recombination operator is the least disruptive recombination operator tested. By exchanging metavariables based on their spatial location it helps to guarantee the building blocks in the parents will be preserved. The similar-metavariable operator acts to preserve similarities in the parents and in the process it forms small subsets of metavariables that could be considered building blocks. The

similar-metavariable operator does have the advantage that it could be applied to problems that have no spatial component, such as portfolio optimization.

Building blocks in the sensor coverage and wind farm problems are formed by metavariables that are spatially near to one another. Disruption is most likely to occur when metavariables from one parent are located next to metavariables from the other parent. With the spatial recombination operator, the region in which disruption can occur is limited to the dividing line [63]. However, with the similar-metavariable recombination operator, the metavariables from both parents are intermixed throughout the entire solution, increasing the chance of disruption. This difference between the operators is illustrated in Fig. 13. Modifying to the similar-metavariable recombination operator such that it forms a few large subsets of metavariables, rather than many smaller subsets, may limit the disruption that occurs and improve its performance.

Children produced by the hidden-metavariable representation will more closely resemble those produced by the similar-metavariable operator, compared to those produced by the spatial operator. However, the hidden-metavariable representation has the disadvantage that it is sensitive to the distribution of the metavariables in the genome. Metavariables can only be exchanged in the parents if they are located at the same locus. Using this representation, populations will converge toward a common distribution of metavariables such that recombination can be useful, however this does come at a cost of overall performance. This effect will be more pronounced if multiple subpopulations or other mechanisms are used in which the distribution of metavariables in the genome may diverge further.

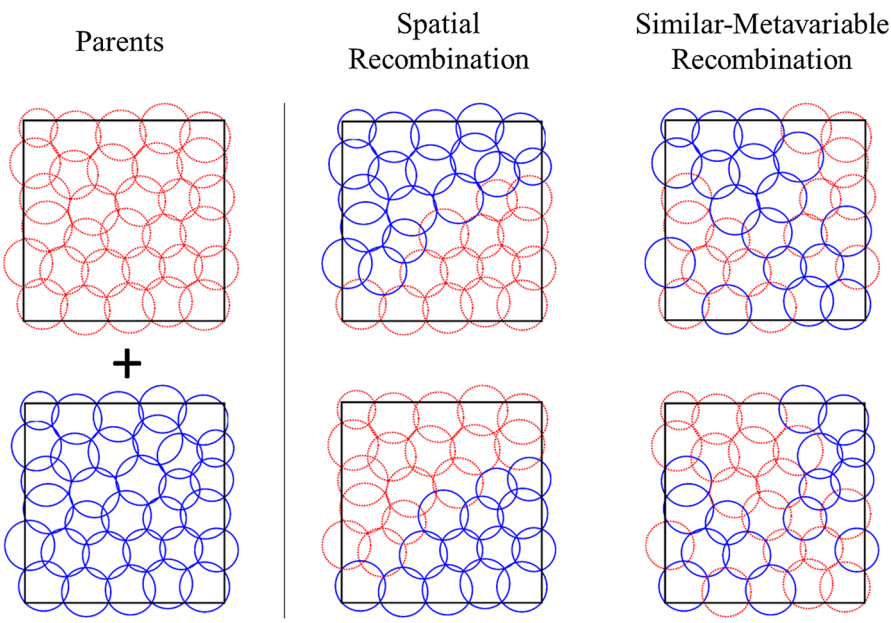

Fig. 13 Sample children produced by the spatial and similar-metavariable recombination operators

When using MGAs, solutions of different lengths, but in the same population, are likely to closely resemble each other. The parents in Fig. 13 are of different lengths, one containing 31 sensors and the other 32. However, 18 of these sensors are identical in both parents, with several others having only minimal differences. The only noticeable differences are located in the top-right quadrant of the domains. Keeping large portions of the solutions identical helps to minimize the disruption that occurs during recombination.

Despite their disruptive nature, MGAs using recombination operators have a higher overall performance compared to the mutation-only MGA.

6 Conclusions

A modified GA, the Metameric Genetic Algorithm, is presented and used to solve several metameric variable-length problems. The performance of different representations, recombination operators, and selection operators were compared for the sensor coverage, wind farm, and laminate stacking problems. Compared to fixed-length algorithms, the MGAs produced significantly better results for the sensor coverage and wind farm problems, and equivalent results for the laminate stacking problems. This is true even if the optimal number of components is assumed known a priori when using the fixed-length algorithms.

The best performing recombination operators are those considered the most respectful. Even if the recombination operators are frequently disruptive, they still improve overall performance compared to a mutation-only method. The helper objective is essential for the sensor coverage problem but not the windfarm or laminate stacking problems. Still, compared to the single-objective algorithm, the helper objective produces equivalent or better results for these problems. Additionally, the helper objective has the benefit of producing several solutions near the optimal length, as opposed to the single solution length typically produced by the single-objective algorithm.

Acknowledgments This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454 to BEACON Center for the Study of Evolution in Action. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. G.D. Edgecombe, G. Giribet, Evolutionary biology of centipedes (Myriapoda: Chilopoda). Annu. Rev. Entomol. **52**, 151–170 (2007)
2. C. Kettle, J. Johnstone, T. Jowett, H. Arthur, W. Arthur, The pattern of segment formation, as revealed by *engrailed* expression, in a centipede with a variable number of segments. Evol. Dev. **5**(2), 198–207 (2003)
3. W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone, *Genetic Programming—An Introduction* (Morgan Kaufmann, San Francisco, 1998)
4. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, 1989)

5. M. Mitchell, *An Introduction to Genetic Algorithms* (The MIT Press, Cambridge, 1998)
6. A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*, 2nd edn. (Springer, Berlin Heidelberg, 2015)
7. T.M. Chan, K.F. Man, K.S. Tang, S. Kwong, A Jumping-Genes Paradigm for Optimizing Factor WLAN Network. IEEE Ind. Inform. **3**(1), 33–43 (2007)
8. G. Soremekun, Z. Gürdal, R.T. Haftka, L.T. Watson, Composite laminate design optimization by genetic algorithm with generalized elitist selection. Comput. Struct. **79**(2), 131–143 (2001)
9. C.H. Park, W.I. Lee, W.S. Han, A. Vautrin, Improved genetic algorithm for multidisciplinary optimization of composite laminates. Comput. Struct. **86**(19–20), 1894–1903 (2008)
10. R. Le Riche, R.T. Haftka, Improved genetic algorithm for minimum thickness composite laminate design. Compos. Eng. **5**(2), 143–161 (1995)
11. C.H. Park, W.I. Lee, W.S. Han, A. Vautrin, Multiconstraint optimization of composite structures manufactured by resin transfer molding process. J. Compos. Mater. **39**(4), 347–374 (2005)
12. J.A. Hageman, R. Wehrens, H.A. van Sprang, L.M.C. Buydens, Hybrid genetic algorithm-tabu search approach for optimizing multilayer optical coatings. Anal. Chim. Acta **490**, 211–222 (2003)
13. A. Gad, O. Abdelkhalik, Hidden genes genetic algorithm for multi-gravity-assist trajectories optimization. J. Spacecr. Rockets **48**(4), 629–641 (2011)
14. S. Bandyopadhyay, U. Maulik, Genetic clustering for automatic evolution of clusters and application to image classification. Pattern Recogn. **35**, 1197–1208 (2002)
15. S. Das, A. Abraham, A. Konar, Automatic clustering using an improved differential evolution algorithm. IEEE Trans. Syst. Man Cybern. A **38**(1), 218–237 (2008)
16. S. Das, S. Sil, Kernel-induced fuzzy clustering of image pixels with an improved differential evolution algorithm. Inform. Sci. **180**, 1237–1256 (2010)
17. S.-M. Pan, K.-S. Cheng, Evolution-based tabu search approach to automatic clustering. IEEE Trans. Syst. Man Cybern. C **37**(5), 827–838 (2007)
18. R.S. Zebulum, M.A. Pacheco, M. Vellasco, Variable length representation in evolutionary electronics. Evol. Comput. **8**(1), 93–120 (2000)
19. N.J. Radcliffe, Forma analysis and random respectful recombination, in *ICGA'91* (Morgan-Kaufmann, San Mateo, 1991), pp. 222–229
20. H. Kargupta, K. Deb, D.E. Goldberg, Ordering genetic algorithms and deception, in *PPSN II* (Elsevier, Amsterdam, 1999), pp. 47–56
21. M. Ryerkerk, R. Averill, K. Deb, E. Goodman, Optimization for variable-size problems using genetic algorithms, in *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* (AIAA, Indianapolis, 2012)
22. X. Han, X. Cao, E.L. Lloyd, C-C. Shen, Deploying directional sensor networks with guaranteed connectivity and coverage, in *SECON'08* (IEEE, San Francisco, 2008), pp. 153–160
23. S.S. Dhillon, K. Chakrabarty, Sensor placement for effective coverage and surveillance in distributed sensor networks, in *WCNC'03*, vol. 3 (IEEE, New Orleans, 2003), pp. 1609–1614
24. X. Boukerche, A. Fei, Coverage-preserving scheme for wireless sensor network with irregular sensing range. Ad Hoc Netw. **5**(8), 1303–1316 (2007)
25. C.-F. Huang, Y.-C. Tseng, The coverage problem in a wireless sensor network. Mobile Netw. Appl. **10**(4), 519–528 (2005)
26. Y-C. Wang, C-C. Hu, Y-C. Tseng, Efficient deployment algorithms for ensuring coverage and connectivity of wireless sensor networks, in *WICON'05* (IEEE, Budapest, 2005), pp. 114–121
27. M. Younis, K. Akkaya, Strategies and techniques for node placement in wireless sensor networks: a survey. Ad Hoc Netw. **6**(4), 621–655 (2008)
28. J. Jia, J. Chen, G. Chang, Z. Tan, Energy efficient coverage control in wireless sensor networks based on multi-objective genetic algorithm. Comput. Math. Appl. **57**(11–12), 1756–1766 (2009)
29. R. Kershner, The number of circles covering a set. Am. J. Math. **61**(3), 665–671 (1993)
30. X. Bai, S. Kumar, D. Xuan, Z. Yun, T.H. Lai, Deploying wireless sensors to achieve both coverage and connectivity, in *MobiHoc'06* (ACM, Florence, 2006), pp. 131–142
31. K.J. Nurmela, P.R.J. Östergård, *Covering a Square with up to 30 Equal Circles, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Research Report A62* (Espoo, Finland, 2000)
32. N.A.A. Aziz, K.A. Aziz, W.Z.W. Ismail, Coverage strategies for wireless sensor networks. World Acad. Sci. Eng. Technol. **50**, 145–150 (2009)
33. C.-H. Wu, K.-C. Lee, Y.-C. Chung, A delaunay triangulation based method for wireless sensor network deployment. Comput. Commun. **30**(14), 2744–2752 (2007)

34. Z. Yangyang, J. Chunlin, Y. Ping, L. Manlin, W. Chaojin, W. Guangxing, Particle swarm optimization for base station placement in mobile communication, in *ICNSC '04* (IEEE, Taipei, 2004), pp. 428–432
35. C.-K. Ting, C.-N. Lee, H.-C. Chang, J.-S. Wu, Wireless heterogeneous transmitter placement using multiobjective variable-length genetic algorithm. IEEE Ttans. Syst. Man Cybern. B **39**(4), 945–958 (2009)
36. D.B. Jourdan, O.L. de Weck, Layout optimization for a wireless sensor network using a multi-objective genetic algorithm, in *VTC '04-Spring* (IEEE, Milan, 2004), pp. 2466–2470
37. M. Marks, A survey of multi-objective deployment in wireless sensor networks. J. Telecommun. Inform. Technol. **3**, 36–41 (2010)
38. A. Tonda, E. Lutton, G. Squillero, A benchmark for cooperative coevolution. Memet. Comp. **4**(4), 262–277 (2012)
39. G. Mosetti, C. Poloni, B. Diviacco, Optimization of wind turbine positioning in large windfarms by means of a genetic algorithm. J. Wind Eng. Ind. Aerod. **51**(1), 105–116 (1994)
40. S.A. Grady, M.Y. Hussaini, M.M. Abdullah, Placement of wind turbines using genetic algorithms. Renew. Energy **30**(2), 259–270 (2010)
41. H-S. Huang, Distributed genetic algorithm for optimization of wind farm annual profits, in *ISAP'07* (IEEE, Kaohsiung, 2007), pp. 418–423
42. S. Şişbot, Ö. Turgut, M. Tunç, Ü. Çamdali, Optimal positioning of wind turbines on Gökçeada using multi-objective genetic algorithm. Wind Energy **13**(4), 297–306 (2010)
43. A. Emami, P. Noghreh, New approach on optimization in placement of wind turbines within wind farm by genetic algorithms. Renew. Energy **35**(7), 1559–1564 (2010)
44. Duan, J. Wang, H. Gu, Modified genetic algorithm for layout optimization of multi-type wind turbines, in *ACC 2014* (IEEE, Portland, 2014), pp. 3633–3638
45. J.C. Mora, J.M.C. Barón, J.M.R. Santos, M.B. Payán, An evolutive algorithm for wind farm optimal design. Neurocomputing **70**(16), 2651–2658 (2007)
46. J.S. González, A.G.G. Rodriguez, J.C. Mora, J.R. Santos, M.B. Payán, Optimization of wind farm turbines layout using an evolutive algorithm. Renew. Energy **35**(8), 1671–1681 (2010)
47. J.S. González, A.G.G. Rodriguez, J.C. Mora, M.B. Payán, J.R. Santos, Overall design optimization of wind farms. Renew. Energy **36**(7), 1973–1982 (2011)
48. Wan, J. Wang, G. Yang, X. Zhang, Optimal micro-siting of wind farms by particle swarm optimization, in *ICSI 2010* (Springer, Beijing, 2010), pp. 198–205
49. S. Saavedra-Moreno, A. Salcedo-Sans, L. Paniagua-Tineo, A. Prieto, Portilla-Figueras, seeding evolutionary algorithms with heuristics for optimal wind turbines positioning in wind farms. Renew. Energy **36**(11), 2838–2844 (2011)
50. B.L. DuPont, J. Cagan, An extended pattern search approach to wind farm layout optimization. J. Mech. Des. **134**(081002), 081002-18 (2012)
51. M. Wagner, J. Day, F. Neumann, A fast and effective local search algorithm for optimizing the placement of wind turbines. Renew. Energy **51**, 64–70 (2013)
52. J.F. Herbert-Acero, O. Probst, P.-E. Réthoré, G.C. Larsen, K.K. Castillo-Villar, A review of methodological approaches for the design and optimization of wind farms. Energies **7**(11), 6930–7016 (2014)
53. J.S. González, M.B. Payán, J.R. Santos, F. González-Langatt, A review and recent developments in the optimal wind-turbine micro-siting problem. Renew. Sust. Energy Rev. **30**, 133–144 (2014)
54. N.O. Jensen, A note of wind generator interaction, Risø National Laboratory, Technical report Riso-M-2411 (Roskilde, 1983)
55. S. Frandsen, On the wind speed reduction in the center of large clusters of wind turbines. J. Wind Eng. Ind. Aerod. **39**(1), 251–265 (1992)
56. S. Venkataraman, R.T. Haftka, Optimization of composite panels—a review, in *Proceedings of the 14th Annual Technical Conference of the American Society of Composites* (Dayton, 1999), pp. 479–488
57. L.A. Schmit, B. Farshi, Optimum design of laminated fibre composite plates. Int. J. Numer. Method Eng. **11**(4), 623–640 (1977)
58. S.N. Omkar, R. Khandelwal, S. Yathindra, G.N. Naik, S. Gopalakrishnan, Artificial immune system for multi-objective design optimization of composite structures. Eng. Appl. Artif. Intel. **21**(8), 1416–1429 (2008)
59. I.M. Daniel, O. Ishai, *Engineering Mechanics of Composite Materials*, 2nd edn. (Oxford University Press, New York, 2006)

60. A.R.M. Rao, N. Arvind, A scatter search algorithm for stacking sequence optimization of laminate composites. Compos. Struct. **70**(4), 383–402 (2005)
61. F.-X. Irisarri, D.H. Bassir, N. Carrere, J.-F. Maire, Multiobjective stacking sequence optimization for laminated composite structures. Compos. Sci. Technol. **69**(7–8), 983–990 (2009)
62. D.R. Dasgupta, S.G.A. Mcgregor, A structured genetic algorithm, University of Strathclyde, Department of Computer Science, Technical report IKBS-8-92 (Glasgow, 1992)
63. D.M. Cherba, W. Punch, Crossover gene selection by spatial location, in *GECCO'06* (ACM, Seattle, 2006), pp. 1111–1116
64. B. Hutt, K. Warwick, Synapsing variable-length crossover: meaningful crossover for variable-length genomes. IEEE Trans. Evolut. Comput. **11**(1), 118–131 (2007)
65. K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms* (Wiley, Chichester, 2001)
66. K.O. Stanley, R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies, in *GECCO'02* (Morgan Kaufmann Publishers, New York, 2002), pp. 569–577