Evolving Generalizable Multigrid-Based Helmholtz Preconditioners with Grammar-Guided Genetic Programming

Jonas Schmitt
jonas.schmitt@fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany

Harald Köstler
harald.koestler@fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany

ABSTRACT

Solving the indefinite Helmholtz equation is not only crucial for the understanding of many physical phenomena but also represents an outstandingly-difficult benchmark problem for the successful application of numerical methods. Here we introduce a new approach for evolving efficient preconditioned iterative solvers for Helmholtz problems with multi-objective grammar-guided genetic programming. Our approach is based on a novel context-free grammar, which enables the construction of multigrid preconditioners that employ a tailored sequence of operations on each discretization level. To find solvers that generalize well over the given domain, we propose a custom method of successive problem difficulty adaption, in which we evaluate a preconditioner's efficiency on increasingly ill-conditioned problem instances. We demonstrate our approach's effectiveness by evolving multigrid-based preconditioners for a two-dimensional indefinite Helmholtz problem that outperform several human-designed methods for different wavenumbers up to systems of linear equations with more than a million unknowns.

CCS CONCEPTS

• **Theory of computation → Genetic programming**; **Preconditioning**; Grammars and context-free languages; • **Mathematics of computing → Solvers**; Partial differential equations.

KEYWORDS

Genetic Programming, Multigrid, Preconditioning, Helmholtz Equation, Grammar-Based, Shifted Laplacian, Generalization, Multi-Objective, Partial Differential Equations, Artificial Intelligence

ACM Reference Format:
Jonas Schmitt and Harald Köstler. 2022. Evolving Generalizable Multigrid-Based Helmholtz Preconditioners with Grammar-Guided Genetic Programming. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3512290.3528688

1 INTRODUCTION

Automated algorithm design is a long-standing challenge in artificial intelligence (AI) and has two essential goals: *Generalization* and *efficiency*. Thus, the designed algorithm should not only produce the correct output for arbitrary inputs, but the goal is also to achieve better performance than methods designed by a human expert. In this work, we aim to demonstrate that this goal is attainable for the indefinite Helmholtz equation, an important benchmark problem from the domain of numerical mathematics. The Helmholtz equation frequently arises in the study of physical phenomena, such as electromagnetics [47] and acoustics [41], and is given by the linear partial differential equation (PDE)

$$-\nabla^2 u - k^2 u = f, \tag{1}$$

where ∇^2 is the Laplace operator, k the *wavenumber*, and f the source term. In general, the analytic solution u of this equation is unknown, which necessitates the use of numerical methods. Unfortunately, for large wavenumbers, the system of linear equations that arises from the resulting discretization becomes indefinite and highly ill-conditioned, which means that even small perturbations, for instance due to numerical inaccuracies, have a dramatic effect on the overall error of the computed approximation. As a consequence, the efficient solution of the indefinite Helmholtz equation is still an open challenge in numerical mathematics [1, 12, 14]. Even though various methods for solving this equation have been proposed, many of them fail to generalize over different ranges of wavenumbers and are, thus, only limited to certain problem instances. Therefore, the design of an efficient Helmholtz solver is not only of great significance for many real-world problems [4, 20, 36, 54] but also represents a challenging benchmark for the application of AI-based methods.

Data-driven [28, 32] and physics-informed [25, 43] machine learning models have recently achieved significant progress in solving PDEs. While these approaches have shown competitive or even improved performance compared to classical solvers, their behavior on unseen problems is often difficult to predict, and hence generalization is only possible to a limited degree [33]. Additionally, many of these models require an enormous amount of training data, whose generation still depends on the utilization of conventional numerical methods. An alternative approach is the application of AI-based optimization methods to improve the efficiency of an existing solver. In contrast to a trained machine learning model, numerical methods can be formulated in the language of mathematics in a problem-independent manner, which significantly facilitates their generalizability. Furthermore, as this formulation can be understood by a human expert, it is possible to profoundly analyze their behavior based on existing domain knowledge and expertise. A numerical method that has achieved considerable success in solving Helmholtz problems [12, 18, 40], as well as many other complex PDEs [3], is the acceleration of a slowly converging iterative method with the application of a so-called preconditioner M [2].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '22, July 9–13, 2022, Boston, MA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9237-2/22/07…$15.00
https://doi.org/10.1145/3512290.3528688

This approach is based on the idea of considering the modified system of linear equations

$$AM^{-1}\hat{u} = f, \tag{2}$$

where A represents the discretized operator of the original system. The main requirement is then to have an efficient method for solving the system

$$Mu = \hat{u}, \tag{3}$$

such that $u \approx M^{-1}\hat{u}$. Multigrid methods are a class of numerical methods for solving discretized PDEs that can fulfill this requirement [6, 22, 53]. If properly constructed, these methods achieve h-independent convergence while only requiring $O(n)$ operations, which means that the number of iterations required for solving a system of linear equations with n unknowns is independent of the discretization width h. While a suitable preconditioning matrix M can often be obtained by analyzing the properties of the system matrix A, constructing an efficient multigrid method for its inversion is usually less intuitive and can, thus, be considered a problem of optimal algorithm design. Since their invention by Federenko and Brandt [5, 16], multigrid solvers have been designed predominantly by hand. Only in recent decades, the automated optimization of these methods has become an active field of research.

Related Work on Multigrid Solver Design. In principle, a multigrid method is characterized by a finite number of components and design choices that determine its computational structure: The choice of the *smoother*, *prolongation* and *restriction* operator, *coarse grid solver* and *cycle type* [6, 53]. A common approach to automate multigrid solver design is to formulate the task as a discrete optimization problem, which is then solved, for instance, using an evolutionary algorithm [39], branch-and-bound [52], or minimax approach [7]. A different direction is the application of machine learning methods either to optimize the individual components of a multigrid solver, as the prolongation operator [21, 26, 35] and smoother [15, 24], or by replacing certain steps within the method altogether by a machine learning system [51]. All these approaches have in common that they consider a multigrid method's algorithmic structure immutable. Each step of the method employs a fixed sequence of operations in the form of a particular cycle. Multigrid cycles are commonly classified into three different categories, V-, W-, and F-cycles, where each cycle type exhibits a distinct computational pattern that represents a compromise between the amount of work performed and the expected speed of convergence [53]. In [49, 50] we have proposed a context-free grammar that allows alternating each step of a multigrid solver independently. Consequently, the search space produced by this grammar includes methods that do not fit into any of the known categories. While until recently, solvers of such unconventional structure have not been considered, in [50] it could be demonstrated that multigrid methods evolved by a grammar-based genetic programming approach can achieve higher efficiency in solving certain PDEs than traditional variants. However, in contrast to the indefinite Helmholtz equation, the PDEs considered in this work can already be efficiently solved with standard multigrid cycles without requiring any further optimization. Furthermore, while we have demonstrated that the solvers obtained by this approach can also function for larger problem instances than those considered within the search, a systematic approach to generalize a multigrid method to a family of problem instances that share common characteristics is still missing. To overcome these limitations and extend the context-free grammar introduced in [49, 50] to the domain of multigrid preconditioners, we make the following contributions.

Our Contributions. We introduce a multi-objective grammar-guided evolutionary search method for finding multigrid preconditioners that generalize well over a sequence of increasingly difficult problem instances and demonstrate its effectiveness by evolving preconditioners for the discretized Helmholtz equation with increasingly high wavenumbers.

- To apply our evolutionary search method to the domain of multigrid preconditioners, we adapt the class of context-free grammars presented in [49, 50] such that the generated methods can be integrated into an existing iterative solver as a preconditioner. To our knowledge, this is the first formal system that enables the application of grammar-guided genetic programming (GGGP) [37, 55] to the design of preconditioned iterative solvers in a generalizable way.
- Since our grammar-based representation of multigrid preconditioners is problem-size independent, each method can be ported and applied to similar problem instances without the need to adapt its internal structure.
- Our evolutionary search method is based on classical tree-based GGGP but copes with the high computational demands for solving PDEs numerically by combining multi-objective optimization with a custom method of successive problem difficulty adaption based on the h-independent convergence of multigrid methods.
- We demonstrate that our implementation of GGGP can be scaled up to recent clusters and supercomputers by running our experiments on multiple nodes of SuperMUC-NG, currently one of the largest supercomputing systems in Europe.
- The multigrid preconditioners evolved with our method outperform all common multigrid cycles [6, 53] with optimized relaxation factors for representative instances of the indefinite Helmholtz equation with different wavenumbers. Furthermore, a subset of these methods yields a converging solver for a problem of higher difficulty and size than those considered within the optimization and for which all common multigrid cycles fail to achieve convergence.

2 A FORMAL GRAMMAR FOR GENERATING MULTIGRID PRECONDITIONERS

We can derive a formal grammar for generating multigrid preconditioners from the one formulated in [49, 50] by replacing the system matrix A with the respective preconditioning matrix M and the right-hand side f with $\hat{u}$. Table 1a contains the resulting productions for generating a multigrid preconditioner that operates on a hierarchy of three grids, where a spacing of h is used on the finest grid and the only operation allowed on the coarsest grid is the application of a direct solver, denoted by the multiplication with the inverse of M_{4h}. Each rule then defines the set of expressions by which a certain variable, denoted by $\langle\cdot\rangle$, can be replaced. Starting with the symbol $\langle S\rangle$, each expression can be substituted recursively

Table 1: Context-free grammar for generating three-grid preconditioners.

(a) Productions

$\langle S \rangle$	$\models$	$\langle s_h \rangle$
$\langle s_h \rangle$	$\models$	$\text{ITERATE}(\omega, \langle P \rangle, \text{APPLY}(\langle B_h \rangle, \langle c_h \rangle))$ |
		$\text{ITERATE}(\omega, \lambda, \text{CGC}(I_{2h}^{h}, \langle s_{2h} \rangle))$ | $(u_h^0, \hat{u}_h, \lambda, \lambda)$
$\langle c_h \rangle$	$\models$	$\text{RESIDUAL}(M_h, \langle s_h \rangle)$
$\langle B_h \rangle$	$\models$	$\text{INVERSE}(M_h^{+})$ with $M_h = M_h^{+} + M_h^{-}$
$\langle c_{2h} \rangle$	$\models$	$\text{RESIDUAL}(M_{2h}, \langle s_{2h} \rangle)$ |
		$\text{COCY}(M_{2h}, u_{2h}^0, \text{APPLY}(I_h^{2h}, \langle c_h \rangle))$
$\langle s_{2h} \rangle$	$\models$	$\text{ITERATE}(\omega, \langle P \rangle, \text{APPLY}(\langle B_{2h} \rangle, \langle c_{2h} \rangle))$ |
		$\text{ITERATE}(\omega, \lambda, \text{APPLY}(I_{4h}^{2h}, \langle c_{4h} \rangle))$
$\langle B_{2h} \rangle$	$\models$	$\text{INVERSE}(M_{2h}^{+})$ with $M_{2h} = M_{2h}^{+} + M_{2h}^{-}$
$\langle c_{4h} \rangle$	$\models$	$\text{APPLY}(M_{4h}^{-1}, \text{APPLY}(I_{2h}^{4h}, \langle c_{2h} \rangle))$
$\langle P \rangle$	$\models$	PARTITIONING | λ

(b) Semantics

function $\text{ITERATE}(\omega, P, (u, \hat{u}, \delta, state))$
  $\tilde{u} \leftarrow u + \omega \cdot \delta$ with P
  return $(\tilde{u}, \hat{u}, \lambda, state)$
end function
function $\text{APPLY}(B, (u, \hat{u}, \delta, state))$
  $\tilde{\delta} \leftarrow B \cdot \delta$
  return $(u, \hat{u}, \tilde{\delta}, state)$
end function
function $\text{RESIDUAL}(M, (u, \hat{u}, \lambda, state))$
  $\delta \leftarrow \hat{u} - Mu$
  return $(u, \hat{u}, \delta, state)$
end function
function $\text{COCY}(M_H, u_H^0, (u_h, \hat{u}_h, \delta_H, state_h))$
  $u_H \leftarrow u_H^0$
  $\hat{u}_H \leftarrow \delta_H$
  $\tilde{\delta}_H \leftarrow \hat{u}_H - M_H u_H$
  $state_H \leftarrow (u_h, \hat{u}_h, \lambda, state_h)$
  return $(u_H, \hat{u}_H, \tilde{\delta}_H, state_H)$
end function
function $\text{CGC}(I_H^h, (u_H, \hat{u}_H, \lambda, state_H))$
  $(u_h, \hat{u}_h, \lambda, state_h) \leftarrow state_H$
  $\delta_h \leftarrow I_H^h \cdot u_H$
  return $(u_h, \hat{u}_h, \delta_h, state_h)$
end function

according to the specified rules until it contains either exclusively terminal symbols or the empty string λ [34]. The resulting derivation tree uniquely represents a multigrid preconditioner on the specified hierarchy of grids. To obtain a grammar for generating multigrid preconditioners that operate on an extended hierarchy, for instance a four or five-grid method, we have to replicate the production rules formulated on the second finest level ($2h$) in Table 1a for each subsequent one. Similar as in [49, 50] we can then

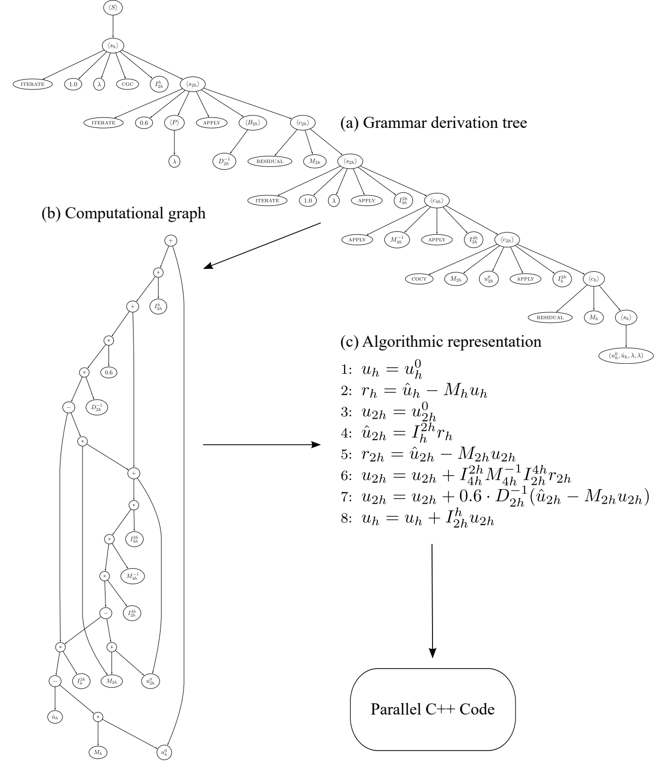

Figure 1: Visualization of the process of mapping the grammar derivation tree of a three-grid V-cycle with a single step of Jacobi post-smoothing on the second finest level ($2h$) to an algorithmic representation.

formulate semantic evaluation rules, which are shown in Table 1b. These rules guide the derivation of the corresponding sequence of multigrid operations obtained in form of a directed acyclic graph (DAG). Figure 1 illustrates the resulting process of algorithm generation with the example of a three-grid V-cycle that performs a single underrelaxed Jacobi post-smoothing step on the second finest discretization level. The resulting algorithm is formulated in Figure 1c, while Figure 1a shows the corresponding derivation tree based on the productions formulated in Table 1a. Through recursive application of the rules in Table 1b the computational DAG shown in Figure 1b is obtained. Note that the recursive application of these rules in the end always returns a tuple of the form $(u_h, \hat{u}_h, \lambda, \lambda)$, whereas u_h is the resulting computational graph. Based on this intermediate representation, we can obtain an algorithmic formulation of the corresponding multigrid preconditioner by introducing variables for the approximate solution u, right-hand side $\hat{u}$ and residual r on each discretization level, which, again, leads to Figure 1c. By making use of recent code generation techniques, it is then possible to automate the generation of optimized C++ code for a given multigrid-based solver specified in an algorithm-like fashion [27, 31].

Finally, note that while Table 1a fixes the number of coarsening steps until the respective problem can be solved directly, all operations are formulated relative to the discretization width h. As a

consequence, the computational structure of the resulting preconditioner is independent of the actual size of the grid. It is, therefore, possible to translate a multigrid preconditioner formulated on a hierarchy of grids with a certain depth to another one consisting of different-sized grids of the same depth. For this purpose, we only have to replace the initial approximate solution u_h^0, operator M_h and right-hand side $\hat{u}_h$ in Table 1a by their counterparts and reformulate each operator on the respective grid within the new hierarchy. For instance, the derivation tree in Figure 1a can be translated to the computational DAG of a structurally similar multigrid preconditioner for a different problem discretized on a hierarchy of three grids. Consequently, every multigrid method produced by a grammar formulated on a particular hierarchy of grids is generalizable over the set of all structurally equivalent grammars that employ the same number of coarsening steps. We can thus apply a multigrid preconditioner obtained on a specific instance of the Helmholtz equation to similar problems. In the following, we utilize this principle to evolve efficient multigrid-based preconditioners that can be generalized to a whole class of Helmholtz problems.

3 EVOLUTIONARY SEARCH METHOD

After establishing a generalizable representation for multigrid preconditioners of arbitrary structure, the next task is to formulate a search method that can identify those leading to an efficient solver for different Helmholtz problem instances. While the branching factor of the productions formulated in Table 1a may seem small at first, it has already been shown in [50] that, in practice, the resulting search space exceeds any size for which a simple exhaustive search is applicable. In [50] we have provided $3 \cdot 10^{14}$ as a lower bound for the size of the search space of a three-grid method with a limited number of choices for smoothing on each level. Since the construction of a multigrid preconditioner comprises a similar number of choices, this lower bound also applies in the present case, rendering a mere brute-force search unfeasible. Search heuristics can often find an acceptable approximation for the global optimum when the search space is too large to evaluate all possible solutions. In principle, the quality of a preconditioner can be assessed by considering two objectives. First of all, preconditioning aims to minimize the condition number of the matrix AM^{-1}. Since, in practice, the inverse M^{-1} is not computed explicitly, the effectiveness of a preconditioner depends on its approximation accuracy, which directly affects the *number of iterations* required by an iterative method to achieve a certain error reduction. On the other hand, a method that achieves the same quality of approximation but can be executed faster on modern computer architectures achieves a lower *execution time per iteration*. The task of finding an optimal multigrid preconditioner can thus be considered as a multi-objective search problem.

Fitness Evaluation and Generalization. As shown in [12] the number of iterations required for solving Equation (1) using a preconditioned solver grows with the wavenumber k. This work aims to obtain multigrid methods that can be generalized over different problem instances. Therefore, before evaluating a given preconditioner on problems with a large wavenumber, we first consider an instance of the same problem with a smaller wavenumber and hence lower difficulty. When we start with a random initialization, the

Algorithm 1 Evolutionary Search

```
Construct the grammar G_0 for the initial problem
Initialize the population P_0 based on G_0
Evaluate P_0 on the initial problem
for i := 0, ..., n do
    if i > 0 and i mod m = 0 then
        j := i/m
        Increase the problem difficulty
        Construct the corresponding grammar G_j
        Adapt the current population P_i to G_j
        Evaluate P_i on the new problem
    end if
    Generate new solutions C_i based on P_i and G_j
    Evaluate C_i on the current problem
    Select P_{i+1} from C_i ∪ P_i
end for
```

probability of generating multigrid methods that do not represent effective preconditioners is high, even for a problem instance that is, in principle, comparably easy to solve. As the search progresses, the average quality of the obtained preconditioners is expected to improve. Hence, the probability that they can also be successfully applied to instances with higher difficulty increases. On the other hand, most multigrid methods that are efficient in preconditioning a problem instance with a large wavenumber can be expected to function also on a problem with a smaller wavenumber. We, therefore, propose a stepwise adaption of the difficulty of the evaluated problem. To perform the actual search, we employ a multi-objective GGGP-based algorithm that operates on a population of derivation trees [37, 55]. Each tree represents a certain point in the search space considered at the current step of the method. In every new step, the search progresses by creating a new population of trees based on the current one. The resulting procedure is summarized in Algorithm 1, in which the difficulty of the problem considered for evaluation is adjusted in every mth iteration of the search.

The question that remains to be answered is how a new population P_{i+1} should be generated in each step of Algorithm 1 based on the current one. In principle, the current population P_i represents the subspace of possible multigrid preconditioners considered within step i of the search. Accordingly, the generation of P_{i+1} represents moving the search to a new subspace, which is expected to contain solutions that, according to both objectives, correspond to multigrid methods representing more efficient preconditioners for the given problem than those located in the current subspace. Consequently, we need to evaluate the quality of a subspace represented by the current population in terms of its potential to obtain efficient preconditioners from it. In principle, the number of iterations required to achieve a particular error reduction with a preconditioned iterative method could be predicted with local Fourier analysis [9]. However, there has been only a limited amount of research on the accuracy of this method for evaluating nonstandard multigrid methods. In particular, the experiments performed in [49] indicate that the predictions obtained with this method are not always consistent with experimentally determined behavior. Alternatively, we can obtain all relevant performance characteristics of a solver through its direct application to a representative test problem. For this purpose,

it is necessary to automatically generate an implementation based on the algorithmic representation of those multigrid preconditioners obtained through semantical evaluation of the derivation trees produced by the respective grammar. ExaStencils [31] is a framework that has been specifically designed for the automatic generation of scalable multigrid implementations based on a tailored domain-specific language (DSL) called ExaSlang [48]. It enables the specification of solvers in a discretization level-independent manner. At the same time, the actual size of the problem can be controlled utilizing simple configuration files, which grants us the possibility to automatically generate implementations for a specific solver that are executable on a wide range of different computer architectures. These can then be evaluated on the target platform for the two objectives, i.e., number of iterations and execution time per iteration.

Implementation of Grammar-Guided Genetic Programming. In each iteration of the search, we create a new population based on the existing one using GGGP, where each individual represents a derivation tree of the form of Figure 1a. To apply this method to the grammar formulated in Table 1a, we first need to consider its unique structure. Note that except for the variables $\langle s_h \rangle$, $\langle B_h \rangle$, $\langle B_{2h} \rangle$ and $\langle P \rangle$ none of the expressions generated by any of the productions of a variable exclusively consists of terminals. Consequently, the grammar does not permit the construction of a derivation tree with branches of equal length. We, therefore, employ the *grow* strategy as described in [29, 42] to initialize the population. A derivation tree is extended by randomly choosing a production from the combined set of terminal and non-terminal productions until the longest path within the tree exceeds a certain depth. Since the grammar comprises an even branching factor of two for all non-terminal productions, there is no need to adapt the probability of selecting a particular production, but choosing uniformly from the combined set already results in a sufficient diversity in the population.

To create new individuals based on an existing population, we employ mutation and recombination. For this purpose, we first select several individuals using a binary tournament selection based on the dominance relation and crowding distance between individuals, as described in [11]. Mutation is performed by randomly selecting a variable node within the given derivation tree. The subtree for which this variable represents the root node is then replaced by a new randomly generated tree, which is created using the *grow* initialization operator. However, we also permit the insertion of the replaced subtree as a branch within the new one. Note that this insertion, which is only allowed once, is only possible if the variable that represents the root node of the original subtree occurs within the new one. Consequently, if this condition is never fulfilled, the original subtree is replaced without insertion. Therefore, our mutation operator can either perform subtree replacement or insertion in case the newly generated subtree can connect the original one to its root node. While mutation is performed on a single individual, within recombination we create two new individuals by combining the derivation trees of two individuals selected as *parents*. For this purpose, we employ standard subtree crossover as described in [42], whereby we choose the crossover point uniformly among all possible nodes within both trees. Finally, after the creation and evaluation of a certain number of novel solutions, we employ the sorting procedure described in [17] to identify the non-dominated solutions in the combined set of the newly created and existing ones. These individuals then form the new population P_{i+1} in the next step of Algorithm 1.

4 EXPERIMENTAL EVALUATION

To evaluate the effectiveness of our evolutionary search method in finding multigrid methods that act as efficient preconditioners, we consider the two-dimensional Helmholtz equation on a unit square with Dirichlet boundary conditions at the top and bottom, and Robin radiation conditions at the left and right, as defined by

$$\begin{aligned}(-\nabla^2 - k^2)u &= f \quad \text{in } (0,1)^2 \\ u &= 0 \quad \text{on } (0,1)\times\{0\}, (0,1)\times\{1\} \\ \partial_{\mathbf{n}} u - iku &= 0 \quad \text{on } \{0\}\times(0,1), \{1\}\times(0,1) \\ f(x,y) &= \delta(x-0.5, y-0.5),\end{aligned}$$

where $\delta(\mathbf{x})$ represents the Dirac delta function. We discretize this equation on a uniform Cartesian grid using the classical five-point stencil

$$\frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4-(kh)^2 & -1 \\ & -1 & \end{bmatrix},$$

while the Dirac delta function is approximated with a second-order Zenger correction [30]. The step size h of the grid is chosen to fulfill the second-order accuracy requirement $hk = 0.625$ as suggested in [13]. Since the analytic solution of this equation is not known in advance, we consider an approximate solution to be sufficient if the initial residual has been reduced by a factor of 10^{-7} for $k \leq 160$ and 10^{-6} for all larger wavenumbers. The resulting complex-valued system of linear equations is indefinite, and the required number of iterations for solving it with a non-preconditioned Krylov subspace method increases drastically with the wavenumber k [12, 13]. As a solver, we, therefore, employ a biconjugate gradient stabilized method (BiCGSTAB) [46], right-preconditioned with a shifted Laplacian

$$M = -\nabla^2 - (k^2 + 0.5ik^2),$$

which is among the suggested solvers in [12]. In each step of this iterative scheme, it is necessary to compute an approximate solution for two systems of linear equations of the form $Mu = \hat{u}$, each of which is achieved through the application of a single multigrid iteration.

4.1 Optimization Settings

To evaluate the behavior of our GGGP-based search method, we perform a total number of ten randomized optimization runs. While we are aware that this number is insufficient for a reasonable statistical evaluation of an evolutionary algorithm's behavior, it still enables us to demonstrate that our method is capable of repeatedly evolving generalizable preconditioners for the given test problem. Even though a more accurate assessment of our method's behavior would be desirable, the high computational and temporal costs of each run, which each take between 24 and 48 hours, put a strict limit on the number of experiments. Within all optimization runs, we choose a step size of $h = 1/2^l$ on each level l, whereby we employ a range of $l \in [l_{max} - 4, l_{max}]$. Accordingly, our goal is to construct

an optimal five-grid preconditioner for the given problem. For this purpose, we consider the following components:

Smoothers: Pointwise and block Jacobi with rectangular blocks up to a maximum number of six terms, red-black Gauss-Seidel
Restriction: Full-weighting restriction
Prolongation: Bilinear interpolation
Relaxation factors: $(0.1 + 0.05i)_{i=0}^{36} = (0.1, 0.15, 0.2, \ldots, 1.9)$
Coarse-grid solver: BiCGSTAB for $l = l_{max} - 4$

To generate block Jacobi smoothers, we define a splitting $M_h = L_h + D_h + U_h$ where D_h is a block diagonal matrix, such that we have to solve a local system whose size corresponds to the size of a block at every grid point. For a more detailed treatment of block relaxation methods, the reader is referred to [46, 53]. The relaxation factor ω for each smoothing and coarse-grid correction step is chosen from the above sequence. By employing the same code generation-based optimizations for each component of a solver, we ensure that the resulting measurements are comparable for all multigrid variants considered in this work. All experiments are performed on the SuperMUC-NG cluster, where each node represents an Intel Skylake Xeon Platinum 8174 processor that consists of eight islands, each with six physical cores. While within the optimization, we evaluate each individual's fitness on a single island, the final evaluation is performed on a full node of the system with 48 cores. We employ GCC 7.5 as a compiler, using the -O3 optimization level and an OpenMP-based parallelization in both cases. To assess each preconditioner's generalizability, we consider the three different wavenumbers 160, 320, and 640, together with a discretization width of $h = 0.625/k$.

4.2 Reference Methods

To establish a baseline, we consider several well-known and commonly used multigrid cycles [53] that are all based on the application of a certain smoother for a fixed number of times. The resulting solver is translated to ExaStencils' DSL, based on which a multi-threaded C++ implementation is generated and executed on a full SuperMUC-NG node using 48 OpenMP threads. To optimize each multigrid cycle's effectiveness as a preconditioner, we experimentally obtain the optimum relaxation factor for $k = 320$ from the mentioned interval. While in [8, 12] a damped Jacobi is employed as a smoother, in the given case, it does not result in a convergent solver for $k > 80$. We have verified this assumption by considering every possible relaxation factor value from the given interval. In contrast, red-black Gauss-Seidel represents an effective smoother for the considered range of k. The second column of Table 2 contains the optimum red-black Gauss-Seidel relaxation factor (ω) for each cycle. For instance, V(2, 1) represents a V-cycle that performs two pre- and one post-smoothing step on each discretization level. Using the same relaxation factor, we employ each cycle as a preconditioner for the three wavenumbers considered. For consistent measurements, each solver is executed ten times to compute the average solving time of all runs, which reduces the deviations to a negligible level. The results are shown in the remaining columns of Table 2. Here omitted values imply that the corresponding preconditioned BiCGSTAB method did not achieve the required error reduction within 20,000 iterations. Additionally, we have evaluated each resulting cycle on a problem with wavenumber $k = 640$, which did not yield a convergent solver in any of the cases. For the other two wavenumbers considered, the W(3, 3)-cycle represents the most effective preconditioner and therefore leads to the lowest number of iterations, while the V(0, 1)-cycle yields the overall fastest solving time on the given platform.

Table 2: Reference methods - Optimum relaxation factors ω for $k = 320$, number of iterations and average time required for solving a problem with the particular wavenumber.

	ω	Iterations		Solving Time (s)	
k		160	320	160	320
V(0, 1)	1.25	2078	6297	6.38	35.11
V(1, 1)	0.6	1880	6297	7.66	44.27
V(2, 1)	0.6	–	5532	–	47.0
V(2, 2)	0.5	1627	5115	9.93	50.54
V(3, 3)	0.4	1753	5168	13.97	76.00
F(0, 1)	1.15	1467	4028	8.15	42.87
F(1, 1)	0.75	1546	3988	11.21	54.51
F(2, 1)	0.55	1146	3934	10.87	67.62
F(2, 2)	0.65	1060	3213	13.92	65.06
F(3, 3)	0.45	1085	3464	18.88	92.97
W(0, 1)	0.75	1265	4215	8.67	72.08
W(1, 1)	0.8	1208	3570	13.08	76.22
W(2, 1)	0.6	1313	3074	17.71	79.67
W(2, 2)	0.5	1069	3376	17.14	101.6
W(3, 3)	0.45	942	2976	19.65	117.8

Table 3: Best preconditioners according to the product of both objectives - Number of iterations and average time required for solving a problem with the particular wavenumber.

	Iterations			Solving Time (s)		
k	160	320	640	160	320	640
EP-1	1178	3399	–	6.29	28.07	–
EP-2	795	2160	8449	7.86	29.89	241.7
EP-3	933	2827	11143	6.08	27.58	257.8
EP-4	637	2509	7901	7.17	41.04	268.2
EP-5	539	1838	7765	5.01	28.39	227.7
EP-6	941	2103	–	9.58	30.76	–
EP-7	955	2701	–	6.45	27.84	–
EP-8	945	2870	10839	7.24	33.02	276.9
EP-9	3436	3872	–	15.15	27.51	–
EP-10	586	1881	8855	6.70	31.39	246.1

4.3 Implementation Details

We have implemented the evolutionary search procedure summarized in Algorithm 1 by extending the approach described in [50]. For this purpose, we generate a new grammar for constructing complex-valued multigrid preconditioners adapted to the respective problem instance in every mth iteration. For evaluating each method's fitness, it is first translated to an ExaSlang [48] representation, which is then automatically integrated into an existing Krylov Subspace method as a preconditioner. Finally, based on the DSL representation of each solver, a multi-threaded C++ implementation is generated, as described in Section 3. To cope with the cost of running a code generation pipeline for each evaluation, together with the growing execution time required for solving the increasingly difficult problem instances, we employ a distributed parallelization using the Message Passing Interface (MPI) library. The resulting implementation is freely available as part of the open-source library EvoStencils[1].

For each experiment, we perform an evolutionary search with a total population size of 128 on eight nodes of SuperMUC-NG using 64 MPI processes, whereby each process is executed on a separate island of the system. As an initial problem, we choose $k = 80$ with a maximum level $l_{max} = 7$, discretized with a step size $h = 1/2^7$. A problem instance with greater difficulty is then constructed by doubling the wavenumber. Note that due to the requirement $hk = 0.625$, this results in a step size half as large as the original one and, in total, a four times larger grid. The relaxation factor for each smoothing and coarse-grid correction step is chosen from the interval specified above. Each preconditioner is evaluated on the respective island using 12 threads. A solver is considered convergent for all wavenumbers if it can reduce the initial residual by 10^{-7} in less than 10,000 iterations. The initial population is obtained through the non-dominated sorting of a randomly generated set of 1024 individuals. The search is then performed for 150 iterations, whereby the difficulty is adjusted every 50 iterations. New derivation trees are created through recombination with a probability of 2/3 or by mutation. In the latter, a terminal symbol is chosen with a probability of 1/3. To evaluate the consistency of the obtained results, we perform ten experiments with a random initialization. At the end of each experiment, we identify the ten best preconditioners according to the product of both objectives and evaluate them under the same conditions as the reference methods, i.e., by executing each solver ten times on a full SuperMUC-NG node using 48 OpenMP threads.

5 RESULTS AND DISCUSSION

Table 3 contains the results for each preconditioner from the set of non-dominated solutions of the respective experiment that achieves the fastest solving time for a wavenumber of $k = 640$. Note that, similar to all reference methods, in four of the ten cases, EP-1, EP-6, EP-7, and EP-9, none of the evaluated solvers could achieve convergence for the largest wavenumber $k = 640$. In these cases, we have selected the preconditioner, which leads to the fastest solving time for $k = 320$. Additionally, Figure 2 shows a direct comparison of the best-performing preconditioners from both groups for different wavenumbers. For better comparability of the results achieved on

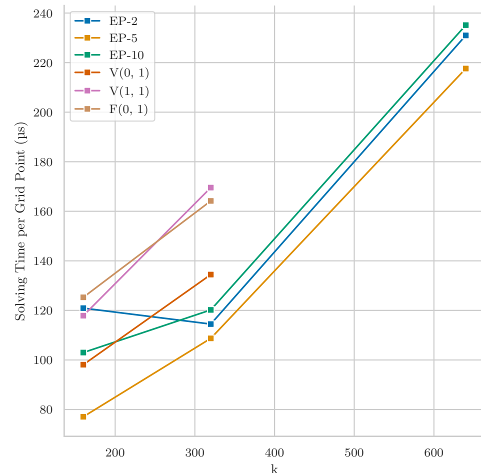

Figure 2: Solving time comparison of the best preconditioners according to the product of both objectives for different wavenumbers (k).

different grid sizes, we measure the solving time per grid point instead of the total time required for solving each problem. Note that all solvers have been evaluated using the same number of measurements on the target platform. All three evolved preconditioners included in this plot represent more efficient methods for $k \geq 320$ than the best of the reference methods, the V(0, 1)-cycle, while remaining competitive for lower wavenumbers. The most efficient preconditioner (EP-5) leads to a consistent improvement of about 20 % compared to the V(0, 1)-cycle for all wavenumbers. Furthermore, while none of the evolved preconditioners has been evaluated on wavenumbers greater than 320 within the search, in six of the ten experiments they lead to a converging solution method for the case of a wavenumber of 640, for which all standard methods fail. In the remaining four experiments, the search still finds competitive preconditioners that generalize well for $k \leq 320$, whereby only in one case (EP-9) preconditioning leads to an inefficient solver for a wavenumber of 160. Therefore, we have demonstrated that our evolutionary search method could find generalizable and efficient methods in the majority of the experiments performed. In addition, our GGGP-based approach was able to evolve methods that surpass the capabilities of standard multigrid cycles in preconditioning Helmholtz problems, which could be demonstrated by solving a problem instance with $k = 640$.

To further investigate our evolutionary algorithm's behavior, Figure 3 shows the distribution of the non-dominated solutions at the end of all ten experiments, whereby the red line represents the non-dominated front of the combined set of solutions. From this distribution, we can conclude that the method can consistently find preconditioners of equal quality for iteration numbers of 4000 or more. For lower iteration numbers, the spread between the solutions increases and, in individual experiments, leads to suboptimal preconditioners in the left part of the solution space. Reducing the number of iterations requires more smoothing and coarse-grid corrections steps with an effective combination of relaxation factors,

[1]EvoStencils: https://github.com/jonas-schmitt/evostencils

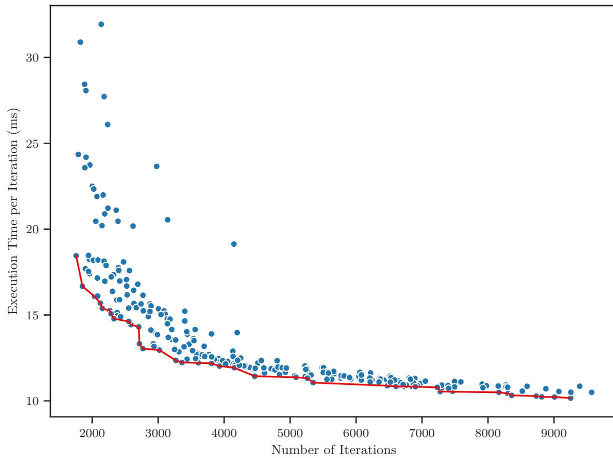

Figure 3: Distribution of non-dominated solutions at the end of all ten experiments for $k = 320$. The red line denotes the combined front.

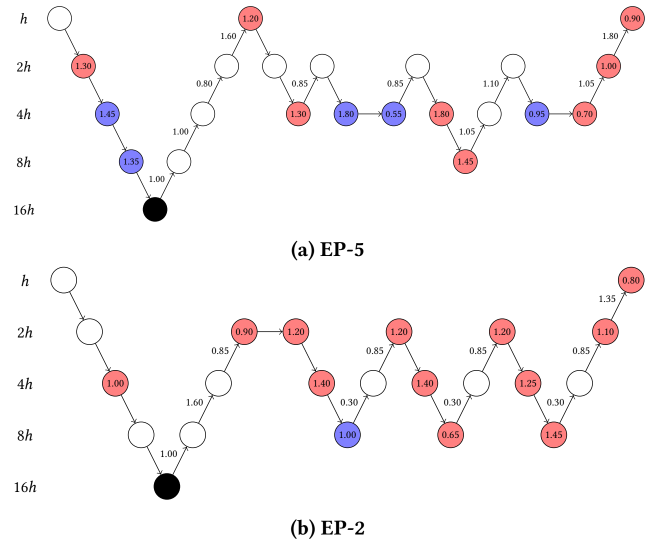

Figure 4: Computational structure of the evolved multigrid preconditioners. The color of the node denotes the type of operation. Black: Coarse-grid solver, Blue: Pointwise Jacobi smoothing, Red: Red-black Gauss-Seidel smoothing, White: No operation. The relaxation factor of each smoothing step is included in each node, while for coarse-grid correction, it is attached to the respective edge.

which results in a growth of the size of the corresponding grammar derivation tree. This effect impedes the search algorithm's ability to find the right combination of productions that leads to the same Pareto-optimal preconditioner in every experiment.

While the efficiency and generalizability of the evolved preconditioners could be demonstrated, we have not yet analyzed how these methods function algorithmically. For this purpose, we consider the two evolved preconditioners, EP-5 and EP-2, that perform best for wavenumbers $k \geq 320$. Figure 4 contains a graphical representation of each method's computational structure, including all relaxation factors. These figures illustrate that, in each of the two cases, our grammatical representation of a multigrid preconditioner has enabled the construction of a unique sequence of computations, whose complexity exceeds those obtained by classical parameter optimization methods such as [7, 39, 52]. While both algorithms resemble a V-cycle, as the coarse-grid solver is only employed once, they include additional smoothing-based coarse-grid correction steps. In contrast to classical multigrid cycles, these corrections are obtained from intermediate discretization levels without traversing the complete hierarchy down to the coarsest grid. Furthermore, in both preconditioners, pre- and post-smoothing steps are omitted on certain levels, while the amount of smoothing is increased on others. In EP-5 the number of smoothing steps is substantially higher on the coarser grids. Since the computational cost of smoothing is significantly decreased with each coarsening step, this greatly reduces the overall execution time of the resulting preconditioned solver. While EP-2 performs more smoothing on the second finest level, only a single step of red-black Gauss-Seidel is employed on the finest level. In both preconditioners, red-black Gauss-Seidel is predominantly used for smoothing. However, especially EP-5 also includes intermediate Jacobi smoothing steps or a combination of both methods. Finally, as shown in Figure 4, both preconditioners combine a wide range of different relaxation factors within their computation.

6 CONCLUSION AND FUTURE WORK

This work demonstrates how grammar-guided genetic programming (GGGP) can evolve multigrid preconditioners for Helmholtz problems that outperform known methods for different wavenumbers and even handle problems for which those methods fail. Despite this accomplishment, further research is needed to investigate under which circumstances the presented approach can achieve consistent results. We also aim to apply our approach to other multigrid variants, such as algebraic multigrid methods [56], and the solution of more challenging and complicated PDEs, such as nonlinear [23] and saddle point problems [3]. Furthermore, the resulting implementation is mainly limited by the compute resources required to evaluate a sufficient number of preconditioners. As a remedy, one could train a machine learning system to learn a model for predicting multigrid preconditioner performance based on the respective grammar representation. Another promising research direction, which has been already mentioned in the introduction, is the grammar-based construction of a multigrid method from those components obtained by a machine learning-based optimization such as [21, 24]. In addition, our approach could be utilized to accelerate the generation of training data for data-driven PDE solvers [28, 32, 33]. Finally, we aim to extend our implementation of GGGP to incorporate alternative initialization, crossover, and mutation operators, such as [10, 19, 44]. Also, while the implementation presented here employs tree-based GGGP, grammatical evolution (GE) [38, 45] represents a promising alternative, which could be as well integrated into our grammar-based approach for multigrid preconditioner design.

REFERENCES

[1] Xavier Antoine and Marion Darbas. 2016. Integral Equations and Iterative Schemes for Acoustic Scattering Problems. In *Numerical Methods for Acoustics Problems*, F. Magoulès (Ed.). Saxe-Coburg Editors. https://hal.archives-ouvertes.fr/hal-00591456

[2] Michele Benzi. 2002. Preconditioning Techniques for Large Linear Systems: A Survey. *J. Comput. Phys.* 182, 2 (2002), 418–477. https://doi.org/10.1006/jcph.2002.7176

[3] Michele Benzi, Gene H. Golub, and Jörg Liesen. 2005. Numerical solution of saddle point problems. *Acta Numerica* 14 (2005), 1–137. https://doi.org/10.1017/S0962492904000212

[4] F. J. Billette and S. Brandsberg-Dahl. 2005. The 2004 BP Velocity Benchmark. Article cp-1-00513. https://doi.org/10.3997/2214-4609-pdb.1.B035

[5] Achi Brandt. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* 31, 138 (1977), 333–390.

[6] William L. Briggs, Van Emden Henson, and Steve F. McCormick. 2000. *A Multigrid Tutorial* (second ed.). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719505

[7] Jed Brown, Yunhui He, Scott MacLachlan, Matt Menickelly, and Stefan M. Wild. 2021. Tuning Multigrid Methods with Robust Optimization and Local Fourier Analysis. *SIAM Journal on Scientific Computing* 43, 1 (2021), A109–A138. https://doi.org/10.1137/19M1308669

[8] Pierre-Henri Cocquet and Martin J. Gander. 2017. How Large a Shift is Needed in the Shifted Helmholtz Preconditioner for its Effective Inversion by Multigrid? *SIAM Journal on Scientific Computing* 39, 2 (2017), A438–A478. https://doi.org/10.1137/15M102085X

[9] Siegfried Cools and Wim Vanroose. 2013. Local Fourier analysis of the complex shifted Laplacian preconditioner for Helmholtz problems. *Numerical Linear Algebra with Applications* 20, 4 (2013), 575–597. https://doi.org/10.1002/nla.1881

[10] Jorge Couchet, Daniel Manrique, Juan Ríos, and Alfonso Rodríguez-Patón. 2007. Crossover and mutation operators for grammar-guided genetic programming. *Soft Computing* 11, 10 (01 Aug 2007), 943–955. https://doi.org/10.1007/s00500-006-0144-9

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[12] Yogi A. Erlangga. 2008. Advances in Iterative Methods and Preconditioners for the Helmholtz Equation. *Archives of Computational Methods in Engineering* 15, 1 (01 Mar 2008), 37–66. https://doi.org/10.1007/s11831-007-9013-7

[13] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. 2006. A Novel Multigrid Based Preconditioner For Heterogeneous Helmholtz Problems. *SIAM Journal on Scientific Computing* 27, 4 (2006), 1471–1492. https://doi.org/10.1137/040615195

[14] O. G. Ernst and M. J. Gander. 2012. *Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods.* Springer Berlin Heidelberg, Berlin, Heidelberg, 325–363. https://doi.org/10.1007/978-3-642-22061-6_10

[15] Vladimir Fanaskov. 2021. Neural Multigrid Architectures. In *2021 International Joint Conference on Neural Networks (IJCNN).* 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533736

[16] Radii Petrovich Fedorenko. 1962. A relaxation method for solving elliptic difference equations. *U. S. S. R. Comput. Math. and Math. Phys.* 1, 4 (1962), 1092–1096. https://doi.org/10.1016/0041-5553(62)90031-9

[17] Félix-Antoine Fortin, Simon Grenier, and Marc Parizeau. 2013. Generalizing the Improved Run-Time Complexity Algorithm for Non-Dominated Sorting. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation* (Amsterdam, The Netherlands) *(GECCO '13)*. Association for Computing Machinery, New York, NY, USA, 615–622. https://doi.org/10.1145/2463372.2463454

[18] Martin J. Gander and Hui Zhang. 2019. A Class of Iterative Solvers for the Helmholtz Equation: Factorizations, Sweeping Preconditioners, Source Transfer, Single Layer Potentials, Polarized Traces, and Optimized Schwarz Methods. *SIAM Rev.* 61, 1 (2019), 3–76. https://doi.org/10.1137/16M109781X

[19] M. García-Arnau, D. Manrique, J. Ríos, and A. Rodríguez-Patón. 2007. Initialization Method for Grammar-Guided Genetic Programming. In *Research and Development in Intelligent Systems XXIII*, Max Bramer, Frans Coenen, and Andrew Tuson (Eds.). Springer London, London, 32–44. https://doi.org/10.1007/978-1-84628-663-6_3

[20] Samuel H. Gray and Kurt J. Marfurt. 1995. Migration from topography: Improving the near-surface image. *Canadian Journal of Exploration Geophysics* 31, 1-2 (1995), 18–24.

[21] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. 2019. Learning to Optimize Multigrid PDE Solvers. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2415–2423. https://proceedings.mlr.press/v97/greenfeld19a.html

[22] Wolfgang Hackbusch. 1985. *Multi-Grid Methods and Applications.* Springer-Verlag.

[23] Van Emden Henson. 2003. Multigrid methods nonlinear problems: an overview. In *Computational Imaging*, Charles A. Bouman and Robert L. Stevenson (Eds.), Vol. 5016. International Society for Optics and Photonics, SPIE, 36 – 48. https://doi.org/10.1117/12.499473

[24] Ru Huang, Ruipeng Li, and Yuanzhe Xi. 2021. Learning optimal multigrid smoothers via neural networks. (2021). https://doi.org/10.48550/ARXIV.2102.12071

[25] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. 2021. Physics-informed machine learning. *Nature Reviews Physics* 3, 6 (01 Jun 2021), 422–440. https://doi.org/10.1038/s42254-021-00314-5

[26] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. 2020. Black-box learning of multigrid parameters. *J. Comput. Appl. Math.* 368 (2020), 112524. https://doi.org/10.1016/j.cam.2019.112524

[27] Harald Köstler, Marco Heisig, Nils Kohl, Sebastian Kuckuk, Martin Bauer, and Ulrich Rüde. 2020. Code generation approaches for parallel geometric multigrid solvers. *Analele Universitatii Ovidius Constanta - Seria Matematica* 28, 3 (2020), 123–152.

[28] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2021. Neural Operator: Learning Maps Between Function Spaces. (2021). https://doi.org/10.48550/ARXIV.2108.08481

[29] John R. Koza. 1994. *Genetic programming as a means for programming computers by natural selection.* Vol. 4. 87–112 pages. https://doi.org/10.1007/BF00175355

[30] H. Köstler and U. Rüde. 2004. Extrapolation Techniques for Computing Accurate Solutions of Elliptic Problems with Singular Solutions. In *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 410–417. https://doi.org/10.1007/978-3-540-25944-2_54

[31] Christian Lengauer, Sven Apel, Matthias Bolten, Shigeru Chiba, Ulrich Rüde, Jürgen Teich, Armin Größlinger, Frank Hannig, Harald Köstler, Lisa Claus, et al. 2020. ExaStencils: Advanced Multigrid Solver Generation. In *Software for Exascale Computing - SPPEXA 2016-2019*, Hans-Joachim Bungartz, Severin Reiz, Benjamin Uekermann, Philipp Neumann, and Wolfgang E. Nagel (Eds.). Springer International Publishing, Cham, 405–452. https://doi.org/10.1007/978-3-030-47956-5_14

[32] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier Neural Operator for Parametric Partial Differential Equations. (2020). https://doi.org/10.48550/ARXIV.2010.08895

[33] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. 2021. Physics-Informed Neural Operator for Learning Partial Differential Equations. (2021). https://doi.org/10.48550/ARXIV.2111.03794

[34] Peter Linz and Susan H. Rodger. 2022. *An Introduction to Formal Languages and Automata.* Jones & Bartlett Learning.

[35] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. 2020. Learning Algebraic Multigrid Using Graph Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 6489–6499. https://proceedings.mlr.press/v119/luz20a.html

[36] Gary S. Martin, Robert Wiley, and Kurt J. Marfurt. 2006. Marmousi2: An elastic upgrade for Marmousi. *The Leading Edge* 25, 2 (2006), 156–166. https://doi.org/10.1190/1.2172306

[37] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. 2010. Grammar-based Genetic Programming: a survey. *Genetic Programming and Evolvable Machines* 11, 3 (01 Sep 2010), 365–396. https://doi.org/10.1007/s10710-010-9109-y

[38] M. O'Neill and C. Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (2001), 349–358. https://doi.org/10.1109/4235.942529

[39] C. W. Oosterlee and R. Wienands. 2003. A Genetic Search for Optimal Multigrid Components Within a Fourier Analysis Setting. *SIAM Journal on Scientific Computing* 24, 3 (2003), 924–944. https://doi.org/10.1137/S1064827501397950

[40] Daniel Osei-Kuffuor and Yousef Saad. 2010. Preconditioning Helmholtz linear systems. *Applied Numerical Mathematics* 60, 4 (2010), 420 – 431. https://doi.org/10.1016/j.apnum.2009.09.003 Special Issue: NUMAN 2008.

[41] Allan D. Pierce. 2019. *Acoustics: An Introduction to Its Physical Principles and Applications.* Springer. 10.1007/978-3-030-11214-1

[42] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming.* Lulu Enterprises, UK Ltd.

[43] M. Raissi, P. Perdikaris, and G. E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[44] Pablo Ramos Criado, D. Barrios Rolanía, Daniel Manrique, and Emilio Serrano. 2020. Grammatically uniform population initialization for grammar-guided genetic programming. *Soft Computing* 24, 15 (01 Aug 2020), 11265–11282. https://doi.org/10.1007/s00500-020-05061-w

[45] Conor Ryan, Michael O'Neill, and J. J. Collins. 2018. *Handbook of Grammatical Evolution.* Springer. https://doi.org/10.1007/978-3-319-78717-6

[46] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems* (second ed.). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718003

[47] Sheppard Salon and M. Chari. 1999. *Numerical methods in electromagnetism*. Elsevier.

[48] Christian Schmitt, Sebastian Kuckuk, Frank Hannig, Harald Köstler, and Jürgen Teich. 2014. ExaSlang: A Domain-Specific Language for Highly Scalable Multigrid Solvers. In *2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. 42–51. https://doi.org/10.1109/WOLFHPC.2014.11

[49] Jonas Schmitt, Sebastian Kuckuk, and Harald Köstler. 2020. Constructing Efficient Multigrid Solvers with Genetic Programming. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 1012–1020. https://doi.org/10.1145/3377930.3389811

[50] Jonas Schmitt, Sebastian Kuckuk, and Harald Köstler. 2021. EvoStencils: a grammar-based genetic programming approach for constructing efficient geometric multigrid methods. *Genetic Programming and Evolvable Machines* (03 Sep 2021). https://doi.org/10.1007/s10710-021-09412-w

[51] Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. 2021. Optimization-Based Algebraic Multigrid Coarsening Using Reinforcement Learning. 34 (2021), 12129–12140. https://proceedings.neurips.cc/paper/2021/file/6531b32f8d02fece98ff36a64a7c8260-Paper.pdf

[52] A. Thekale, T. Gradl, K. Klamroth, and U. Rüde. 2010. Optimizing the number of multigrid cycles in the full multigrid algorithm. *Numerical Linear Algebra with Applications* 17, 2-3 (2010), 199–210. https://doi.org/10.1002/nla.697

[53] Ulrich Trottenberg, Cornelius W. Oosterlee, and Anton Schuller. 2000. *Multigrid*. Elsevier.

[54] Roelof Versteeg. 1994. The Marmousi experience: Velocity model determination on a synthetic complex data set. *The Leading Edge* 13, 9 (1994), 927–936. https://doi.org/10.1190/1.1437051

[55] Peter A. Whigham et al. 1995. Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, Vol. 16. 33–41.

[56] Jinchao Xu and Ludmil Zikatanov. 2017. Algebraic multigrid methods. *Acta Numerica* 26 (2017), 591–721. https://doi.org/10.1017/S0962492917000083