Learning Overlap Optimization for Domain Decomposition Methods

Steven Burrows[1], Jörg Frochte[2], Michael Völske[1],
Ana Belén Martínez Torres[2], and Benno Stein[1]

[1] Web Technology and Information Systems,
Bauhaus-Universität Weimar, 99421 Weimar, Germany
{steven.burrows,michael.voelske,benno.stein}@uni-weimar.de
[2] Electrical Engineering and Computer Science,
Bochum University of Applied Science, 42579 Bochum, Germany
{joerg.frochte,ana.martinez}@hs-bochum.de

Abstract. The finite element method is a numerical simulation technique for solving partial differential equations. Domain decomposition provides a means for parallelizing the expensive simulation with modern computing architecture. Choosing the sub-domains for domain decomposition is a non-trivial task, and in this paper we show how this can be addressed with machine learning. Our method starts with a baseline decomposition, from which we learn tailored sub-domain overlaps from localized neighborhoods. An evaluation of 527 partial differential equations shows that our learned solutions improve the baseline decomposition with high consistency and by a statistically significant margin.

Keywords: domain decomposition, numerical simulation.

1 Introduction

Numerical simulation in product development has become a standard. It is used in various applications such as semiconductor manufacturing [2], crash-test simulation [8], and fluidic system design [11]. Numerical simulation can also be supported by machine learning for the purpose of approximating solutions [14]. In this setting, a margin of error is tolerated in return for predictions from learned models that bypass on-the-fly simulation.

A key challenge in numerical simulation concerns the efficiency and stability of parameterized numerical simulation methods. These methods are often difficult to deploy for end-users. As a result, the most robust and parameter-*independent* methods (such as direct solvers for linear systems) are often preferred over the most efficient and parameter-*dependent* methods (such as domain decomposition and iterative solvers) in many engineering contexts. Our key idea is that the parameters in this latter group (for example, the overlaps of sub-domains when simulating partial differential equations) can be learned to converge the ease of use towards the parameter-independent methods. A consequential benefit is that the efficiency of the learned solutions can be increased.

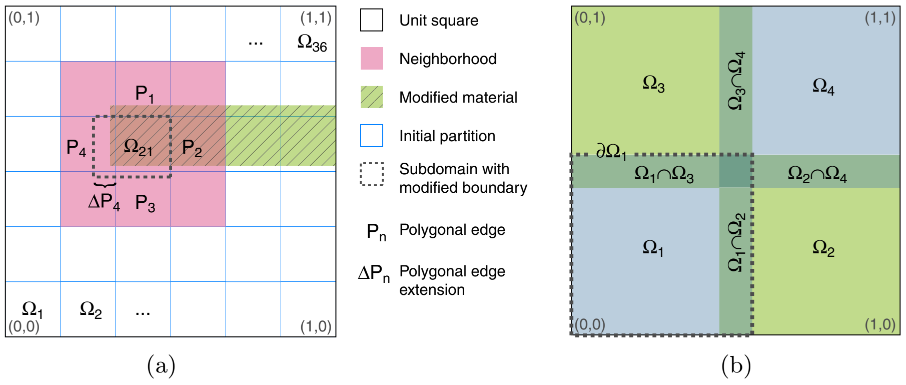


Fig. 1. (a) Our strategy for implementing a learnable domain decomposition problem. **(b)** A smaller problem showing overlaps (e.g., $\Omega_1 \cap \Omega_2$) and boundaries (e.g., $\partial\Omega_1$).

Parallelizing the simulation of models using domain decomposition in natural and engineering sciences is based on partial differential equations on a given domain. The approach requires the decomposition of a domain $\Omega \subset \mathbf{R}^m$, $m \in \{1, 2, 3\}$, into some number of sub-domains Ω_i, $i = 1..n \in \mathbf{N}$. For overlapping domain decomposition methods, the size of the sub-domain overlaps influence the stability and computational cost of the approach. Finding a near-optimal choice of parameters for optimizing the overlaps is an open problem. In many applications it is chosen with human intuition and experience using only a global setting. In this paper, we demonstrate that when using machine learning we can automate the choice of these parameters with local settings.

Our approach is to improve the efficiency of a default checkerboard sub-domain pattern such as the example shown in Figure 1a. Here, we consider the properties of the boundaries of each sub-domain as features. When dealing in two dimensions, each sub-domain Ω_i contains interesting relationships with the adjacent sub-domains $\{\Omega_i, i = 1..36 \mid \partial\Omega_{21} \cap \Omega_i \neq \emptyset\}$ that we may capture. This is represented in Figure 1a as the red *neighborhood* of nine sub-domains. In this example, a modified material setting passes through most of sub-domain Ω_{21}, and we should extend some of the sub-domain boundaries such that this material boundary is not too close to the initial partition. So for example, area boundary P_4 has been extended by some amount ΔP_4 so that the modified western sub-domain boundary is not too close to the material boundary.

For the purposes of machine learning, we are interested to learn the relative computational costs of the neighborhoods and then combine this knowledge to reduce the computational costs of the whole domain. To do this, the numerical simulation of the domain is computed for several variations of the neighborhood, and the relative improvement or degradation of the computational cost is recorded. So in the case of Figure 1a, we have 36 neighborhoods to consider including some interesting cases around the perimeter. The neighborhood features that we wish to capture include material regions that cross sub-domain boundaries or have close proximity to these boundaries. Therefore the mapping between these features and the computational cost can be cast as a regression problem, and we wish to learn solutions that reduce the cost.

Our contributions in this paper are summarized as follows: (1) We propose a machine learning approach for automating and optimizing the overlap construction for domain decomposition methods. (2) We create a unique problem set of interest to both novice and expert users. (3) We develop and apply a novel taxonomy of the feature space in our problem setting. (4) We devise a machine learning framework for overlap optimization including a training corpus and an appropriate performance measure.

The remainder of this paper is organized as follows. In Section 2 we provide the necessary background on partial differential equations and domain decomposition. In Section 3 we give the details of our methodology including data and the evaluation measure. In Section 4 we compare the results of our method to an expert human baseline. Finally in Section 5 we offer concluding remarks.

2 Solving PDEs Using Domain Decomposition

Numerical methods for solving any partial differential equation (PDE) [3] tend to involve a huge number of unknowns, especially in three dimensions. The power of a single computer is often no longer sufficient to solve the resulting equations. In this case, parallel computing with domain decomposition is one of the most successful strategies to solve these equations efficiently [10,12].

A PDE-based model consists of four components: the equations, the related parameter sets (or material properties), the domain these equations are solved on, and the boundary values given for the domain. The goal of domain decomposition is to split the domain into smaller sub-domains and iterate to coordinate the merging of the solution between these sub-domains.

Schwarz and other domain decomposition methods have overlapping and non-overlapping variants for solving boundary-value problems [10,12]. If a domain decomposition method is overlapping, then some portion of each problem is solved redundantly. Conversely if a domain decomposition method is non-overlapping, then the sub-domain boundaries are only touching. Most overlapping methods are categorized as additive or multiplicative, concerning the transfer data from one sub-domain to another. Additive methods have better properties concerning parallelization than multiplicative ones.

The overlapping additive Schwarz method, as utilized in this paper, is a simple and robust approach for applying domain decomposition. With sufficient overlap as demonstrated in Figure 1b, it can be applied to nearly every PDE. A disadvantage of this method is its slow convergence. For example, faster but more complicated non-overlapping methods such as FETI approaches [12] exist for some structural mechanic problems. These approaches can work without overlaps on the application domains, but they are less robust. They are not suitable, for example, for many fluid mechanic problems where overlapping Schwarz methods are also applicable. Nevertheless, all overlapping domain decomposition methods share the same basic demands and requirements that we want to solve in this paper, so our results are broadly applicable.

The parameters of domain decomposition are the positions of the introduced artificial boundaries and thus the size of the overlap. This means that the efficiency of domain decomposition is highly parameter-dependent since each new sub-domain creates artificial boundaries. An artificial boundary introduces errors arising from the domain decomposition procedure, which in turn leads to more iterations. The numerical effect of an artificial boundary diminishes when moving from the boundary to the inner part of a sub-domain. Thus, a bigger overlap leads to more information exchange between the sub-domains and therefore to artificial boundary conditions that are more closely related to the solution of the PDE, which leads to faster and more stable convergence. Beyond this, it is known that jumps in the material parameters influence convergence behavior if they occur next to the artificial boundaries.

In summary, a bigger overlap will decrease the number of iterations, however this will increase the size of the sub-domains and the computational overhead of the domain decomposition approach. This is a critical trade-off that influences the division of Ω into sub-domains. Beyond this trade-off, the number of computation units must be kept in mind when applying parallelization technology. For a computing cluster with m units, at least m sub-domains are desired, otherwise domain decomposition is not used to its fullest potential. However, we are less interested in obeying this technical constraint in this work, as our goal is to instead evaluate performance with a generalized strategy that is independent of specific computing infrastructure.

3 Overlap Optimization Learning Approach

In this section, we first describe our general problem specification, and the approaches for generating the associated data and feature sets. Following this, we provide the details of our evaluation measure and describe our learning objective together with the methodology that we deploy.

3.1 Problem Definition

As an example PDE for solving in this paper, we use Poisson's equation, which is a prototype of so-called elliptic PDEs of second order, with some Dirichlet boundary conditions:

$$-\varepsilon(x)\nabla^2 u = f(x) \text{ on } \Omega \text{ ; } u = g(x) \text{ on } \partial\Omega \tag{1}$$

This equation has application in modeling stationary heat, and we use it as an example to motivate our work. Consider Ω as the geometry on which we want so solve the heat equation such as a bar, $f(x) \geq 0$ as heat sources, $\varepsilon(x)$ as the material property, and $g(x)$ as known temperatures on the boundary $\partial\Omega$ of the domain Ω. A direct connection of two materials in a model could represent an ε-jump, and a blending of materials could represent a smoother ε-transition. We note that Poisson's equation is a quite simple PDE, but there are additional applications in Newtonian gravity and electrostatics, which is why we use it in this paper. The results concerning machine learning and domain decomposition achieved on this example can easily be transferred to other problems with a similar behavior, such as stress modeling used in engineering science. However, there are some models arising in fluid dynamics, for example, that behave differently concerning domain decomposition. These are less stable and need more care concerning parameter fitting. Transferring our approach to these models might need more work, but there are bigger benefits to gain because it is harder for humans to fit the model parameters.

For solving PDEs, domain decomposition can be applied to numerical methods such as spectral methods [5] and finite volumes [13]. We concentrate on the finite element method (FEM) [3], which is a standard method in most engineering software solutions. This problem is applied on the unit square using finite elements with continuous piecewise linear base functions on a regular triangulation, thus in Equation 1 we have $\Omega = [0, 1] \times [0, 1]$. Apart from the unit-square restriction, we only use rectangular partitioning in order to constrain the initial problem space. Notice that for the unit square with a structured grid, the checkerboard pattern with equal-size squares is a common and good default choice for the sub-domains. In this setting, the sub-domains all contain the same number of unknowns, and they have a good ratio between area and boundary.

3.2 Approach for Generating Diffusion Specifications

Each diffusion specification, or set of material values within the unit square to solve Poisson's equation, is a unique problem. This can imply that each problem must be learned individually and that results cannot carry over between different diffusion specifications. To address this, we are interested in learning patterns from neighborhoods of sub-domains as per Figure 1a, so that the knowledge about common patterns can be reapplied to whole problems. In this respect, we develop a deterministic algorithm for producing a large dataset of diffusion specifications for the neighborhood patterns to be learned from.

Our dataset is based on the placement of shapes with different material values in the domain, whereas in every shape the material value is constant. The shapes are circles and squares of various sizes that fit within the unit square or are truncated at the boundary. The shapes are arranged in one of three patterns with up to four shapes each: The Nested pattern uses a nested arrangement of shapes where the first shape is the largest and all successive shapes are included within. The Isolated pattern is comprised of stand-alone shapes without overlap or connection. The Sequence pattern uses different shapes arranged from a start point in a specific straight-line direction that can be just in contact or overlapping. The patterns, shape positions, and shape sizes are selected with a pseudo-random number generator with a deterministic sequence and fixed seed to make the data reproducible. Note also that the last-defined material setting takes precedence in the case of overlap. In general, one can expect that a skilled human will often be able to do a better job than machine learning for simple two-dimensional problems such as the Isolated case. But for more complicated problems in two dimensions (such as Nested and Sequence problems) and certainly in three dimensions, which is a typical application area for domain decomposition, machine learning will be helpful for both novice and expert users.

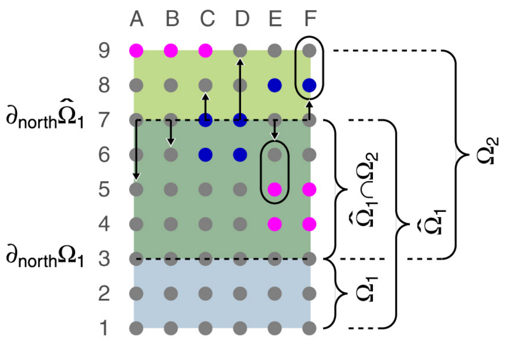
3.3 Approach for Generating Domain Specifications

Considering any checkerboard organization of sub-domains with uniform overlap, our goal is to improve on this baseline by learning from various permutations. Specifically, we can learn from permutations when the boundaries of the sub-domains are extended, retracted, or left alone in the north, east, south, and west directions. Since our goal is to learn from individual 9-region neighborhoods by modifying the boundaries of the central sub-domain, we can apply boundary modifications to each sub-domain in isolation. For example, when considering a uniform overlap of 0.4%, we can optionally modify the boundaries by ±0.2% to create three variations per sub-domain (0.2%, 0.4%, and 0.6%) when the boundaries are adjusted uniformly therefore creating $3 \times 16 = 48$ combinations for a 4×4 checkerboard. Other combinations are possible, such as adjusting boundaries in all individual combinations instead of uniformly, but we leave this for future work.

3.4 Extracting Features from Neighborhoods

In consideration of a 9-region neighborhood as per Figure 1a, we have developed features that capture interesting changes in the material setting ϵ around and within the overlapping region of a pair of sub-domains. Figure 2a provides an example for a modified sub-domain $\widehat{\Omega}_1$ and the northern overlapping region $\widehat{\Omega}_1 \cap \Omega_2$ with Ω_2. The example shows nine rows of unknowns surrounding the boundary $\partial_{north}\Omega_1$ and the modified boundary $\partial_{north}\widehat{\Omega}_1$. In this case the rows are of key interest but the columns are not, as they capture changes in the materials that are perpendicular to the overlapping regions, and extending or shrinking the overlapping region does not affect the measurement.

The features we are interested in come from a single-line or multi-line region immediately above or below the $\partial_{north}\widehat{\Omega}_1$ boundary as shown in Figure 2a. The number of lines in such a region is variable, and for now we simply consider two lines per region. For example, Figure 2a shows two lines for regions 'E' and 'F'. In this respect, we propose two feature sets called Fine and Coarse based on single-line and multi-line features respectively. We also propose a third feature set called Combined for Fine and Coarse together. For all three feature sets, we can capture various maximums, minimums, and differences between epsilon values within the regions of interest. Specifically, we capture (1) the maximum value in a region, (2) the minimum value in a region, (3) the maximum difference between values in a region, (4) the maximum difference between values in a region and the boundary, and (5) the minimum difference between values in a region and the boundary. Figure 2 provides a worked example where Fine is regions A–D (20 features), Coarse is regions E–F (10 features), and Combined



A
B
C
D
E
F
9
8
7
6
5
4
3
2
1
$\partial_{north}\widehat{\Omega}_1$
$\partial_{north}\Omega_1$
$\widehat{\Omega}_1 \cap \Omega_2$
$\widehat{\Omega}_1$
Ω_1
Ω_2

(a)

Region	Max Value in Region	Min Value in Region	Max Diff in Region	Max Diff to Boundary	Min Diff to Boundary
A	10 000	1 000	9 000	9 000	0
B	1 000	100	900	0	0
C	1 000	100	900	900	0
D	10 000	1 000	9 000	9 900	0
E	10 000	100	9 900	9 000	0
F	10 000	100	9 900	9 900	0

(b)

Fig. 2. (a) A fragment of a unit square for two overlapping sub-domains $\widehat{\Omega}_1$ and Ω_2. Notations 'A' to 'F' denote our features as the relationship between the rows referenced with arrows and the $\partial_{north}\widehat{\Omega}_1$ boundary. **(b)** Extracted values assuming the pink, gray, and blue unknowns take $\epsilon = 10\,000$, $\epsilon = 1\,000$, and $\epsilon = 100$ respectively.

is regions A–F (30 features). Then the full feature sets are realized when the eastern, southern, and western boundaries are processed.

3.5 The FPO Evaluation Measure

Developing a measure to evaluate the goodness of a sub-domain specification for any diffusion specification is non-trivial. Key variables such as the number of iterations and the amount of overlap have a complex relationship between one another, so these need to be carefully combined. Our approach should optimize the use of domain decomposition techniques that are designed to speed up simulation for parallel platforms. In a real-world application, a user is interested in optimizing the real-world time that a simulation needs, such as the chosen implementation, the computing network, and the use of cluster computing. Given this variability, we want to concentrate on the theoretical aspects of the algorithm together with a given abstract hardware scenario. Our goal is to minimize the number of floating point operations (FPO).

To justify this choice, first assume that we have a hardware architecture with s computation nodes. To use this architecture in an optimal way, let s also represent the number of sub-domains, n_i be the number of unknowns in a sub-domain, and l be the number of domain decomposition iterations. In a single iteration step, s linear equation systems have to be solved whereas the size of all equation systems is n_i. If one now assumes that a direct solver such as LU decomposition [9] is used, the first domain decomposition iteration of the matrix has the complexity of $O(n^3)$. For all remaining l iterations, one just has to solve one upper right and one lower left matrix of the complexity $O(n^2)$. Hence, FPO is defined as:

$$FPO \approx \sum_{i=1}^{s} \frac{n_i^3}{3} + l \cdot n_i^2 \ .$$

FPO is only meaningful when comparing solutions with the same number of sub-domains on the same hardware architecture.

3.6 Machine Learning Methodology

In order to learn from 9-region neighborhoods, we must simulate a large number of diffusion specifications with multiple sub-domain boundary settings in each neighborhood and derive the corresponding FPO scores. We do not simulate the neighborhoods of the unit square in isolation, because introduced additional artificial boundary conditions will influence the numerical behavior and so perturb the machine learning. Instead, the areas outside the neighborhood of interest are simply considered constant, and we are interested in relative changes to FPO when varying the sub-domain boundaries.

With a database of simulation results, we aim to predict the FPO scores for unseen neighborhoods with regression. Then we adopt the boundary recommendations that minimize FPO for each neighborhood and combine these to create a new solution. Using 527 diffusion specifications each having 48 domain decomposition permutations as per Section 3.3, the steps are as follows:

1. Training. For each diffusion file:
   (a) Perform feature extraction for all 48 permutations of the neighborhoods.
   (b) Compute FPO for all 48 permutations with simulation.
   (c) Record the mapping from the set of input features to FPO.
2. Testing. For each diffusion file:
   (a) Perform feature extraction for all 48 permutations of the neighborhoods.
   (b) Predict FPO for all 48 permutations using a regression model with the data from Step 1c.
   (c) Identify the minimum FPO value for each neighborhood.
3. Evaluation. For each diffusion file:
   (a) Create a new domain specification using the best results from Step 2c and compute FPO with simulation.
   (b) Compare the FPO score from Step 3a with that of the baseline.

The training and testing described above is performed with 10-fold cross-validation. A separate validation set of diffusion specifications was used during the development of our approach to avoid overfitting.

4 Analysis and Results

In this section we determine an expert human baseline, analyze our data, report improvements achieved by our method, and offer a forward plan with ideas to generate further improvements.

4.1 Baseline Overlap Decision

The baseline comparison for our methodology should be a human solution, since we are aiming to improve the FPO estimator from what humans can achieve. One solution is to apply a global overlap to all sub-domains. From our expertise of numerical simulation, the overlapping regions may consume up to around 5% of the available unknowns as a guideline. Guidelines are rarely given in the literature, but the work of Bjorstad and Hvidsten [1] provides one example based on 6%. An analysis of several checkerboard grid sizes and global overlap settings will guide the decision. The required data is given in Table 1 with our highlighted choice in bold. We adopt this choice because: (1) The 4×4 checkerboard gives a good mixture of center, boundary, and corner neighborhoods, (2) the 0.4% global overlap avoids extremes, and (3) it is compatible with the literature.

4.2 Data Analysis

We examined the data from all diffusion specifications from our validation and test sets to better understand the effectiveness of our feature sets. Partial results for 1 000 diffusion specifications are given in Table 2. The columns show the number of times the invalid (i.e., outside of unit square), no difference, default setting, and other measurements are observed. As shown, 25% of all values are invalid cases, which accounts for attempted measurements outside the unit square for a 4×4 grid — this effect diminishes for larger grids. We could give special consideration to boundary cases, but we leave this for future work for now, as many regression algorithms can handle missing values. We also see large numbers of no difference and default cases, however this redundancy is mitigated by the fact that our method uses vectors of measurements.

The other cases are where we can learn the most from. However, features (2) and (4) indicate that we do not have enough data for our problem setting, so we omit these. Also features (1) and (3) have the same number of measurements, and our analysis showed that these are correlated in almost all cases, so one should be omitted here too due to redundancy. In general, relative values (such as differences) are more interesting than absolute values (such as maximums and minimums), since the absolute values are dependent on the specific problem settings. With all the above considerations in mind, we only consider features (3) and (5), cutting the feature sets to 40% of those proposed in Section 3.4.

Table 1. Baseline choice considering experiment size, global overlap, and the literature

Global	Total overlap for various grid sizes (% of unknowns)							
overlap	1×1	2×2	3×3	$\mathbf{4 \times 4}$	5×5	6×6	7×7	8×8
minimum	0.00	0.40	0.80	1.19	1.59	1.99	2.38	2.77
0.2%	0.00	1.19	2.38	3.56	4.73	5.90	7.06	8.21
0.4%	0.00	1.99	3.95	**5.90**	7.82	9.73	11.62	13.48
0.6%	0.00	2.77	5.51	8.21	10.87	13.48	16.06	18.60
0.8%	0.00	3.56	7.06	10.49	13.85	17.16	20.40	23.57

Table 2. The feature extraction data demonstrates some redundancy in the feature sets. All values shown are for the "north" boundary and the "Fine A" region.

	Feature	Invalid	No Diff	Default	Other	Total
(1)	Max Value in Region	12 000	0	34 171	1 829	48 000
(2)	Min Value in Region	12 000	0	35 990	10	48 000
(3)	Max Diff in Region	12 000	34 171	0	1 829	48 000
(4)	Min Diff to Boundary	12 000	36 000	0	0	48 000
(5)	Max Diff to Boundary	12 000	35 361	0	639	48 000

4.3 Regression Algorithms and Feature Sets

We now compare the learned FPO scores with the baseline. We consider the three feature sets, Combined, Fine, and Coarse, and four regression algorithms, namely simple linear regression, nearest neighbor regression, decision tree regression, and support vector machine regression.[1] We found that only the nearest neighbor regression algorithm offered improvement. Since our interesting features are sparse (cf. Table 2), this indicates that we only have so many interesting near neighbors to learn from for each prediction, making nearest neighbor a good choice as the learning algorithm.

The median learned FPO scores for the nearest neighbor regression algorithm expressed as a fraction of the baseline are 0.9778 for Combined, 0.9791 for Fine, and 0.9830 for Coarse. This improvement is statistically significant in all cases ($p < 2.2 \times 10^{-16}$) when using a paired Student's t-test and the effect size is large (Cohen's $d = 0.85$ for Combined versus baseline, $d = 0.79$ for Fine versus baseline, and $d = 0.62$ for Coarse versus baseline). We also examined the differences between the features sets, but we found these of little interest as the effect sizes are small.

We must point out that our baseline is an expert human baseline, which we consider as the best baseline. In contrast, the novice baseline setting can be considered the minimum overlap comprising one line of unknowns. This baseline can lead to extreme behavior in many cases and simulation that does not converge in a reasonable amount of time. As a result we cannot compute this baseline, but we note that our approach does not exhibit the behavior of the novice baseline.

So far FPO is reduced to around 0.98 of the baseline and we have statistically significant improvements with large effect sizes. An explanation for this result is that our method provides a very consistent improvement for our test instances. For instance, Figures 3a and 3b show a shift in the score distributions leaving little overlap. Specifically, our method improves the baseline score for all instances except for 33 of 527 as shown in Figure 3c. We would like to further improve the cost saving to a 0.95 or 0.90 fraction to be completely satisfied, which we aim to achieve in future work with the following forward plan.

[1] Weka 3.7.7 [7]: weka.classifiers.functions.LinearRegression, weka.classifiers.lazy.IBk, weka.classifiers.trees.M5P, and weka.classifiers.functions.SMOreg respectively.

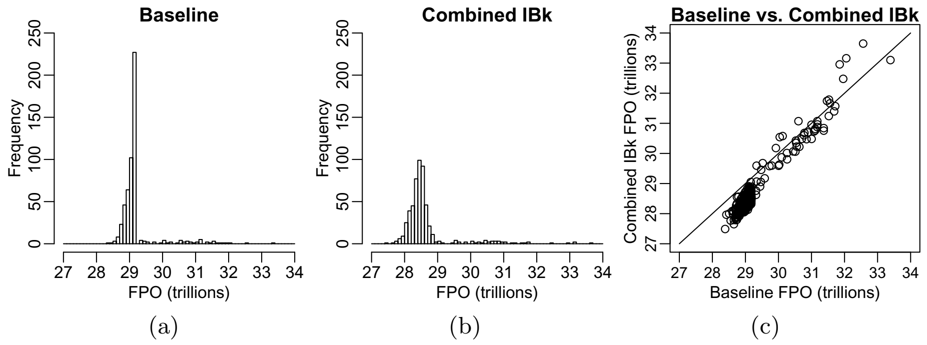


Fig. 3. The histogram shift and the scatterplot demonstrates consistent improvement

4.4 Forward Plan

The results presented above are our first for overlap optimization. An end goal is to consider three-dimensional problems later. First we wish to improve our approach for two-dimensional problems by implementing and experimenting with several extensions. For the first extension we wish to increase the checkerboard size for more fine-grained and precise learning. Second, we want to increase the training set size with additional diffusion specifications. Third, we wish to apply non-uniform boundary adjustments with sub-domains. Finally, we would like to drop the checkerboard constraint in favor of polygonal boundaries. We anticipate that these items each offer incremental improvements, and the final sum will be of most interest.

5 Conclusions

In this paper, we proposed a machine learning method for optimizing overlaps of domain decomposition problems. The key idea proposed was to learn properties of sub-domain neighborhoods, so that a complete solution can be assembled automatically and solved more efficiently with domain decomposition. To achieve this, our method introduced a novel feature set with the purpose of capturing interesting properties of sub-domain boundaries. When compared with an expert human baseline, our method offered a consistent and statistically significant improvement for the Poisson's equation. In addition, several avenues of future work have been identified that we expect will offer further improvements.

Finally, we emphasize that this work represents one part of a many-fold application of machine learning in a practical numerical simulation setting. Our earlier work in this field has demonstrated the behavioral learnability of bridge models [4] in an integrative structural design setting [6] for identifying robust solutions in civil engineering. Conversely, domain decomposition concerns parallelization for efficiency, so the bringing together of both dimensions provides potency for machine learning to have a high impact in numerical simulation.

References

1. Bjorstad, P., Hvidsten, A.: Iterative Methods for Substructured Elasticity Problems in Structural Analysis. In: Glowinski, R. (ed.) Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, pp. 301–312. SIAM, Paris (1987)
2. Brady, T.F., Yellig, E.: Simulation Data Mining: A New Form of Computer Simulation Output. In: Kuhl, M.E., Steiger, N.M., Armstrong, F.B., Joines, J.A. (eds.) Proceedings of the Thirty-Seventh Winter Simulation Conference, pp. 285–289. ACM, Orlando (2005)
3. Brenner, S.C., Scott, L.R.: The Mathematical Theory of Finite Element Methods, 2nd edn. Springer, Berlin (2002)
4. Burrows, S., Stein, B., Frochte, J., Wiesner, D., Müller, K.: Simulation Data Mining for Supporting Bridge Design. In: Christen, P., Li, J., Ong, K.L., Stranieri, A., Vamplew, P. (eds.) Proceedings of the Ninth Australasian Data Mining Conference, pp. 163–170. ACM, Ballarat (2011)
5. Canuto, C.G., Hussaini, M.Y., Quarteroni, A., Zang, T.A.: Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics, 1st edn. Scientific computation. Springer, Heidelberg (2007)
6. Gerold, F., Beucke, K., Seible, F.: Integrative Structural Design. Journal of Computing in Civil Engineering 26(6), 720–726 (2012)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1), 10–18 (2009)
8. Mei, L., Thole, C.A.: Data Analysis for Parallel Car-Crash Simulation Results and Model Optimization. Simulation Modelling Practice and Theory 16(3), 329–337 (2008)
9. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: LU Decomposition and its Applications. In: Numerical Recipes in Fortran: The Art of Scientific Computing, 2nd edn., pp. 34–42. Cambridge University Press, Cambridge (1992)
10. Quarteroni, A., Valli, A.: Domain Decomposition Methods for Partial Differential Equations, 1st edn. Numerical Mathematics and Scientific Computation. Oxford Science Publications, New York City (1999)
11. Stein, B., Curatolo, D.: Selection of Numerical Methods in Specific Simulation Applications. In: del Pobil, A.P., Mira, J., Ali, M. (eds.) IEA/AIE 1998. LNCS, vol. 1416, pp. 918–927. Springer, Heidelberg (1998)
12. Toselli, A., Widlund, O.: Domain Decomposition Methods – Algorithms and Theory., 1st edn. Springer Series in Computational Mathematics, vol. 34. Springer, Heidelberg (2004)
13. Versteeg, H.K., Malalasekera, W.: An Introduction to Computational Fluid Dynamics: The Finite Volume Method, 2nd edn. Pearson Education, Essex (2007)
14. Yang, S.-H., Hu, B.-G.: Reformulated Parametric Learning based on Ordinary Differential Equations. In: Huang, D.-S., Li, K., Irwin, G.W. (eds.) ICIC 2006. LNCS (LNAI), vol. 4114, pp. 256–267. Springer, Heidelberg (2006)