



AFRL-RI-RS-TR-2019-218

DESIGN SPACE EXPLORATION FOR CYBER PHYSICAL SYSTEMS

PERSPECTA LABS

NOVEMBER 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2019-218 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

STEVEN DRAGER

Work Unit Manager

/ S /

DONALD TELESKA

Acting Technical Advisor

Computing & Communications Division

Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) NOVEMBER 2019		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) SEP 2018 – JUN 2019	
4. TITLE AND SUBTITLE DESIGN SPACE EXPLORATION FOR CYBER PHYSICAL SYSTEMS				5a. CONTRACT NUMBER FA8750-18-C-0184	
				5b. GRANT NUMBER N/A	
				5c. PROGRAM ELEMENT NUMBER 61101E	
6. AUTHOR(S) Sanjai Narain Emily Mak Todd Huster Jeremy Cohen Kishore Pochiraju Brendan Englot Niraf Jha				5d. PROJECT NUMBER BRAS	
				5e. TASK NUMBER SV	
				5f. WORK UNIT NUMBER EN	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Perspecta Labs 150 Mount Airy Road Basking Ridge, NJ 07920				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505 DARPA 675 North Randolph St Arlington, VA 22203-2114				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2019-218	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. PA# 88 ABW-2019-5681 Date Cleared: 20 Nov 2019					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The goal of this effort was to evaluate the feasibility of creating a toolkit for design-space exploration of cyber physical systems and identify toolkit success metrics. This toolkit needed to accomplish tasks, including: synthesis or inverse design, composition of heterogeneous models, alleviating curse of dimensionality, minimizing blackbox function evaluation, resilience to blackbox evaluation failure, non-parametric synthesis, solution to nonlinear constraints and optimization. This research proposed Constraint satisfaction with Neural networks, Mixed integer linear programming (MILP) and Active learning (CNMA), a new method of solving nonlinear constraints on nonlinear functions. CNMA is based on the idea that the constraints can be solved by learning the function as a neural network, converting it into an equivalent MILP, and solving with industrial-strength MILP solvers. Since learning is always approximate, an incorrect solution can be returned. CNMA adds a new error-correction step that assures solution correctness. Additionally, it focuses learning of the function only on that part of its domain that is relevant to solving the constraint. Thus, the learning is reduced by orders of magnitude over the case where the function has to be learnt in its entire domain.					
15. SUBJECT TERMS cyber physical systems, inverse design, synthesis, nonlinear constraint solving, blackbox optimization, neural networks, mixed integer linear programs, active learning, gaussian processes, wave glider, wave energy propulsion, power grid fault location					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 65	19a. NAME OF RESPONSIBLE PERSON STEVEN DRAGER
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) (315) 330-2735

Table of Contents

List of Figures	iii
List of Tables	iv
Executive Summary	1
1 Introduction	5
1.1 What Are Limitations of Current Practice?	6
1.2 Overview of CNMA	8
1.3 Creating CPS Design-Space Exploration Toolkit With CNMA	9
1.4 Evaluating CNMA	12
2 Methods Assumptions and Procedures	13
2.1 CNMA With Single Function	13
2.2 CNMA With Multiple Functions	15
2.3 Designing a Watercraft Propelled Solely With Wave Energy	17
2.3.1.1 Definition of Hydrofoil Function	18
2.3.2 Definition of Boat Function	21
3 Results and Discussions	22
3.1 Synthesis of Hydrofoil With Single Function CNMA	22
3.2 Synthesis of Watercraft With Multiple Function CNMA	28
3.3 Evaluating CNMA Performance	29
3.4 Non-Parametric Synthesis And Optimization	30
3.5 Designing Water Craft With Traditional Optimization Methods	32
3.5.1 Design with Gaussian Processes	33
3.5.2 Design With COBYLA	33
3.6 Synthesizing Logical CPS	34
3.6.1 Parallel Parking Maneuver	34
3.6.2 Robot Path Planning	35
3.7 Synthesizing CPS With Discrete and Continuous Variables	37
3.8 Sensitivity of Traditional Optimization Methods to Starting Points	41
3.9 Solving Nonlinear Constraints	42

3.10	<i>Using Neural Network Compression With CNMA</i>	47
4	Conclusions	51
5	Recommendations	54
6	References	55
7	List of Symbols, Abbreviations, and Acronyms	57
8	Glossary	58

List of Figures

Figure 1. CNMA Overview.....	8
Figure 2. Boat Propelled Solely With Wave Energy.....	12
Figure 3. CNMA With Single Function.....	13
Figure 4. CNMA With Multiple Functions.....	15
Figure 5. Structure of Wave Energy Propelled Watercraft With Recursive Relationships ...	17
Figure 6. Hydrofoil Forces	18
Figure 7. xfoil Screenshot.....	20
Figure 8. Hydrofoil Forces Repeated.....	22
Figure 9. CNMA Iteration 1.....	25
Figure 10. CNMA Iteration 2	25
Figure 11. CNMA Iteration 3	26
Figure 12. CNMA Iteration 4.....	26
Figure 13. CNMA Iteration 5	27
Figure 14. CNMA Performance	29
Figure 15. Finding Parameters of Sigmoid Curve For Safe Parallel Parking With Moving Obstacles	34
Figure 16. Learning <i>safe</i> (x_0, y_0, x_1, y_1).....	35
Figure 17. Safe Path Computed By CNMA	36
Figure 18: Test circuit diagram. G1 is a single generator, Z1-3 are loads, M1-8 are metering devices, and L1-4 are power lines (with status 0/1). M1, for example, represents (i1,v1l), and M2 indicates (i1,v1r).....	37
Figure 19. Plot of $x * \sin(x)$	41
Figure 20. CNMA Performance With and Without Compression	47
Figure 21. Number of CNMA Iterations With Different Types of SCANN Tuning.....	48
Figure 22. Average Training Time With and Without SCANN Tuning	49
Figure 23. Times Taken To Solve Water Craft Design Query With Compression.....	50

List of Tables

Table 1. Hydrofoil Training Set	23
Table 2: Training Data Inputs	38
Table 3: Training Data Outputs	39

Executive Summary

The goal of this effort was to evaluate the feasibility of creating a toolkit for design-space exploration of cyber physical systems (CPS) and identify metrics for evaluating these. CPS are marked by the presence of discrete and continuous inputs and outputs usually governed by complex nonlinear relationships. CPS need not be just physical. They can also be logical e.g., parallel parking or obstacle avoiding robot maneuvers. Such a toolkit would allow a designer to specify requirements on design and performance. It would then automatically suggest designs satisfying requirements. It would also allow exploration of design tradeoffs. Tasks that such a toolkit needs to accomplish include:

- Synthesis, a.k.a. “inverse design”
- Composition of heterogeneous CPS subsystem models
- Scalability a.k.a. alleviating curse of dimensionality
- Minimizing number of calls to expensive design/blackbox tools
- Resilience to design tool failure
- Non-parametric synthesis
- Solving nonlinear constraints
- Computing global optima

Metrics for evaluating CPS design-space exploration toolkits can be based on how well they accomplish these tasks.

Most relevant to building such a toolkit are the mature nonlinear optimization tools. These optimize an objective function subject to constraints. However, these have several shortcomings:

- Derivative-based methods [28, 20] are infeasible when the cost of evaluating functions is high. Derivative computation can dramatically increase the number of function evaluations.
- Derivative-free methods [29, 31, 24] are conservative in the number of times they evaluate functions. They build surrogate functions based on current samples and use those to predict the best point to sample next. But, their complexity is high, for example that of Gaussian process is $O(n^3)$ where n is the sample size [24].
- Derivative-free methods are not resilient to design tool failure. The information gaps caused by such failures disrupt surrogate construction. Such methods are also inadequate as constraint solvers.
- Constrained optimization accomplishes constraint solving only indirectly by reducing it to unconstrained optimization. This is done by use of penalty or barrier functions encoding the cost of violating constraints. However, construction of such functions requires considerable creativity on the part of the user [17, 27, 20].
- Optimization methods that require starting points are highly sensitive to their choice. It is hard to find good starting points in high-dimensional spaces.

- Interior point optimization methods require the starting point to be within the feasible region – in other words, they need to solve constraints before even beginning.
- There is no obvious way of specifying recursive relationships. Such specification can be avoided by flattening the design space into a high-dimensional one, but that gives rise to the curse of dimensionality: we have to solve a single optimization problem, rather than being able to break it up into simpler ones and combining their solutions.

This research proposes Constraint Satisfaction with Neural networks, Mixed Integer Linear Programming (MILP) and Active learning (CNMA), a new approach for building such a toolkit, which overcomes the above shortcomings. CNMA solves the fundamental problem of finding x, y such that $F(x) = y \wedge P(x, y)$ where F is a possibly nonlinear function modeling a CPS, x is a design vector of discrete and continuous values, y is the associated output vector of discrete and continuous values and P is a possibly nonlinear design constraint on both x, y . CNMA is based on the idea that the constraint can be solved by learning F as a neural network, converting it into an equivalent Mixed Integer Linear Program, and solving it along with P with industrial-strength mixed integer linear program solvers. However, since learning is always approximate, the solver can return incorrect results. Adding a new error-correction step focusses the learning on that part of F 's domain that is relevant to solving the constraint. Thus, the learning is reduced by orders of magnitude.

CNMA is evaluated as follows:

- We use it to explore designs of a water craft propelled solely with wave energy. It is a moderately high-dimensional problem with 13 design variables. Such design requires solutions to all of the problems listed above. We show how CNMA can efficiently accomplish those tasks. In particular, a design was obtained in about 30 seconds with ~ 100 calls to the water craft simulators. To create a sufficiently accurate neural network model of the water craft without CNMA would require $\sim 10^6$ calls to these simulators. The water craft is inspired by the Wave Glider [6, 8].
- We formulate the water craft design problem with two mature nonlinear, derivative-free optimization methods: Gaussian Processes [24] and Constrained Optimization by Linear Approximation (COBYLA) [31], and show that CNMA's performance in terms of several metrics, time to compute design, number of function calls and resilience to design tool failure, is far superior. Gaussian Processes did not even terminate. COBYLA did but took about 1348 calls to the simulators.
- We do *not* evaluate derivative-based methods as they greatly increase the number of function evaluations.
- We further evaluate CNMA against traditional nonlinear optimization benchmarks. One of these has constraints on transcendental functions of quadratic expressions [30]. When functions to be optimized are amenable to mathematical analysis, current methods significantly outperform CNMA. However, because they are also sensitive to starting points, they can sometimes fail to produce answers, whereas CNMA will produce an answer if it exists.
- We show how, despite using an MILP solver (where L stands for Linear), CNMA allows P to be nonlinear. This is accomplished by adding, for each nonlinear expression in P , an

extra output to the neural network for F and then expressing P as a linear constraint on these outputs.

- We show how CNMA can be used to synthesize logical CPS such as parallel parking and obstacle avoiding robot maneuvers. Such problems are traditionally solved with temporal-logic based synthesis engines. These examples show we might reduce them to non-temporal constraint solving by abstracting away time.
- We show how CNMA can be used to solve the synthesis problem for a CPS with both discrete and continuous variables. This problem is that of identifying a failure pattern in a smart grid that is consistent with sensor readings at a limited number of points in the smart-grid.

New research directions we have identified are:

- **Scalability.** While CNMA can efficiently design a CPS with 13 design variables, what innovations, if any, are needed to scale the size by at least an order of magnitude to e.g., 100 variables?
- **Combining strengths of existing optimization tools with CNMA.** For example, CNMA could be used to find feasible starting points for interior-point methods. CNMA could also be integrated into Satisfiability Modulo Theories (SMT) solvers and temporal-logic based synthesis tools, both of which need nonlinear constraint solving.
- **Extending CNMA to neural networks with general activation functions.** At present, CNMA uses neural networks with the Rectified Linear Units (ReLU) activation function to permit their transformation into equivalent MILPs. When other activation functions such as tanh or sigmoid are more appropriate, we could use traditional nonlinear solvers while retaining the benefits of CNMA.
- **Extending CNMA to new neural network architectures.** At present, only fully connected, static neural networks are used. What new CPS designs are possible if we could use Convolutional Neural Networks, Long Short-Term Memory, or Recurrent Neural Networks e.g., [21]?
- **Automatically identifying hyper parameters.** At present, neural network hyper parameters, as well as the size of the initial training set are manually selected. Can we automatically identify these?
- **CNMA with Diversity.** Can we use an ensemble of neural networks with different hyper parameters to accelerate solution convergence? Each neural network may learn a different aspect of the function.
- **UNSAT-Core.** At present, CNMA does not terminate if the constraint is unsatisfiable. It keeps learning new points in the domain of the function in the hopes that a better approximation would let it solve the constraint. How do we overcome this significant limitation of CNMA?
- **Fluent user interfaces.** Can we make the design-space experience fluent e.g., in Massachusetts Institute of Technology (MIT) Northstar system [26]?
- **Identifying new application domains.** We are talking to experts in satellite design and propulsion. All we need are simulators for a domain that map designs into outputs. Then, CNMA can assist with design-space exploration.

- **Using neural network compression.** Compression of a neural network produces another whose size is much smaller than that of the original, but whose accuracy is the same. The MILP equivalents of smaller neural networks are much easier to solve. We have tried to use one such compression method Synthesis of Compact and Accurate Neural Networks (SCANN) [1] but found that while the compressed neural network was solved in about the same time as the original, the overhead of compression at each CNMA iteration was considerable. Thus, the current implementation of CNMA does not use compression. This may change as we deal with much larger neural network for realistic CPS. For example, the Neural Network Synthesis Tool (NeST) system compresses images [11].

1 Introduction

Our goal was to evaluate the feasibility of creating a toolkit for design-space exploration of cyber physical systems and identify metrics for evaluating these. CPS are marked by the presence of discrete and continuous inputs and outputs. CPS need not be just physical. They can also be logical e.g., parallel parking or obstacle avoiding robot maneuvers. Such a toolkit would allow a designer to specify requirements on design and performance and automatically suggest designs and allow exploration of design tradeoffs. Tasks that such a toolkit needs to accomplish include:

- **Synthesis.** By synthesis is meant a solution to the “inverse design” problem. Design tools work in the forward direction mapping designs into performance. We’d like to compute the inverse mapping. Often, the design tools are only available as a blackbox, such as a simulator, in which the underlying math is not available for analysis. How do we compute the inverse mapping?
- **Composition of heterogeneous subsystems.** A CPS can consist of a number of subsystems. Relationships between these can be mutually recursive. How do we design subsystems separately and compose resulting designs with assurances of global mutual consistency? Note that subsystems can be heterogeneous in that there may not exist a single tool for designing all subsystems.
- **Scalability.** A realistic CPS can be defined by a very large number of design and performance variables. How do we efficiently explore the resulting astronomically large design spaces?
- **Minimizing number of calls to expensive design tools.** Design tools can often take a very long time to compute performance for a given design. How do we minimize the number of times these are called while searching for a design satisfying requirements?
- **Resilience to design tool failure.** Sometimes design tools, such as computational fluid dynamics simulators, fail to compute performance for a given a design. How do we accommodate the resulting gaps in information in the search for designs?
- **Non-parametric synthesis.** Given an overall CPS design requirement, how do we automatically compute the components and the architectures into which they are to be integrated? Parametric synthesis where components and architectures are known, is hard enough.
- **Solving nonlinear constraints.** CPS design and performance variables are often governed by complex, nonlinear constraints. How do we efficiently solve such constraints?
- **Computing global optima.** How do we compute global optima in the face of above challenges?

1.1 What Are Limitations of Current Practice?

Most relevant to building a design-space exploration toolkit are the mature nonlinear optimization tools. These optimize an objective function subject to constraints. However, these have several shortcomings:

- Derivative-based methods are infeasible when the cost of evaluating functions is high as derivative computation can dramatically increase the number of function evaluations.
- Derivative-free methods are conservative in the number of times they evaluate functions. They build surrogate functions (e.g., linear, quadratic, polynomial, radial-basis function (RBF), Gaussian) based on current samples and use those to predict the best point to sample next, till optimization halting criteria are satisfied. But, their complexity is high, for example that of Gaussian process is $O(n^3)$ where n is the sample size [24]. Kriging is a Gaussian process-based optimization method that, in turn, is an example of a Bayesian optimization method.
- Derivative-free methods are not resilient to design tool failure. The information gaps caused by such failures disrupt surrogate construction. Such methods are also inadequate as constraint solvers.
- Optimization methods that require starting points are highly sensitive to their choice. It is hard to find good starting points in high-dimensional spaces.
- Interior point optimization methods require the starting point to be within the feasible region – in other words, they need to solve constraints before even beginning.
- There is no obvious way of specifying recursive relationships. Such specification can be avoided by flattening the design space into a high-dimensional one, but that gives rise to the curse of dimensionality: we have to solve a single optimization problem, rather than being able to break it up into simpler ones and combining their solutions.
- They are inadequate as constraint solvers. Constrained optimization methods do constraint solving only indirectly by reducing it to unconstrained optimization by use of penalty functions that drastically increases the value of the objective function if the constraint is violated. For example, to model the constraint $x \leq 1$, the term $-(\log(1 - x))$ is added to the objective function. Whenever $x > 1$, the term would evaluate to infinity so the optimization engine would search away from that point. However, formulating good penalty functions requires considerable creativity on the part of the designer. Here are some observations on penalty functions from [17,27]:
 - The goal of penalty functions is to convert constrained problems into unconstrained problems by introducing an artificial penalty for violating the constraint. Because exterior penalty functions start outside the feasible region and approach it from the outside, they only find extremes that occur on the boundaries of the feasible region. *They will not find interior extremes.* In order to accomplish that, these are often used in combination with interior penalty functions
 - A second type of penalty function begins with an initial point inside the feasible region, which is why these procedures are called interior penalty functions or “barrier methods”. Before beginning, all constraints must be converted into constraints of the form $g(x) \leq 0$. *This method will not work with equality constraints.*

- Penalty methods are not just plug-and-play! There are many variables involved, including the initial value of the objective function coefficient, how fast this coefficient grows, the starting point, and the step size in the minimizer. In an actual problem, these variables are tried in different combinations, and typically both interior and exterior methods are used.

The Automatic Learning of Algebraic Models (ALAMO) [33] system approach presents another option to learning complex functions using integer-programming and adaptive sampling. The main idea of the ALAMO method is to learn an algebraic expression that can approximate the behavior of an arbitrary nonlinear function. First, nonlinear transformations are performed on initial data to create a regression design matrix. Then, a linear model of the transformed data that optimizes a chosen model fitness metric is found using Mixed Integer Quadratic Programming. The model fitness metric must express a trade-off between model accuracy and over-fitting by incorporating the sum of squared residuals and penalizing the number of non-zero regression coefficients. Next, adaptive sampling is used to improve the model. The error maximization sampling (EMS) approach finds samples that maximizes $\left(\frac{z(x)-z'(x)}{z(x)}\right)^2$ where $z(x)$ is the function being learned, and $z'(x)$ is the algebraic model of $z(x)$. Because $z(x)$ is unknown, derivative free methods must be used to find the optimal samples. By sampling the unknown function where the model is inaccurate, EMS has been shown to need less data than other sampling methods like Latin Hypercube Sampling. The samples found can also be used as an estimate of the true relative error squared, and if the actual relative error is less than a set tolerance, the algorithm ends. Otherwise, the samples are added to the training set and the model is rebuilt. This process continues until the samples found have a true error less than the specified tolerance.

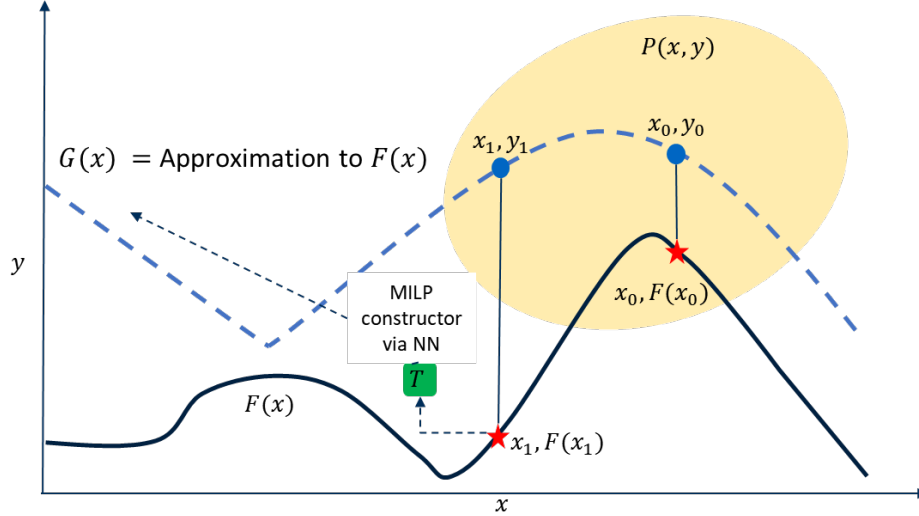
Both CNMA and ALAMO try to learn a function using iterative processes where samples are added to a training set to get better approximations, and both algorithms rely on integer programming. However, the main differences between CNMA and ALAMO is the type of surrogate model, and sampling method used.

While CNMA uses neural networks as the model, ALAMO uses algebraic functions.

Additionally, CNMA is query driven, and only tries to sample the space relevant to the query, while ALAMO uses error maximization sampling to *find an accurate algebraic model over the whole domain*.

1.2 Overview of CNMA

We present a new method called CNMA for building a CPS design-space exploration toolkit. It solves the fundamental problem of finding x, y such that $F(x) = y \wedge P(x, y)$ where F is a function modeling a CPS, x is a design vector of discrete and continuous values, y is the associated output vector of discrete and continuous values, P is a design constraint on both x, y and F and P can both be nonlinear.



To find x, y : $F(x) = y \wedge P(x, y)$

- T is the training set for learning $G(x)$, a neural network as MILP approximation to $F(x)$
- The yellow region is $\{(x, y) \mid P(x, y)\}$
- Let x_0, y_0 be a solution to $P(x, y) \wedge G(x) = y$. If $P(x_0, F(x_0))$, output $(x_0, F(x_0))$ and stop
- Let x_1, y_1 be a solution to $P(x, y) \wedge G(x) = y$. If $\neg P(x_1, F(x_1))$, add $(x_1, F(x_1))$ to T and retry
- If no solution found, select z , add $(z, F(z))$ to T and retry

Figure 1. CNMA Overview

We assume that a tool for evaluating F for any x is available, e.g., as a simulator. As shown in Figure 1, CNMA evaluates F for different samples in the domain of F and records associated outputs to form training instances. These training instances are used to learn an approximation to F in the form of a neural network, with the ReLU activation function. The neural network is then converted into an equivalent Mixed Integer Linear Program. This program is solved along with $P(x, y)$ to produce values of x and y , say x_0, y_0 respectively. CNMA then checks whether $P(x_0, F(x_0))$ holds and if so outputs as solution $x = x_0, y = F(x_0)$. Otherwise, CNMA adds $(x_0, F(x_0))$ to the training set and retries. If the MILP solver fails to produce a solution, CNMA generates a new training instance by selecting a point z in the domain of F , adding $(z, F(z))$ to the training set, and retrying.

The addition of $(x_0, F(x_0))$ is a form of learning from error that has the effect of learning only that part of the domain of F that is relevant to solving $P(x, y)$. In all the examples we have tried, the sampling of F is reduced by many orders of magnitude over the fine-grained sampling needed to learn F over its entire domain.

1.3 Creating CPS Design-Space Exploration Toolkit With CNMA

We sketch how CNMA can be used to accomplish the design-space exploration tasks.

Synthesis. As CNMA's focus is constraint solving, not optimization, it can solve the inverse-design problem. Constraint solvers, in particular, MILP solvers do not distinguish between inputs and outputs. Given constraints/requirements on input and output variables, constraint solvers try to find values of variables satisfying those.

We show how CNMA can be used to synthesize logical CPS such as parallel parking and obstacle avoiding robot maneuvers. Such problems are traditionally solved with temporal-logic based synthesis engines. These examples show that we might be able to abstract away time and reduce the synthesis problem to non-temporal constraint solving.

Composition. Design composition is accomplished by composing the MILP representations of subsystems. MILPs can be composed with Boolean operators such as conjunction. Global consistency is assured by solving the composed MILP. Any solution that satisfies the composed (via conjunction) MILP satisfies each of the constituent MILPs. As there is no notion of input or output variables, there is nothing special about modeling recursive relationships.

Scalability. For scalability, CNMA leverages the above composition plan and the scalability of industrial-strength MILP solvers that can efficiently solve a large number of constraints in a large number of variables. CNMA allows modeling a CPS as a collection of subsystems, each with a smaller number of inputs, and composing their MILP equivalents. It is much more feasible to learn subsystems with a small number of inputs over learning a single system with a large number of inputs. CNMA then lets the MILP solver deal with solving the composition of the constituent MILPs.

Minimizing number of calls to expensive design tools. As CNMA learns that part of the design space relevant to solving the constraint, the number of evaluations of F is greatly reduced. We do not have a proof of minimality, but all empirical studies have shown that the reduction is by orders of magnitude e.g., from 10^6 to about 200.

Handling design tool failure. If for some x , $F(x)$ does not terminate, CNMA just finds another x and restarts. The "surrogate function" that CNMA constructs is a neural network whose construction is resilient to missing information.

Solving nonlinear constraints. CNMA does not approximate constraint solving as optimization. Instead it uses a constraint solver, proper, in the form of an MILP solver. It constructs a sequence of MILPs each of which is a surrogate for F in the part of the domain relevant to solving P . CNMA leverages two important results. The first is the Universal Approximation Theorem that states any continuous function can be approximated, with arbitrary accuracy, with a neural network with just one hidden layer. Second, any neural network with the ReLU activation function can be converted into an equivalent MILP [5]. These results, combined with CNMA's novel idea of using the solution to the current MILP to improve the next MILP leads to rapid convergence to the correct solution.

CNMA has no notion of a starting point (solution), so the question of being sensitive to such a point does not arise. In fact, it is possible to use CNMA to compute a starting solution for interior point optimization methods that need to start inside a feasible region. CNMA is, however, sensitive to the size of the initial training set and the neural network architecture used.

Non-parametric synthesis. The choice of different architectures for satisfying a design requirement can be modeled with disjunction. CNMA would then find an architecture subject to the requirements.

Computing global optima. CNMA can use the optimization capabilities of the MILP solver to compute optima, in particular, to solve minimize $\phi(x, y)$ s.t. $F(x) = y \wedge P(x, y)$. Find the optima at each iteration, using the MILP equivalent of the F 's neural network surrogate, and select the best after a maximum number of iterations:

```

Best_solution = None
Best_objective = inf
For i=1:max_iterations
    Use NN/MILP to find  $x_0$  which minimizes  $\phi(x, y)$  s.t.  $F(x) = y \wedge P(x, y)$ 
    If MILP is infeasible, choose random point
    If  $\phi(x_0, F(x_0)) \leq \text{Best\_objective} \wedge P(x_0, F(x_0))$ :
        Best_solution =  $x_0$ 
        Best_objective =  $\phi(x_0, F(x_0))$ 
    Add  $(x_0, F(x_0))$  to the training set
Return Best_solution

```

This approach is straightforward and mimics that followed by traditional derivative-free optimization methods.

Another option is Binary Search where we maintain an upper and a lower bound on the objective function, and decrease its size by half at each iteration. However, we have to account for the fact that CNMA may not terminate for an unsolvable query. One possibility is to declare unsatisfiability (UNSAT) if no solution is found after a fixed number of iterations, considering we are dealing with blackbox optimization.

Let L be a lower bound on $\phi(x, y)$, and U be an upper bound on $\phi(x, y)$

- Use CNMA to find a solution to $F(x) = y \wedge P'(x, y)$
- 1. Where $P'(x, y) = P(x, y) \wedge \phi(x, y) \leq L$
 - o If solution $(x_0, F(x_0))$ is found, return $(x_0, F(x_0))$
- Use CNMA to find a solution to $F(x) = y \wedge P'(x, y)$
- 2. Where $P'(x, y) = P(x, y) \wedge \phi(x, y) \leq U$
 - o If solution $(x_0, F(x_0))$ is found, let $U = \phi(x_0, F(x_0))$
 - o Else, return Unsat
- While $U - L \geq \text{tol}$

- o $m = \frac{(L+U)}{2}$
- o Use CNMA to find a solution to $F(x) = y \wedge P'(x, y)$
- 3. where $P'(x, y) = P(x, y) \wedge \phi(x, y) \leq m$
 - If solution $(x_0, F(x_0))$ is found, let $U = \phi(x_0, F(x_0))$
 - Else, let $L = m$

1.4 Evaluating CNMA

The principal evaluation of CNMA is by using it to explore designs of a watercraft propelled solely by wave energy. As shown in Figure 2 the rise and fall of the boat floating on the wave pulls and pushes at a hydrofoil below, causing a rotation about an axis. This rotation generates forward force during both the upward and downward wave motion, much like when a swimmer flaps flippers in a pool, his body is propelled forward. The watercraft is inspired by the Wave Glider [2,8].

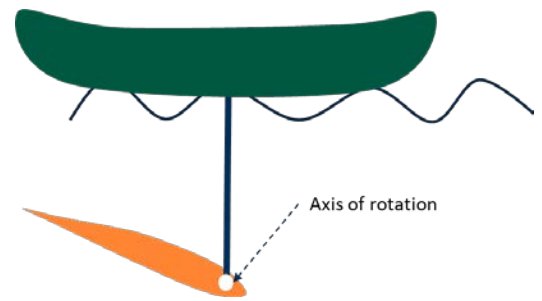


Figure 2. Boat Propelled Solely With Wave Energy

Design-space exploration for this watercraft requires solution to all of the above problems. A synthesis problem is computing the dimensions of the boat and hydrofoil so it can sail at a given steady-state velocity for a given wave condition. There are two separate simulators whose inputs and outputs need to be composed. One simulator is for the boat and another for the hydrofoil with recursive relationships between their inputs and output. There are ten real-valued inputs and outputs, making the system relatively high-dimensional. Simulators can be expensive to run. The hydrofoil computational fluid dynamics simulator often does not terminate. Performance-cost tradeoffs need to be made. The functions that the simulators implement are highly nonlinear. As part of performance-cost tradeoff, global optima need to be computed. We show how CNMA efficiently solves all of these problems. Other validations of CNMA are as follows:

- We formulate the same design problem with two mature nonlinear, derivative-free optimization methods: Gaussian Processes and COBYLA, and show that CNMA's performance in terms of three metrics, time to compute design, number of function calls and resilience to design tool failure, is far superior. Derivative-based methods were not used as they greatly increase the number of function evaluations.
- We evaluate CNMA against traditional nonlinear optimization benchmarks, including those with constraints on transcendental functions. When functions to be optimized are amenable to mathematical analysis, current methods significantly outperform CNMA. However, because they are also sensitive to starting points, they can sometimes fail to produce answers, whereas CNMA will produce an answer if it exists.
- We show that despite using a MILP solver, CNMA allows P to be nonlinear. This is accomplished by adding, for each nonlinear expression in P , an extra output to the neural network for F and then expressing P as a linear constraint on these outputs.
- We show how CNMA can be used to synthesize logical CPS such as parallel parking and obstacle avoiding robot maneuvers. Such problems are often solved with temporal-logic based synthesis engines. These examples show we might reduce them to non-temporal constraint solving by abstracting away time.
- We show how CNMA can be used to solve the synthesis problem for a CPS with both discrete and continuous variables. This problem is that of identifying a failure pattern in a smart grid that is consistent with sensor readings at a limited number of points in the smart-grid.

2 Methods Assumptions and Procedures

2.1 CNMA With Single Function

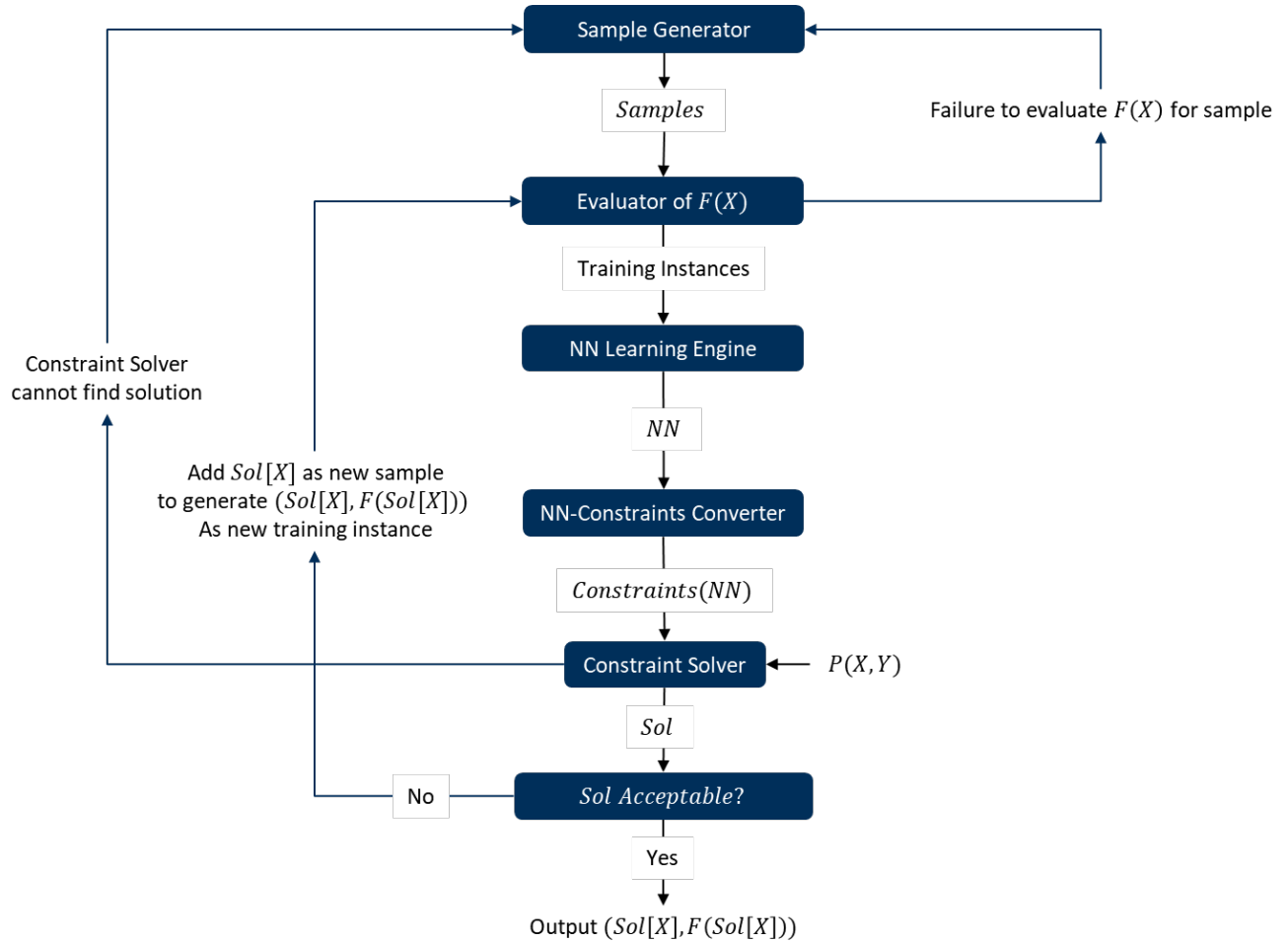


Figure 3. CNMA With Single Function

Figure 3 shows the architecture of CNMA. Let a CPS be modeled by a function $F(X) = Y$ where X and Y are vectors of discrete or continuous variables. Let $P(X, Y)$ be a predicate or condition. Our goal is to find values of X and Y such that $F(X) = Y \wedge P(X, Y)$. We assume that a method of evaluating F is available, e.g., as a simulator. Let Z be the tuple of variables $(z_0, \dots, z_k)$. Let $Z = (v_0, \dots, v_k)$ mean that for each i , the constant v_i is a value of z_i .

1. Let *Samples* be a set of values of X chosen via a method such as Combinatorial Interaction Testing, Random or Sobol Sequences.
2. Create the set T of training instances $(V, F(V))$ for each design point $V \in \text{Samples}$.
3. Train a neural network NN with T with the ReLU activation function. Optionally, compress it with a method such as NeST or SCANN.
4. Transform NN into an equivalent Mixed Integer Linear Program $\text{Constraints}(NN)$ on X, Y and variables in the hidden layers of NN .

5. Use an MILP constraint solver such as CPLEX or GUROBI to solve $Constraints(NN) \wedge P(X, Y)$.
6. If no solution is found, sample additional values of X , add them to $Samples$, and go to step 2.
7. If a solution is found, let it be $Sol = \{X = U_0, Y = V_0\}$
8. Check whether $P(U_0, F(U_0))$. If $F(U_0)$ is not evaluable, sample additional values of X , add them to $Samples$, and go to Step 2.
9. If $P(U_0, F(U_0))$, halt and output the solution $\{X = U_0, Y = F(U_0)\}$
10. Otherwise, let $T = T \cup \{(U_0, F(U_0))\}$ and go to step 3.

2.2 CNMA With Multiple Functions

Sometimes a CPS cannot be modeled as a single function, but instead as a collection of functions in which inputs of some functions are outputs of others. These functions could represent CPS components or subsystems. A single evaluator for the whole CPS may not even be available.

We now generalize our method to the multiple function case. The plan from the single function case remains the same: learn the function as a neural network, convert it into an equivalent constraint, solve it, and if the solution is unacceptable, use the solution to derive a correct mapping between inputs and outputs, add it to the training set and retry.

The novelty for the multiple function case is the idea that neural networks can be composed by Boolean conjunction of their equivalent constraint representations. Thus, we can learn each of the functions separately as neural networks, convert them into constraints, and in the above plan, solve their conjunction. If the solution is unacceptable, we can use the solution to generate new training instances for each of the functions and retry. Figure 4 shows this plan.

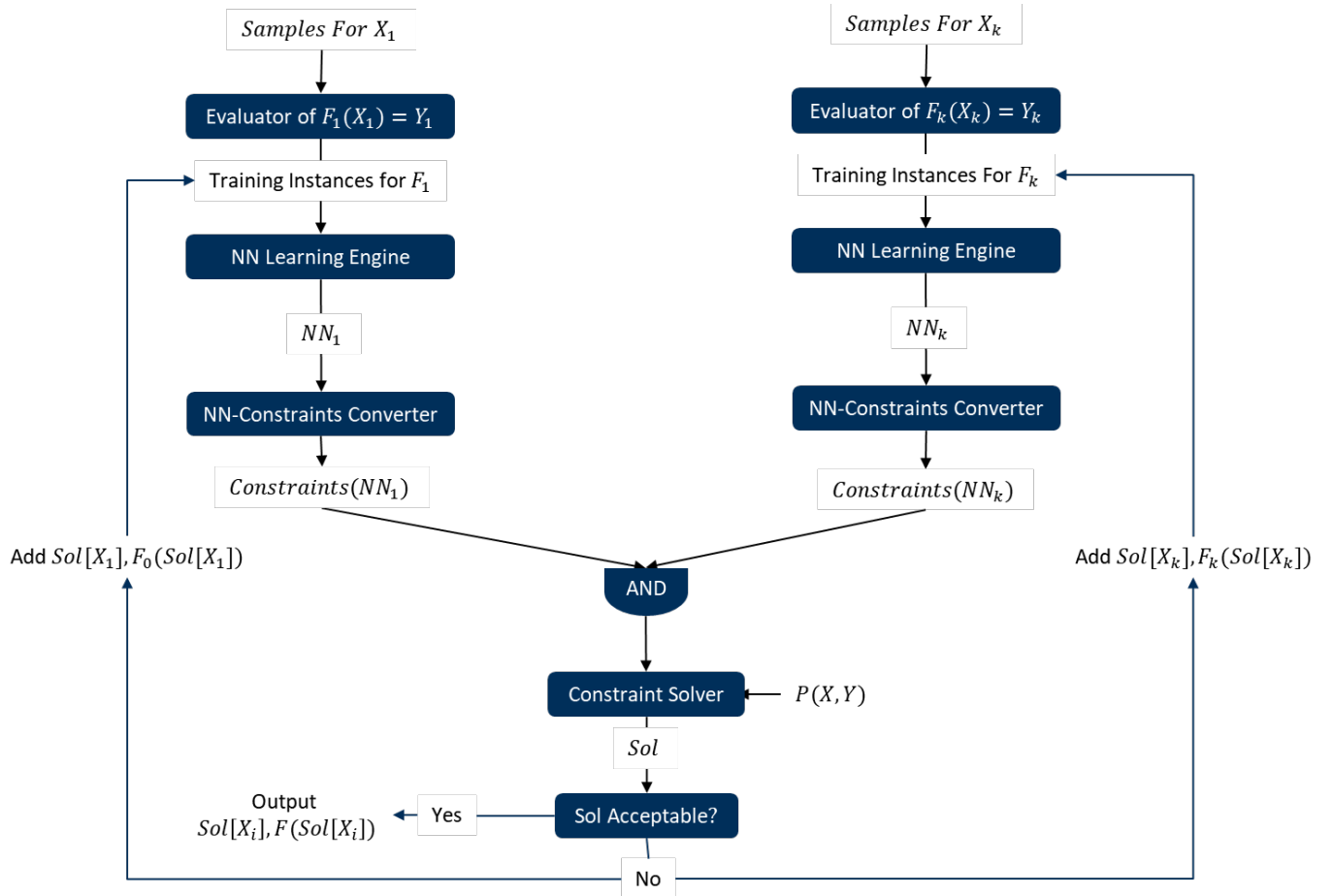


Figure 4. CNMA With Multiple Functions

We now precisely define this plan. Let the CPS be modeled by functions $F_i(X_i) = Y_i$ for $i \in \{1..k\}$ where X_i and Y_i are vectors of discrete or continuous variables. Let $X =$

$concat(X_1, \dots, X_k)$ and $Y = concat(Y_1, \dots, Y_k)$. Let $P(X, Y)$ be a predicate or condition and $Obj(X, Y)$ an objective function. Our goal is to find values of X and Y such that $P(X, Y) \wedge \forall i_{i=1}^k. F(X_i) = Y_i$. We assume methods of evaluating $F_1, \dots, F_k$ e.g., as simulators, are available. We also assume that none of the X_i, Y_i have any variables in common. The constraint $x = y$ can be used to model the fact that x and y are the same variable.

1. For each i , let $Samples[i]$ be an initial set of values of X_i
2. For each i , create the set $T[i]$ of training instances $(V, F_i(V))$ for each value $V \in Samples[i]$.
3. For each i , train a neural network $NN[i]$ with $T[i]$
4. For each i , transform $NN[i]$ into an equivalent MILP $Constraints(NN[i])$ on X_i, Y_i and variables in the hidden layers of $NN[i]$
5. Use a constraint solver to solve $Obj(X, Y)$ subject to $P(X, Y) \wedge \forall i. Constraints(NN[i])$. If no solution is found, for each i , sample additional values X_i , add them to $Samples[i]$, and go to step 2.
6. If a solution is found, let it be Sol where $Sol[X]$ is the value of all variables in X
7. Check whether $P(Sol[X], (F_1(Sol[X_1]), \dots, F_k(Sol[X_k])))$ holds.
8. If so, halt and output $\{X_i = Sol[X_i], Y_i = F_i(Sol[X_i])\}$
9. Otherwise, for each i , let $T[i] = T[i] \cup \{(Sol[X_i], F_i(Sol[X_i]))\}$ and go to step 3.

2.3 Designing a Watercraft Propelled Solely With Wave Energy

As shown in Figure 5, the rise and fall of the boat with the wave pulls and pushes at one end of the hydrofoil causing it to rotate about an axis. The rotation generates forward force during both the upward and downward wave motion.

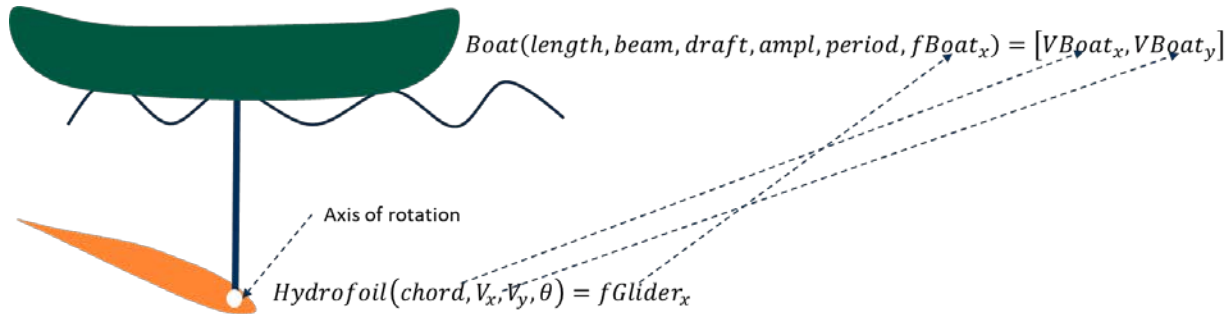


Figure 5. Structure of Wave Energy Propelled Watercraft With Recursive Relationships

The watercraft is modeled with two functions *Hydrofoil* and *Boat* whose domain and range are listed in the table below. Both of these functions are computed by a simulator, the first using a computational fluid dynamics package and the second using a Python script implementing a solution to a differential equation.

Function F	Input X	Output Y
<i>Hydrofoil</i>	$chord \in [0.2, 1.5]$ $V_{Glider_x} \in [0.1, 4]$ $V_{Glider_y} \in [0.1, 4]$ $\theta \in [30, 40]$	$fGlider_x$ This is the forward force output by the hydrofoil of unit span, $chord$ as chord length, moving at a velocity V_x, V_y at an angle of attack θ
<i>Boat</i>	$length \in [0.2, 2]$ $beam \in [0.2, 2]$ $draft \in [0.2, 2]$ $ampl \in [1, 5]$ $period \in [1, 9]$ $fBoat_x \in [10, 250]$	V_{Boat_x} V_{Boat_y} This is the average velocity of the boat of $length, beam, draft$ riding on a wave of $ampl, period$ and being pulled in the forward direction with force $fBoat_x$

The design problem is to find values of the ten variables above such that:

$$\begin{aligned}
 \text{Hydrofoil}(ch, V_x, V_y, \theta) &= fGlider_x \wedge \\
 \text{Boat}(\text{length}, \text{beam}, \text{draft}, \text{ampl}, \text{period}, fBoat_x) &= [VBoat_x, VBoat_y] \wedge \\
 VBoat_x &= VGlider_x \wedge \\
 VBoat_y &= VGlider_y \wedge \\
 fBoat_x &= \text{span} * fGlider_x \wedge \\
 \text{span} &= 0.3 \wedge \\
 \text{ampl} &= 4 \wedge \\
 \text{period} &= 7 \wedge \\
 VBoat_x &\geq VBoat_y
 \end{aligned}$$

The first two constraints state the input/output relationships for the *Hydrofoil* and *Boat* functions. The next three constraints state the equilibrium relationships – *note that these are recursive: force is an output of Hydrofoil but an input to Boat whereas velocities are outputs of the latter and inputs to the former*. The next constraint states the span of the hydrofoil. It is multiplied by $fGlider_x$ (force per unit span) to obtain the force experienced by the boat. The next two constraints state the amplitude and period of the current wave. The last constraint is upon the magnitudes of the speeds in the horizontal and vertical directions.

2.3.1.1 Definition of *Hydrofoil* Function

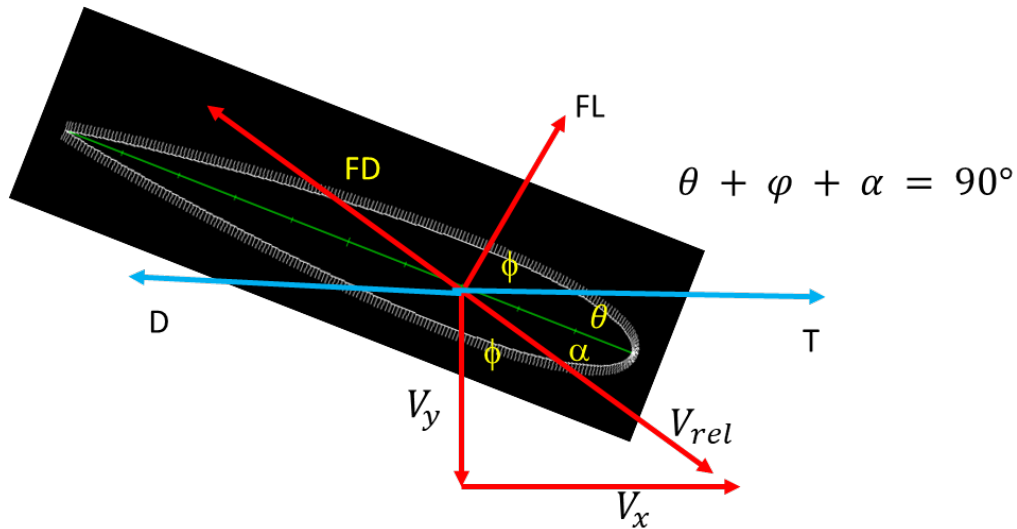


Figure 6. Hydrofoil Forces

As shown in Figure 6, the foil rotates about a point near its rounded end at right and has reached θ , its maximum angle with the horizontal. It is sinking due to downward wave motion. Assume it is moving at velocity V_{rel} with $\phi = \text{atan}\left(\frac{V_x}{V_y}\right)$. Let:

α = angle of attack along V_{rel}

FD = drag force along V_{rel}

FL = lift force perpendicular to V_{rel}

$chord$ = foil length

w = foil width (also called span)

ρ = density of water

$C_L(V_{rel}, chord, \alpha)$ = coefficient of lift of foil with above parameters

$C_D(V_{rel}, chord, \alpha)$ = the coefficient of drag of foil with above parameters

$$K = 0.5 * \rho * chord * w * V_{rel}^2:$$

Then:

$$FL = K * C_L(V_{rel}, chord, \alpha) * w$$

$$FD = K * C_D(V_{rel}, chord, \alpha) * w$$

$$T = FL * \cos(\phi)$$

$$D = FD * \cos(\theta + \phi) = FD * \sin(\phi)$$

Thus, the net forward force, the output of *Hydrofoil* is:

$$T - D = FL * \cos(\phi) - FD * \sin(\phi)$$

Coefficients of lift and drag are computed by a computational fluid dynamics airfoil/hydrofoil simulator called xfoil, from MIT. In Figure 7, a screenshot of xfoil in action, at the bottom is shown the shape of the foil being simulated and the table shows the lift and drag coefficients, among other quantities, associated with a given α , fluid density and viscosity [13].

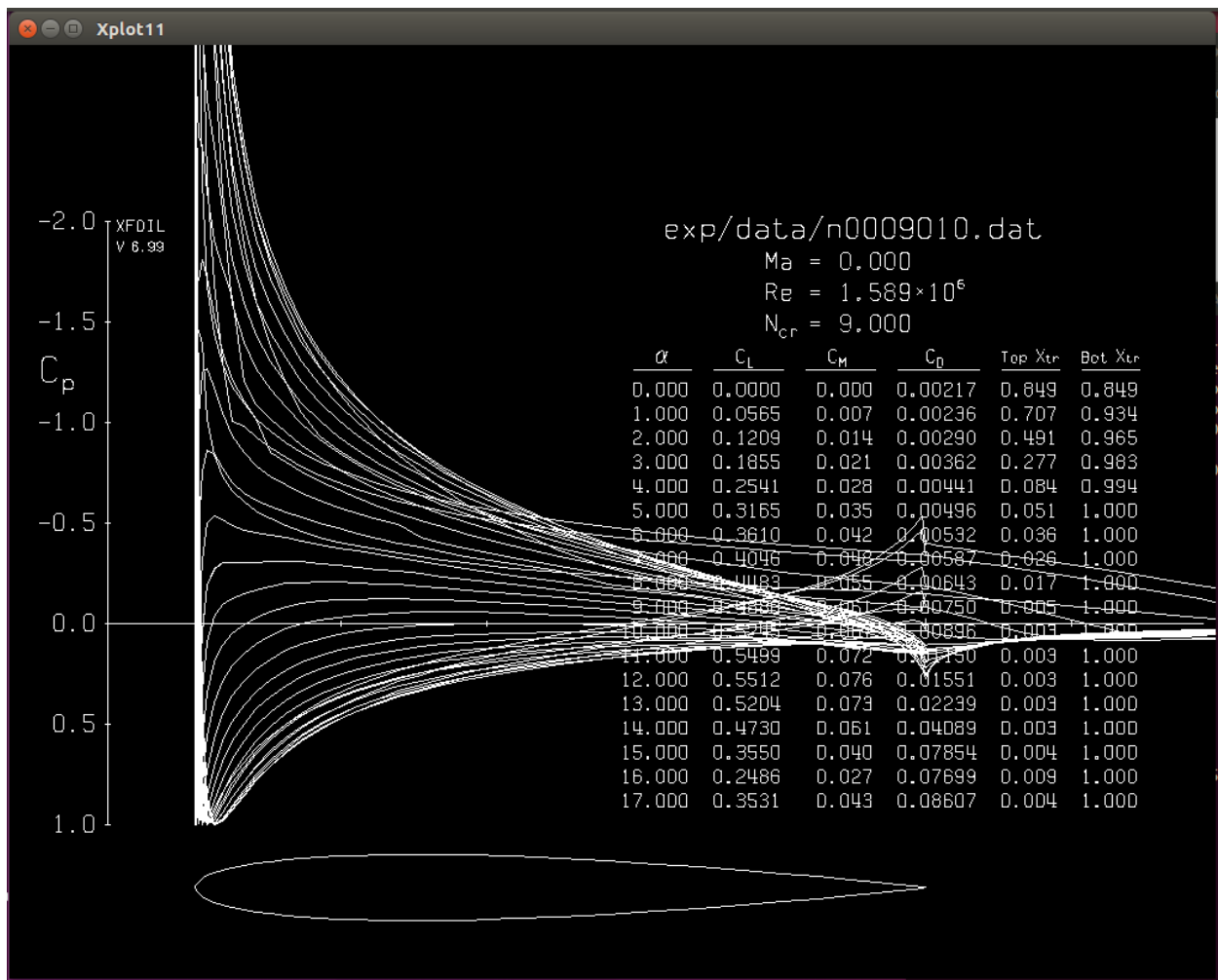


Figure 7. xfoil Screenshot

2.3.2 Definition of *Boat* Function

This is implemented as a Python program below:

```
import matplotlib.pyplot as plt
import numpy as np
import math
import os
from scipy.optimize import fsolve

def terminalV(x, *data):
    cf,S,lf,g,fx = data
    fn = x[0]*1.68781 / math.sqrt(lf*g)
    xfn = fn*fn*fn*fn/cf
    # limit out the CT - at 1.5 times CF
    # Quadratic form upto 0.3 and
    if(xfn < 0.3):
        y = -1.2579*xfn*xfn + 0.7615*xfn + 1.2559
    else:
        y = 1.37
    ct = y*cf
    res = fx - ct*x[0]*x[0]*S
    # print x[0],res,y,ct,xfn,fn
    # print " --"
    return res

def checkBoat(boatDesign):
    [L,B,D,amp,pd,fx] = boatDesign
    # L: Length in M; B: Beam in m; D depth in m
    # amp: Wave amplitude in m (0.88)
    # period: Wave period in seconds (4.6)
    # fx force in Newtons
    lf = L*3.28084
    df = D*3.28084
    bf = B*3.28084
    # Light displacement (0.5 multiplier) wetted area formula
    S = 0.5 *lf*(df + bf)
    cf = 0.0231
    g = 32.2 # in ft/s^2
    #note vx is in knots
    fxpf = fx*0.224809/g # in lbf in lb.
    data = (cf,S,lf,g,fxpf)
    vxkts = fsolve(terminalV, 1.0, maxfev=10000,args=data)
    vx = vxkts[0]*0.514444 # convert to m/s
    vy = 4*amp/float(pd)
    return [L,B,D,amp,pd,fx,vx,vy]
```

3 Results and Discussions

3.1 Synthesis of Hydrofoil With Single Function CNMA

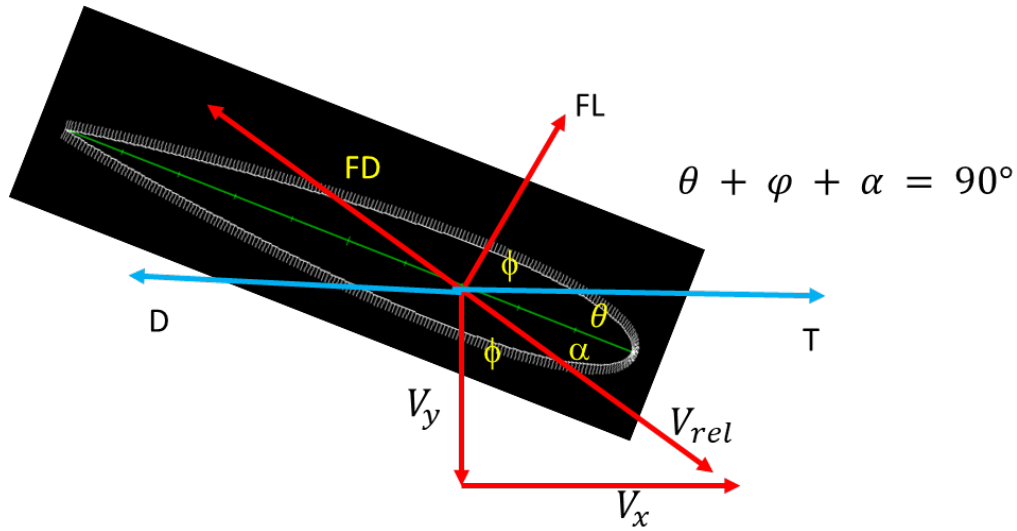


Figure 8. Hydrofoil Forces Repeated

CNMA is now illustrated by using it to design a hydrofoil the “engine” of the water craft that we ultimately intend to design. See Figure 8. The span of the foil, its length viewed from the top, is assumed to be 1. In the function $Hydrofoil(chord, V_x, V_y, \theta)$:

- $chord$ is the length of the hydrofoil
- (V_x, V_y) is the velocity with which the hydrofoil is moving through a fluid
- θ is the angle of attack of the hydrofoil

The function computes T , the forward force that pulls the boat to which it is attached. Letting $chord = 0.5, V_x = V_y = 1$ the synthesis or inverse design problem is to find $\theta \in [20, 40]$ for which $T*500$ is within $[0.25, 4]$. Renaming T to be f_x , we request CNMA to find $(chord, V_x, V_y, \theta, f_x)$ such that:

$$Hydrofoil(chord, V_x, V_y, \theta) = f_x \wedge P(chord, V_x, V_y, \theta, f_x)$$

where

$$P(chord, V_x, V_y, \theta, f_x) =$$

$$(chord = 0.5 \wedge V_x = 1 \wedge V_y = 1 \wedge \theta \in [20, 40] \wedge f_x / 500 \in [0.25, 0.4])$$

Table 1 provides an initial training set of just 22 points in the domain of $Hydrofoil$:

Table 1. Hydrofoil Training Set

chord	vx	vy	theta	fx	fx/500
0.3375	2.5375	3.5125	39.375	873.557	1.747114
0.45625	1.31875	1.31875	30.9375	234.5194	0.469039
0.753125	2.903125	2.415625	29.53125	2800.58	5.601161
1.821875	1.684375	1.196875	26.71875	3979.438	7.958876
1.139063	3.085938	0.526562	3.515625	688.106	1.376212
0.901563	0.648437	1.014062	31.64063	373.2243	0.746449
0.426563	3.573438	1.989062	20.39063	712.4093	1.424819
0.248438	1.014062	1.135937	44.29688	27.42926	0.054859
1.910938	3.451562	1.623437	16.17188	10965.94	21.93189
1.435938	0.526562	0.648437	27.42188	334.5429	0.669086
0.842188	2.720313	1.379687	24.60938	317.1279	0.634256
0.129688	3.207812	3.817187	30.23438	103.042	0.206084
0.144531	1.653906	1.836719	36.21094	46.29936	0.092599
0.619531	2.628906	0.861719	2.460938	284.694	0.569388
0.382031	3.116406	2.324219	8.085938	198.8206	0.397641
0.500781	0.922656	0.617969	22.14844	87.50982	0.17502
0.263281	2.385156	3.055469	27.77344	312.3836	0.624767
1.747656	1.532031	1.714844	40.42969	5191.751	10.3835
1.391406	3.238281	1.958594	26.36719	3469.777	6.939554
0.649219	3.542969	2.750781	27.07031	2917.145	5.83429
0.886719	1.105469	1.288281	21.44531	623.0266	1.246053
0.5	1	1	24.2531	103.8033	0.207607
0.5	1	1	29.08688	89.47022	0.17894
0.511539	1.746255	0.213259	6.728581	0.867927	0.001736
0.5	1	1	40	110.146	0.220292
0.5	1	1	35.21	180.025	0.36

The following figures show how CNMA finds a solution in just 4 iterations. The solid black curve is the plot of $\frac{f_x}{500}$ with θ computed from *Hydrofoil*. The yellow rectangle is the region where P is true. A solution is any point on the black curve within the yellow rectangle. The blue curve is CNMA's model of *Hydrofoil* based on the current training set.

In Figure 9, CNMA does find a satisfying point on the blue curve for which $\theta = 24.253$. However, the true value of $\frac{f_x}{500}$ for this value is 0.207 but (24.253, 0.207) is not satisfying. CNMA will now add the new training instance:

0.5	1	1	24.2531	103.8033	0.207607
-----	---	---	---------	----------	----------

Note that because of the definition of P , any solution produced by CNMA will have ($chord = 0.5, V_x = V_y = 1$). Similarly, in the next iteration, shown in Figure 10, CNMA adds the new training instance:

0.5	1	1	29.08688	89.47022	0.17894
-----	---	---	----------	----------	---------

In Figure 11, CNMA attempts to compute the true value for the design it computed but this time xfoil fails. Thus, CNMA generates a random design, computes the associated force, and adds the following training instance:

0.511539	1.746255	0.213259	6.728581	0.867927	0.001736
----------	----------	----------	----------	----------	----------

Note that since the design is random $chord, V_x, V_y$ are no longer assured to be 0.5, 1, 1 respectively. Finally, after adding one more training instance:

0.5	1	1	40	110.146	0.220292
-----	---	---	----	---------	----------

CNMA produces a solution:

$$chord = 0.5, V_x = 1, V_y = 1, \theta = 35.21, \frac{f_x}{500} = 0.36$$

See Figure 12 and Figure 13.

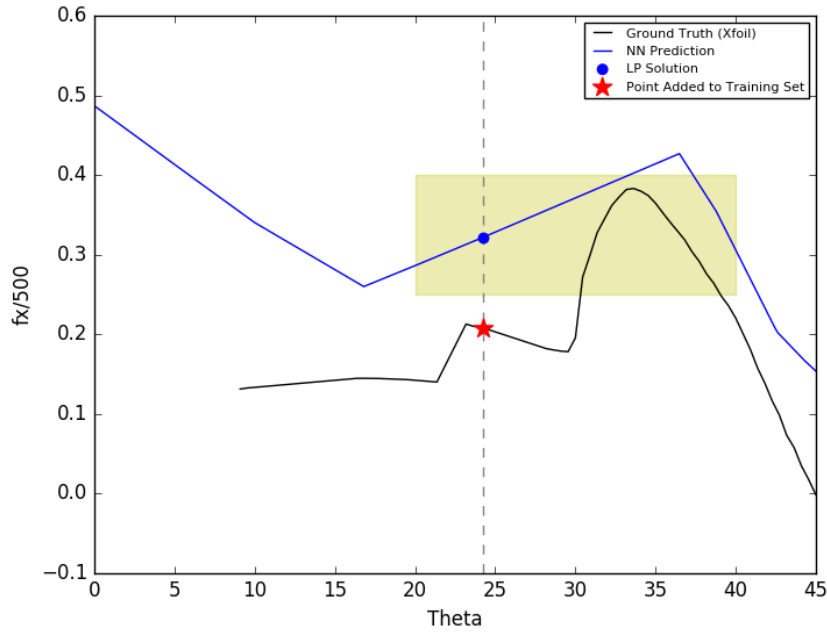


Figure 9. CNMA Iteration 1

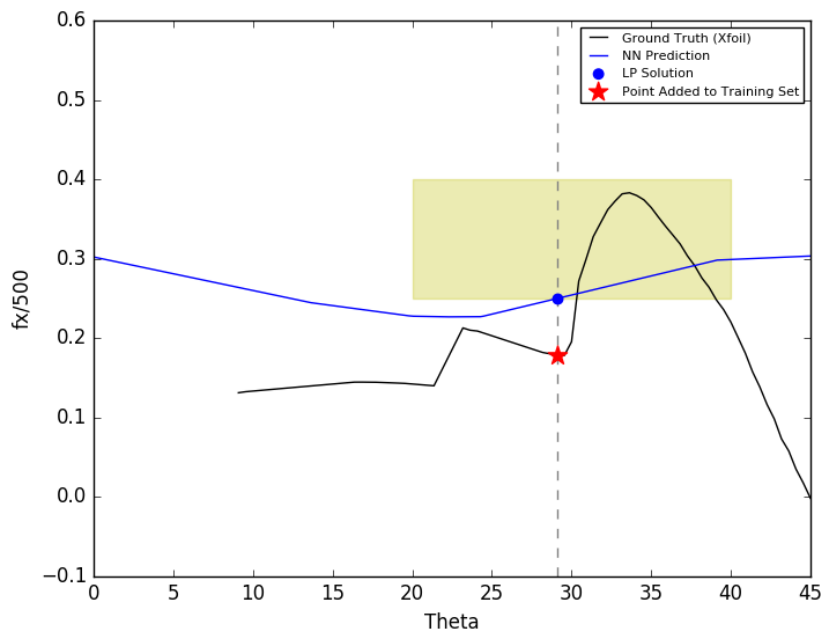


Figure 10. CNMA Iteration 2

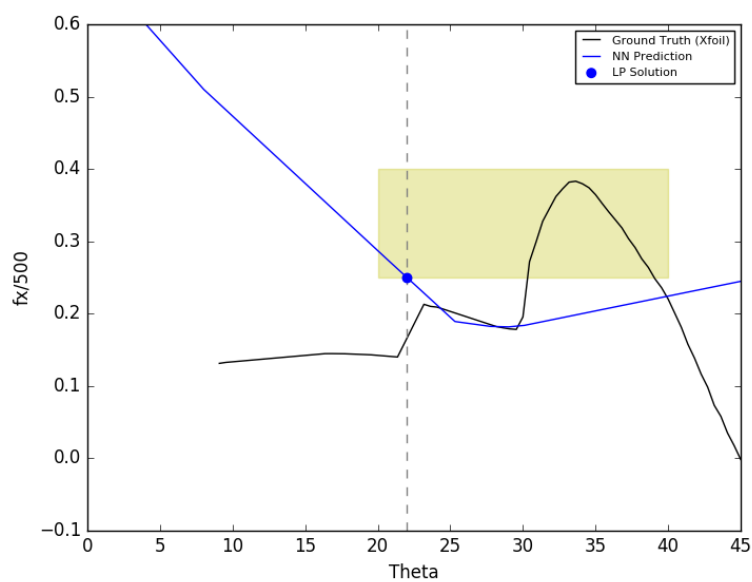


Figure 11. CNMA Iteration 3

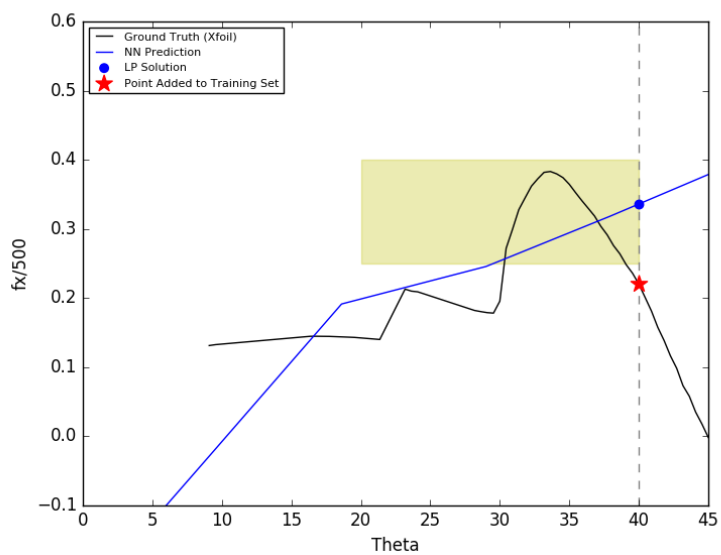
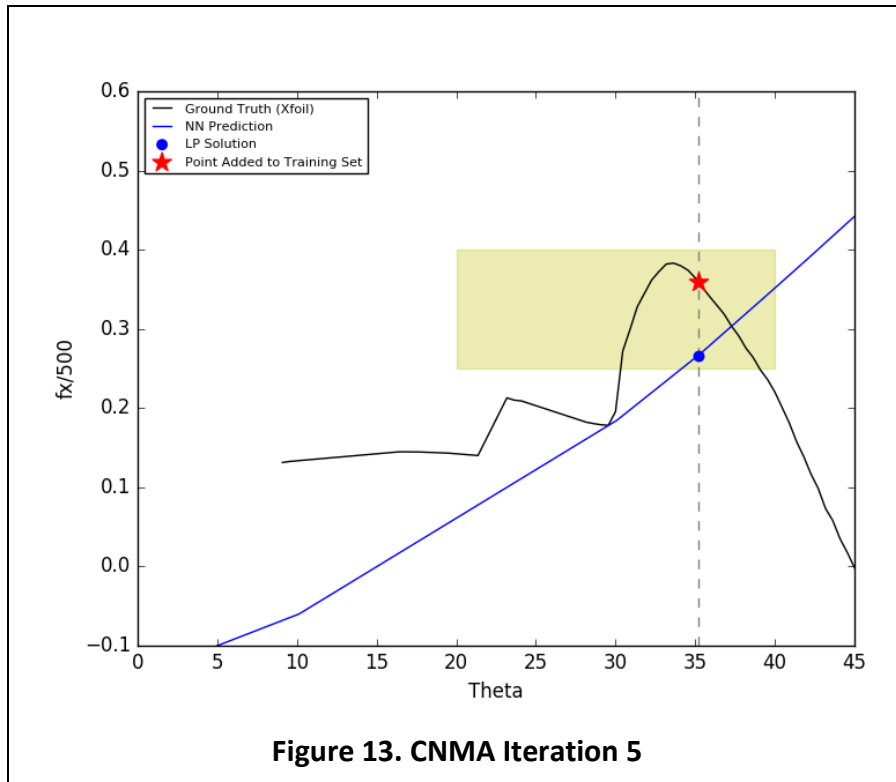


Figure 12. CNMA Iteration 4



3.2 Synthesis of Watercraft With Multiple Function CNMA

We now show how to solve the original design problem of finding values of $chord, V_x, V_y, \theta, length, beam, draft, ampl, and period$ such that:

$$\begin{aligned} Hydrofoil(chord, V_x, V_y, \theta) &= fGlider_x \wedge \\ Boat(length, beam, draft, ampl, period, fBoat_x) &= [VBoat_x, VBoat_y] \wedge \\ VBoat_x &= VGlider_x \wedge \\ VBoat_y &= VGlider_y \wedge \\ fBoat_x &= span * fGlider_x \wedge \\ span &= 0.3 \wedge \\ ampl &= 4 \wedge \\ period &= 7 \wedge \\ VBoat_x &\geq VBoat_y \end{aligned}$$

With a neural network of architecture (6, 20, 2) for the Boat and (4, 20, 1) for the Hydrofoil, CNMA produced the solution (with equality to within 5%):

$$\begin{aligned} chord &= 0.2 \\ VGlider_x &= 2.60984427443 \\ VBoat_x &= 2.65346401167 \\ VGlider_y &= 2.28290476301 \\ VBoat_y &= 2.28571428571 \\ \theta &= 30.0 \\ length &= 0.2 \\ beam &= 0.2 \\ draft &= 0.2 \\ ampl &= 4.0 \\ period &= 7.0 \\ fGlider_x &= 170.081368542 \\ fBoat_x &= 51.9224300031 \\ span &= 0.3 \end{aligned}$$

3.3 Evaluating CNMA Performance

To understand the effect of the initial training set and the neural network architecture on the performance of CNMA, we ran CNMA multiple times, each with a different number of initial samples: 10, 20, 50, 100, 200, and for six different neural network architectures. Figure 14 summarizes our findings. Interestingly enough, a smallish, single hidden-layer neural network performed the best along two metrics: the number of calls to the boat and hydrofoil simulators and the overall query time.

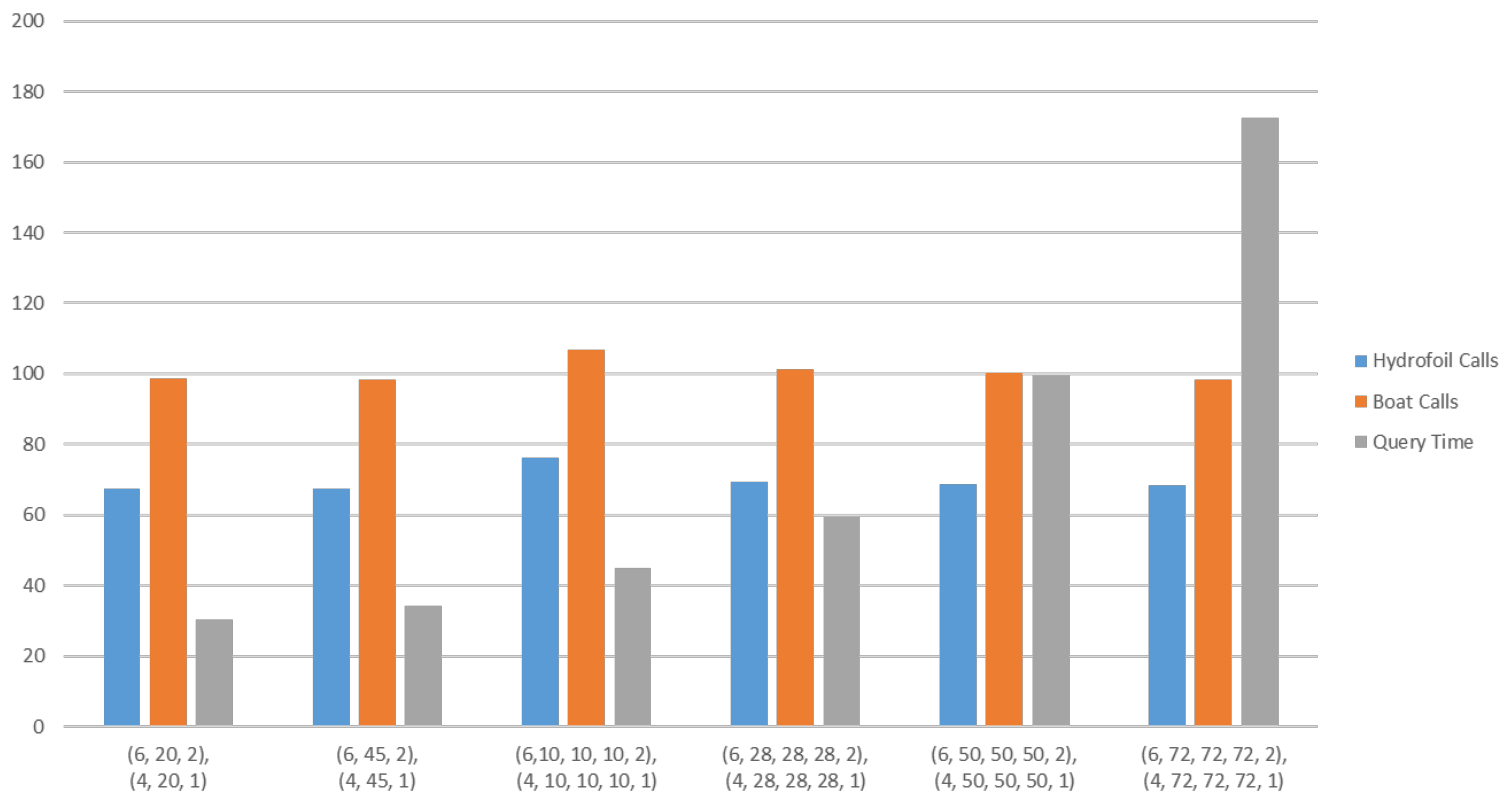


Figure 14. CNMA Performance

3.4 Non-Parametric Synthesis And Optimization

So far, the structure of the watercraft was known: it had a boat and a hydrofoil and the synthesis problem was to find values of input and output variables satisfying requirements. Sometimes, it may be necessary to explore alternative structures if the current one does not satisfy the requirements. For example, an alternative to a hydrofoil-engine is a propeller-engine that can achieve arbitrary speed.

Suppose the cost of a hydrofoil is the product of its chord and span and the cost of the propeller is a constant at 2. Suppose also that our goal is to minimize cost subject to satisfying the velocity requirement.

If the requirement is that the velocity be greater than 2.5m/s, CNMA has a hard time finding a chord and span whose product is less than 2. In that case, the propeller-engine would be chosen. If the requirement is that the velocity be greater than 2m/s, then a hydrofoil-engine is chosen because the chord and span can be more easily found whose product is less than 2.

We now show how alternative structures can be modeled in CNMA and the correct structure chosen depending on the requirement to be satisfied. This is accomplished by introducing mutually-exclusive decision variables, one per structure, and adding constraints that if a decision-variable is true, so is the associated structure. Since MILP solvers only solve conjunction of constraints, implications need to be transformed into conjunctions. Such transformation is accomplished by the Big M method. For example, where Z is a decision variable, the constraint $Z \supset X \leq Y$ is modeled with:

$$\begin{aligned} Z &\in \{0,1\} \\ X - M * (1 - Z) &\leq Y \end{aligned}$$

where M is a large positive number. If $Z = 1$, the constraint becomes $X \leq Y$ as desired. If $Z = 0$ the constraint is vacuously true. To model $Z \supset X = Y$ use $(Z \supset X \leq Y \wedge Z \supset Y \leq X)$.

Let *Hydrofoil* and *Propeller* be two decision variables, corresponding respectively to whether the hydrofoil-engine or propeller-engine is attached to the boat. Mutual exclusivity is expressed as:

$$\mathbf{Hydrofoil + Propeller = 1} \quad (1)$$

To model the choice of the hydrofoil and associated cost, use the constraint below along with one on a new variable, *Cost*:

$$\begin{aligned} &\text{Hydrofoil} \supset \\ &\left(VBoat_x = VGlider_x \wedge Boat_y = VGlider_y \wedge fBoat_x = span * fGlider_x \wedge \right. \\ &\quad \left. span = 1.0 \wedge Cost = chord * span \right) \end{aligned} \quad (2)$$

To model the choice of the propeller and associated cost, use:

$$\text{Propeller} \supset (Cost = 2) \quad (3)$$

Now, CNMA can solve the following query:

Minimize *Cost*, subject to Equations (1), (2), (3) and

$$V_{Boat_x} \geq 2$$

$$Hydrofoil, Propeller \in \{0,1\}$$

$$Hydrofoil + Propeller = 1$$

CNMA chooses the hydrofoil-engine as its cost is less than 2, the cost of the propeller-engine. If instead, $V_{Boat_x} \geq 2.5$ were required, CNMA would choose the propeller-engine as it can achieve high speed with a fixed cost of 2. The hydrofoil-engine would have a higher cost to support that speed.

3.5 Designing Water Craft With Traditional Optimization Methods

To evaluate traditional optimization methods for designing the water craft, we need a single function model of the water craft. In this model, there is a single input vector of the inputs of all subsystems. All output variables are modeled as function calls on parts of this vector. Relationships, even recursive ones, are specified using penalty functions representing the cost of violating these. These functions are added to form an objective function, and unconstrained optimization is used to minimize this function. All constraints would be satisfied when the objective function is 0. We did not use other methods of modeling constraints such as Sequential Unconstrained Minimization Technique (SUMT) and Barrier as these are more complex than penalty functions. The penalty function for $\geq$ is encoded in Python as:

$$gePenalty(x, y) = 10 * (\max(0, y - x))^2$$

The penalty function for $=$ is encoded as:

$$equalsPenalty(x, y, tol) = gePenalty(tol, RelativeError(x, y))$$

stating that x, y are considered equal if they are with a tolerance tol of each other, where:

$$RelativeError(x, y) = \frac{|\frac{x-y}{x}| + |\frac{x-y}{y}|}{2}$$

The input vector has 10 variables, 4 for *Hydrofoil* and 6 for *Boat*:

$$[chord, V_xGlider, V_yGlider, \theta, length, beam, draft, amp, period, f_xBoat]$$

From this vector, the three output variables are computed as follows, with $span = 0.3$:

$$[V_xBoat, V_yBoat] = Boat(length, beam, draft, amp, period, f_xBoat)$$

$$f_xGlider = 0.3 * Hydrofoil(chord, V_x, V_y, \theta)$$

Now, we minimize the following objective function encoding the penalties of violating the constraints of the water craft in Section 2.3:

$$\begin{aligned} & equalsPenalty(f_xGlider, f_xBoat, 0.05) + \\ & equalsPenalty(V_xGlider, V_xBoat, 0.05) + \\ & equalsPenalty(V_yGlider, V_yBoat, 0.05) + \\ & equalsPenalty(amp, 4, 0.05) + \\ & equalsPenalty(period, 7, 0.05) + \\ & gePenalty(V_xBoat, V_yBoat) \end{aligned}$$

A major problem with this approach is that *Hydrofoil* does not terminate for many inputs. Then, an artificial value has to be returned but that significantly disrupts the search for the optima.

A second major problem with this approach is that if the optimization method needs a starting point (an initial value of the input vector) it can be very sensitive to its choice. Choosing a good starting point can be hard in high-dimensional spaces.

3.5.1 Design with Gaussian Processes

We used `gp_minimize` package from Python's `skopt` package and directly implemented the above plan. Because of the above problems, it simply did not terminate even after 500 iterations.

3.5.2 Design With COBYLA

COBYLA is a derivative free optimization method. It uses simplexes of $n + 1$ points (n is the number of dimensions) to create linear approximations of objective and constraints. We used Python's `scipy.optimize` package. COBYLA has built in support for constraints, so we did not directly use the above penalty functions. However, COBYLA does need a starting point. COBYLA stops if any of the following are true:

- A solution to the Wave Glider problem is found
- The objective function is evaluated more than 300 times
- The relative change in the input parameters after an optimization step is less than $1e-10$

We ran COBYLA 20 times, each with randomly chosen starting points. Results were as follows:

- 4 valid solutions were found
- The Hydrofoil simulator was called 5,392 times
- To find a solution, COBYLA needs to call the simulator 1,348 times on average (compared to ~ 100 for CNMA). This is more than one order of magnitude larger.

3.6 Synthesizing Logical CPS

We show how to use CNMA to synthesize logical CPS e.g., control maneuvers. A general principle is to model a maneuver as a configurable plan. Then, ask: if the maneuver were performed with that configuration for an appropriate amount of time, would safety be preserved? Encode the answer as a program or simulator $F(X) = Y$ where X is the plan configuration and $Y = \text{true/false}$. Now use CNMA to find $X: F(X) = Y \wedge Y = \text{true}$. We illustrate this idea using two examples.

3.6.1 Parallel Parking Maneuver

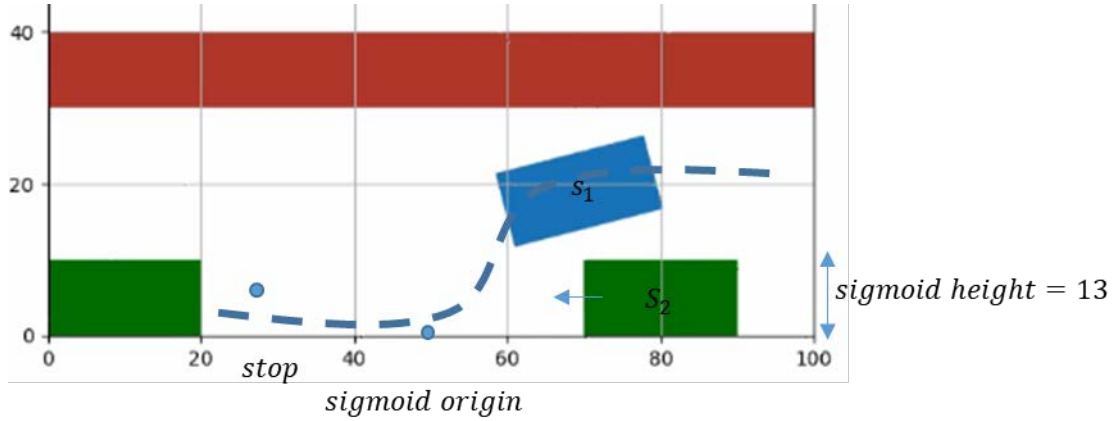


Figure 15. Finding Parameters of Sigmoid Curve For Safe Parallel Parking With Moving Obstacles

This problem is inspired by that of using Gompertz curves for parallel parking from Toyota [32]. See Figure 15. The problem is to find values of s_1 , s_2 , **damp**, **stop** such that the blue car:

- Starts at (100, 13)
- Moves at speed s_1 along a sigmoid $y(x) = \frac{13}{1+e^{-damp \cdot x}}$ with origin (50, 0)
- Stops at (**stop**, 5) where the derivative of the curve is close to 0, so car is straightened
- Never collides with green car at right starting at (100, 5) moving at speed s_2
- Never collides with road divider at top
- Never collides with car at left

We encode $\text{safe}(s_1, s_2, \text{damp}, \text{stop})$ as a program that returns *true/false* depending on whether the above conditions are satisfied, during a long enough period of time. We then use CNMA to find $s_1, s_2, \text{damp}, \text{stop}$ such that $\text{safe}(s_1, s_2, \text{damp}, \text{stop})$. Out of 221K possible training instances, CNMA only needed to generate 110 and solved the query in a few seconds.

3.6.2 Robot Path Planning

We are given a set of obstacles, a robot of a given shape, an initial position (x_0, y_0) and a final position (x_k, y_k) . The problem is to find a safe (collision-free) path $(x_0, y_0), \dots, (x_k, y_k)$. We can encode the predicate $safePath(x_0, y_0, \dots, x_k, y_k)$ as a function that would return *true/false* depending on whether the trajectory is safe for the given set of obstacles, and then compute the path by using CNMA to:

Find $x_0, y_0, \dots, x_k, y_k$ such that $safePath(x_0, y_0, \dots, x_k, y_k)$

However, the feature space is of size $2 * k$ and for a 100x100 grid the total number of combinations with a discrete step of size 10 would be 10^{2*k} .

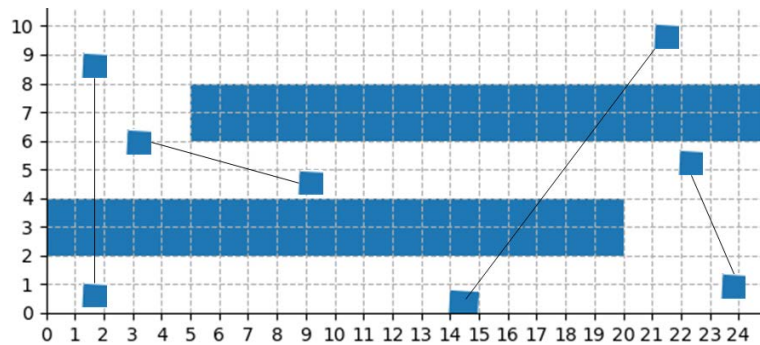


Figure 16. Learning $safe(x_0, y_0, x_1, y_1)$

We can drastically cut down this space by learning just $safe(x_0, y_0, x_1, y_1)$ to mean that the robot can go from x_0, y_0 to x_1, y_1 without colliding. See Figure 16. Then, we can find a path of length k by solving the following in CNMA:

$$safe(x_0, y_0, x_1, y_1) \wedge \\ safe(x_1, y_1, x_2, y_2) \wedge \dots \wedge \\ safe(x_{k-1}, y_{k-1}, x_k, y_k)$$

This would make use of the multiple function CNMA.

For example, if $(x_0, y_0) = (24, 9)$ and $(x_k, y_k) = (6, 1)$ for $k = 5$ CNMA computes the path as shown. The neural network architecture used was (4, 8, 8, 8, 1). Out of a total of 62.5K training instances for a 25x10 grid, CNMA only needed to generate 250 and solved the query in a few seconds. See Figure 17.

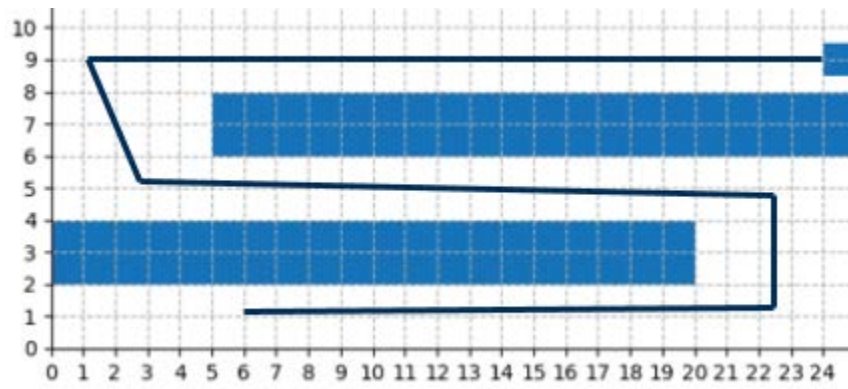


Figure 17. Safe Path Computed By CNMA

3.7 Synthesizing CPS With Discrete and Continuous Variables

We address the problem of failure detection in a power grid system without full observability, i.e., where sensor placement is scarce. The inverse problem that we are trying to solve is: Given currents and voltages from sensors at a limited number of the lines in a power grid, what is a possible status of all other lines, and currents and voltages at those lines?

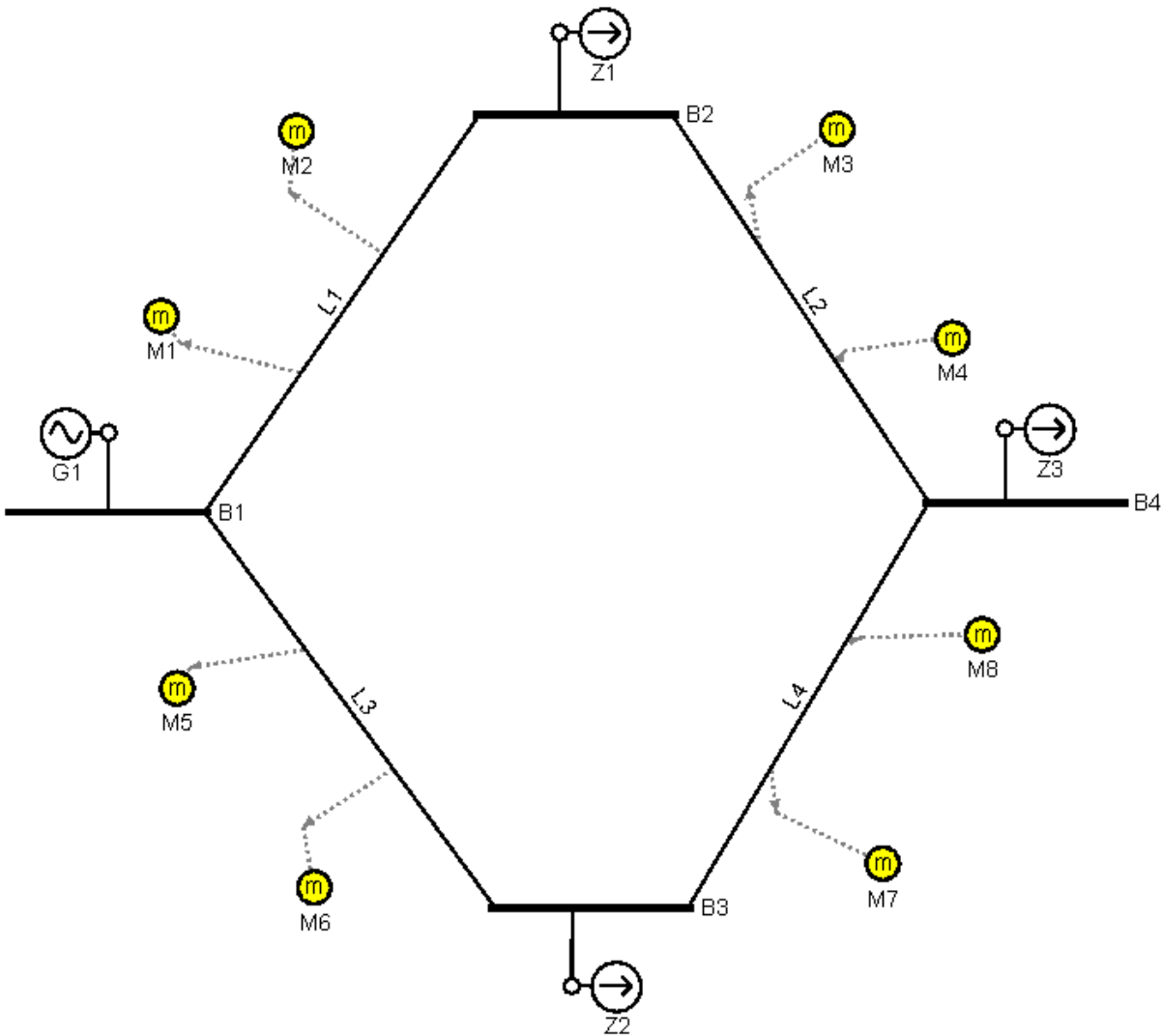


Figure 18: Test circuit diagram. G1 is a single generator, Z1-3 are loads, M1-8 are metering devices, and L1-4 are power lines (with status 0/1). M1, for example, represents $(i1, v1l)$, and M2 indicates $(i1, v1r)$.

We use the public domain Open Distributed System Simulator (OpenDSS) tool that simulates for a given power grid topology, voltage sources and loads, the voltage and current at all lines in the grid. We use this tool along with CNMA to solve the inverse problem.

Figure 18 represents the circuit used to test the functionality of the OpenDSS simulator and its interface with the CNMA solver. This circuit is comprised of four buses, four lines, one generator that supplies voltage to the system, and three loads. We represent this circuit with a neural network of one input layer, one hidden layer, and one output layer. The input layer is comprised of 8 neurons/inputs: 1 for the generator voltage, 3 for the load powers, and 4 for the Boolean line statuses. The hidden layer is comprised of one hundred neurons. The output layer neurons represent the thirteen outputs of the system: the current and left/right voltages observed on each line and the total number of line failures.

To train the neural network in the CNMA solver we generate fifty random instances, each with input values randomly generated within defined upper and lower bounds. Line statuses are represented with values between zero and one, which are converted to Boolean by the OpenDSS solver with a threshold of 0.5, while voltage generator and load powers are randomized within preset ranges. Table 2 lists sample training instances (10 of the 50 generated for CNMA), inputs and corresponding outputs in Table 3.

Table 2: Training Data Inputs

<i>V_{VS}</i> (kV)	<i>P_{R1}</i> (kW)	<i>P_{R2}</i> (kW)	<i>P_{R3}</i> (kW)	<i>L1</i>	<i>L2</i>	<i>L3</i>	<i>L4</i>
5.05	5.05	5.05	5.05	0.5	0.5	0.5	0.5
7.525	2.575	7.525	2.575	0.75	0.25	0.75	0.25
2.575	7.525	2.575	7.525	0.25	0.75	0.25	0.75
3.8125	3.8125	6.2875	1.3375	0.875	0.875	0.125	0.625
8.7625	8.7625	1.3375	6.2875	0.375	0.375	0.625	0.125
6.2875	1.3375	3.8125	3.8125	0.125	0.625	0.875	0.875
1.3375	6.2875	8.7625	8.7625	0.625	0.125	0.375	0.375
1.95625	3.19375	3.19375	6.90625	0.5625	0.1875	0.0625	0.9375
6.90625	8.14375	8.14375	1.95625	0.0625	0.6875	0.5625	0.4375
9.38125	0.71875	5.66875	9.38125	0.3125	0.4375	0.8125	0.6875

Table 3: Training Data Outputs

<i>i1</i>	<i>v1l</i>	<i>v1r</i>	<i>i2</i>	<i>v2l</i>	<i>v2r</i>	<i>i3</i>	<i>v3l</i>	<i>v3r</i>	<i>i4</i>	<i>v4l</i>	<i>v4r</i>	<i>N_F</i>
22.8	2915.6	2915.6	7.6	2915.6	2915.6	22.8	2915.6	2915.6	7.6	2915.6	2915.6	0
11.5	4344.5	4344.5	0	4344.5	0	33.7	4344.5	4344.5	0	4344.5	0	2
0	1486.7	0	0	0	0	0	1486.7	0	0	0	0	2
25.9	2201.1	2201.1	17.3	2201.1	2201.1	0	2201.1	2201.1	14.3	2201.1	2201.1	1
0	5059	0	0	0	0	7	5059	5059	0	5059	0	3
0	3630.1	3630	5	3630	3630	33.5	3630.1	3630	19.3	3630	3630	1
5	772.2	772.2	0	772.2	0	0	772.2	0	0	0	0	3
3.7	1129.4	1129.4	0	1129.4	0	0	1129.4	0	0	0	0	2
0	3987.3	0	0	0	0	33.5	3987.3	3987.3	0	3987.3	0	2
0	5416.2	0	0	0	5416.1	84	5416.2	5416.1	52.4	5416.1	5416.1	2

For the query:

$$\begin{aligned}
 i2 &== 0, \\
 v2l &> 0, \\
 v2r &== 0, \\
 N_F &== 3, \\
 V_{VS} &== 1, \\
 P_{R1} &== 2, \\
 P_{R2} &== 1, \\
 P_{R3} &== 4
 \end{aligned}$$

With this query we use a single line measurement (Line 2), for which we constrain the current $i2$, in addition to the left and right voltages $v2l$ and $v2r$. We specify three total line failures (only one line status L1-4 will be 1), along with the power values of the loads (P_{R1} , P_{R2} , P_{R3}) and the voltage of the generator (V_{VS}). Given these constraints, CNMA yields a solution with the measurements and status of every line in the system.

CNMA produced the solution:

Voltage Source: $V_{VS} = 1.0 \text{ kV}$
Load 1: $P_{R1} = 2.0 \text{ kW}$
Load 2: $P_{R2} = 1.0 \text{ kW}$
Load 3: $P_{R3} = 4.0 \text{ kW}$
Line Status: $(L1, L2, L3, L4) = [1, 0, 0, 0]$

Line 1: $i1 = 1.312 \text{ A}$
 $v1l = 577.35 \text{ V}$
 $v1r = 577.348 \text{ V}$
Line 2: $i2 = 0.0 \text{ A}$
 $v2l = 577.348 \text{ V}$
 $v2r = 0.0 \text{ V}$
Line 3: $i3 = 0.0 \text{ A}$
 $v3l = 577.35 \text{ V}$
 $v3r = 0.0 \text{ V}$
Line 4: $i4 = 0.0 \text{ A}$
 $v4l = 0.0 \text{ V}$
 $v4r = 0.0 \text{ V}$
Number of Failures: $N_F = 3$

Number of Iterations: 2

Total Time = 8.17301 s

This query demonstrates the realistic potential for the detection of failures in a system. With known load values and only a single line current/voltage measurement, the solver is very quickly able to determine the three failed lines in the system (L2, L3, L4), and the current/voltages of the remaining unobservable lines ($i1, i3, i4, v1l, v3l, v4l, v1r, v3r, v4r$), all of which are consistent with the constraints and the circuit topology. Only two iterations are required, where the 50 training samples are retrained once with a single additional instance.

Even for a simple circuit the full input domain is large, and as simulated power systems grow in size, fully sampling the domain becomes infeasible. With four lines, there are 16 combinations of line status inputs, and current/voltage values for the generator and loads are in a continuous range of real numbers within a maximum and minimum bound. The current CNMA implementation uses 50 sample instances to train the Neural Network, and with active learning is able to determine a correct solution (within tolerance) with just a few retraining iterations given a set of constraints. As increasingly complex systems with many more inputs and outputs are present, CNMA seems to offer an efficient method for determining line failures while only sampling a small fraction of the full domain.

3.8 Sensitivity of Traditional Optimization Methods to Starting Points

The problem is find x : $x * \sin(x) = 24$, $0 \leq x \leq 40$. See Figure 19.

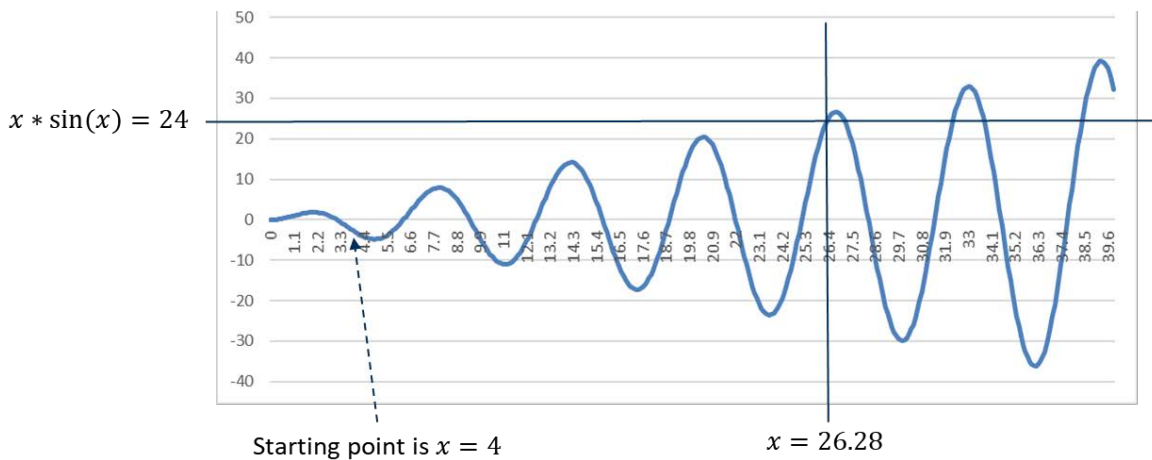


Figure 19. Plot of $x * \sin(x)$

A solution is $x = 26.28$. An encoding of it in the standard A Mathematical Programming Language is:

```
var x: = 4, >= 0, <= 40;
    minimize func: 0;
subject to cond: x * sin(x) = 24;
        solve;
        display x;
        display x * sin(x);
```

Minimizing the objective function $func = 0$ is equivalent to solving the constraint. We uploaded the AMPL model to the Network Enabled Optimization System (NEOS) server at <https://neos-server.org/neos/> for solution. None of the nonlinear solvers, except for Lipschitz-continuous Global Optimizer (LGO), could solve this problem. The reason is solvers often need a starting point to be set. As this is set to 4, the solvers have a hard time traversing the sinusoid to the correct solution.

However, CNMA could solve this problem (within 1% accuracy) as it does not need a starting point.

An interesting use of CNMA could be to find good starting points for interior point nonlinear solvers. The starting point for these solvers is one that is feasible, i.e., satisfies all constraints governing the objective function being optimized.

3.9 Solving Nonlinear Constraints

We show that despite using a MILP solver, CNMA allows P to be nonlinear. This is accomplished by adding, for each nonlinear expression in P , an extra output to the neural network for F and then expressing P as a linear constraint on these outputs. We show how to solve a set of 10 nonlinear constraints on transcendental functions of quadratic expressions, each of the form $0 \geq \exp[i]$. This is a nonlinear optimization benchmark [30]. An example is:

```
subject to f1:
  0 >= (exp((x1-sin(0.0+1.0+1.0))*(x1-sin(0.0+1.0+1.0)))) +
  0.5*(exp((x2-sin(0.0+2.0+2.0))*(x2-sin(0.0+2.0+2.0)))) +
  0.3333333333333333*(exp((x3-sin(0.0+3.0+3.0))*(x3-sin(0.0+3.0+3.0)))) +
  0.25*(exp((x4-sin(0.0+4.0+4.0))*(x4-sin(0.0+4.0+4.0)))) +
  0.2*(exp((x5-sin(0.0+5.0+5.0))*(x5-sin(0.0+5.0+5.0)))) +
  0.1666666666666666*(exp((x6-sin(0.0+6.0+6.0))*(x6-sin(0.0+6.0+6.0)))) +
  0.14285714285714285*(exp((x7-sin(0.0+7.0+7.0))*(x7-sin(0.0+7.0+7.0)))) +
  0.125*(exp((x8-sin(0.0+8.0+8.0))*(x8-sin(0.0+8.0+8.0)))) +
  0.1111111111111111*(exp((x9-sin(0.0+9.0+9.0))*(x9-sin(0.0+9.0+9.0)))) +
  0.1*(exp((x10-sin(0.0+10.0+10.0))*(x10-sin(0.0+10.0+10.0)))) +
  0.09090909090909091*(exp((x11-sin(0.0+11.0+11.0))*(x11-sin(0.0+11.0+11.0)))) -
  u;
```

There are altogether 12 variables, and the objective is to find the minimum value of the variable u satisfying the constraints. We create a neural network with 12 inputs $X = (x_1, \dots, x_{11}, u)$, $x_i \in [-1, 1]$, $u \in [-1, 10]$ and 10 outputs $Y = [y_1, \dots, y_{10}]$, each y_i representing the value of $\exp[i]$. Now:

$$P(x_1, \dots, x_{11}, u) = 0 \geq y_1 \wedge \dots \wedge 0 \geq y_{10} \wedge u \leq 6.5$$

With a neural network architecture (12, 50, 50, 10), an initial training set of 100 samples, CNMA solved this problem in 3 iterations to produce $u = 6.18$. The true minimum is 5.99

We also tried to minimize u : $P(x_1, \dots, x_{11}, u) = 0 \geq y_1 \wedge \dots \wedge 0 \geq y_{10}$. This time, however, CNMA did not return even after 120 iterations. However, this problem can be solved in a couple of seconds by the LOQO solver at the neos-server.org.

The full problem definition is:

```
# Problem :
# *****
# A nonlinear minmax problem in eleven variables.
# Note: the original statement of the problem contains an inconsistant
#       index i. This has been replaced by 1, assuming a very common typo.
#       But the optimal solution of the resulting problem differs from that
#       quoted in the source.
# Source:
# E. Polak, D.H. Mayne and J.E. Higgins,
# "Superlinearly convergent algorithm for min-max problems"
# JOTA 69, pp. 407-439, 1991.
# SIF input: Ph. Toint, Nov 1993.
# classification LOR2-AN-12-10
# All variables are free
# Solution
    param im1 := -1 + (10);
    param rim1 := 9.0;
    param jm1 := -1 + (11);
    param rjm1 := 11;
    param rj := 11;

    var x1 := 1.0;
    var x2 := 1.0;
    var x3 := 1.0;
    var x4 := 1.0;
    var x5 := 1.0;
    var x6 := 1.0;
    var x7 := 1.0;
    var x8 := 1.0;
    var x9 := 1.0;
    var x10 := 1.0;
    var x11 := 1.0;
    var u := 1.0;

minimize obj:
    u;
```

subject to f1:

```
0 >= (exp((x1-sin(0.0+1.0+1.0))*(x1-sin(0.0+1.0+1.0)))) +  
0.5*(exp((x2-sin(0.0+2.0+2.0))*(x2-sin(0.0+2.0+2.0)))) +  
0.3333333333333333*(exp((x3-sin(0.0+3.0+3.0))*(x3-sin(0.0+3.0+3.0)))) +  
0.25*(exp((x4-sin(0.0+4.0+4.0))*(x4-sin(0.0+4.0+4.0)))) +  
0.2*(exp((x5-sin(0.0+5.0+5.0))*(x5-sin(0.0+5.0+5.0)))) +  
0.1666666666666666*(exp((x6-sin(0.0+6.0+6.0))*(x6-sin(0.0+6.0+6.0)))) +  
0.14285714285714285*(exp((x7-sin(0.0+7.0+7.0))*(x7-sin(0.0+7.0+7.0)))) +  
0.125*(exp((x8-sin(0.0+8.0+8.0))*(x8-sin(0.0+8.0+8.0)))) +  
0.1111111111111111*(exp((x9-sin(0.0+9.0+9.0))*(x9-sin(0.0+9.0+9.0)))) +  
0.1*(exp((x10-sin(0.0+10.0+10.0))*(x10-sin(0.0+10.0+10.0)))) +  
0.09090909090909091*(exp((x11-sin(0.0+11.0+11.0))*(x11-sin(0.0+11.0+11.0)))) -  
u;
```

subject to f2:

```
0 >= (exp((x1-sin(1.0+1.0+1.0))*(x1-sin(1.0+1.0+1.0)))) +  
0.5*(exp((x2-sin(1.0+2.0+2.0))*(x2-sin(1.0+2.0+2.0)))) +  
0.3333333333333333*(exp((x3-sin(1.0+3.0+3.0))*(x3-sin(1.0+3.0+3.0)))) +  
0.25*(exp((x4-sin(1.0+4.0+4.0))*(x4-sin(1.0+4.0+4.0)))) +  
0.2*(exp((x5-sin(1.0+5.0+5.0))*(x5-sin(1.0+5.0+5.0)))) +  
0.1666666666666666*(exp((x6-sin(1.0+6.0+6.0))*(x6-sin(1.0+6.0+6.0)))) +  
0.14285714285714285*(exp((x7-sin(1.0+7.0+7.0))*(x7-sin(1.0+7.0+7.0)))) +  
0.125*(exp((x8-sin(1.0+8.0+8.0))*(x8-sin(1.0+8.0+8.0)))) +  
0.1111111111111111*(exp((x9-sin(1.0+9.0+9.0))*(x9-sin(1.0+9.0+9.0)))) +  
0.1*(exp((x10-sin(1.0+10.0+10.0))*(x10-sin(1.0+10.0+10.0)))) +  
0.09090909090909091*(exp((x11-sin(1.0+11.0+11.0))*(x11-sin(1.0+11.0+11.0)))) -  
u;
```

subject to f3:

```
0 >= (exp((x1-sin(2.0+1.0+1.0))*(x1-sin(2.0+1.0+1.0)))) +  
0.5*(exp((x2-sin(2.0+2.0+2.0))*(x2-sin(2.0+2.0+2.0)))) +  
0.3333333333333333*(exp((x3-sin(2.0+3.0+3.0))*(x3-sin(2.0+3.0+3.0)))) +  
0.25*(exp((x4-sin(2.0+4.0+4.0))*(x4-sin(2.0+4.0+4.0)))) +  
0.2*(exp((x5-sin(2.0+5.0+5.0))*(x5-sin(2.0+5.0+5.0)))) +  
0.1666666666666666*(exp((x6-sin(2.0+6.0+6.0))*(x6-sin(2.0+6.0+6.0)))) +  
0.14285714285714285*(exp((x7-sin(2.0+7.0+7.0))*(x7-sin(2.0+7.0+7.0)))) +  
0.125*(exp((x8-sin(2.0+8.0+8.0))*(x8-sin(2.0+8.0+8.0)))) +  
0.1111111111111111*(exp((x9-sin(2.0+9.0+9.0))*(x9-sin(2.0+9.0+9.0)))) +  
0.1*(exp((x10-sin(2.0+10.0+10.0))*(x10-sin(2.0+10.0+10.0)))) +  
0.09090909090909091*(exp((x11-sin(2.0+11.0+11.0))*(x11-sin(2.0+11.0+11.0)))) -  
u;
```

subject to f4:

```
0 >= (exp((x1-sin(3.0+1.0+1.0))*(x1-sin(3.0+1.0+1.0)))) +
0.5*(exp((x2-sin(3.0+2.0+2.0))*(x2-sin(3.0+2.0+2.0)))) +
0.3333333333333333*(exp((x3-sin(3.0+3.0+3.0))*(x3-sin(3.0+3.0+3.0)))) +
0.25*(exp((x4-sin(3.0+4.0+4.0))*(x4-sin(3.0+4.0+4.0)))) +
0.2*(exp((x5-sin(3.0+5.0+5.0))*(x5-sin(3.0+5.0+5.0)))) +
0.1666666666666666*(exp((x6-sin(3.0+6.0+6.0))*(x6-sin(3.0+6.0+6.0)))) +
0.14285714285714285*(exp((x7-sin(3.0+7.0+7.0))*(x7-sin(3.0+7.0+7.0)))) +
0.125*(exp((x8-sin(3.0+8.0+8.0))*(x8-sin(3.0+8.0+8.0)))) +
0.1111111111111111*(exp((x9-sin(3.0+9.0+9.0))*(x9-sin(3.0+9.0+9.0)))) +
0.1*(exp((x10-sin(3.0+10.0+10.0))*(x10-sin(3.0+10.0+10.0)))) +
0.09090909090909091*(exp((x11-sin(3.0+11.0+11.0))*(x11-sin(3.0+11.0+11.0)))) -
u;
```

subject to f5:

```
0 >= (exp((x1-sin(4.0+1.0+1.0))*(x1-sin(4.0+1.0+1.0)))) +
0.5*(exp((x2-sin(4.0+2.0+2.0))*(x2-sin(4.0+2.0+2.0)))) +
0.3333333333333333*(exp((x3-sin(4.0+3.0+3.0))*(x3-sin(4.0+3.0+3.0)))) +
0.25*(exp((x4-sin(4.0+4.0+4.0))*(x4-sin(4.0+4.0+4.0)))) +
0.2*(exp((x5-sin(4.0+5.0+5.0))*(x5-sin(4.0+5.0+5.0)))) +
0.1666666666666666*(exp((x6-sin(4.0+6.0+6.0))*(x6-sin(4.0+6.0+6.0)))) +
0.14285714285714285*(exp((x7-sin(4.0+7.0+7.0))*(x7-sin(4.0+7.0+7.0)))) +
0.125*(exp((x8-sin(4.0+8.0+8.0))*(x8-sin(4.0+8.0+8.0)))) +
0.1111111111111111*(exp((x9-sin(4.0+9.0+9.0))*(x9-sin(4.0+9.0+9.0)))) +
0.1*(exp((x10-sin(4.0+10.0+10.0))*(x10-sin(4.0+10.0+10.0)))) +
0.09090909090909091*(exp((x11-sin(4.0+11.0+11.0))*(x11-sin(4.0+11.0+11.0)))) -
u;
```

subject to f6:

```
0 >= (exp((x1-sin(5.0+1.0+1.0))*(x1-sin(5.0+1.0+1.0)))) +
0.5*(exp((x2-sin(5.0+2.0+2.0))*(x2-sin(5.0+2.0+2.0)))) +
0.3333333333333333*(exp((x3-sin(5.0+3.0+3.0))*(x3-sin(5.0+3.0+3.0)))) +
0.25*(exp((x4-sin(5.0+4.0+4.0))*(x4-sin(5.0+4.0+4.0)))) +
0.2*(exp((x5-sin(5.0+5.0+5.0))*(x5-sin(5.0+5.0+5.0)))) +
0.1666666666666666*(exp((x6-sin(5.0+6.0+6.0))*(x6-sin(5.0+6.0+6.0)))) +
0.14285714285714285*(exp((x7-sin(5.0+7.0+7.0))*(x7-sin(5.0+7.0+7.0)))) +
0.125*(exp((x8-sin(5.0+8.0+8.0))*(x8-sin(5.0+8.0+8.0)))) +
0.1111111111111111*(exp((x9-sin(5.0+9.0+9.0))*(x9-sin(5.0+9.0+9.0)))) +
0.1*(exp((x10-sin(5.0+10.0+10.0))*(x10-sin(5.0+10.0+10.0)))) +
0.09090909090909091*(exp((x11-sin(5.0+11.0+11.0))*(x11-sin(5.0+11.0+11.0)))) -
u;
```

```

subject to f7:
    0 >= (exp((x1-sin(6.0+1.0+1.0))*(x1-sin(6.0+1.0+1.0)))) +
    0.5*(exp((x2-sin(6.0+2.0+2.0))*(x2-sin(6.0+2.0+2.0)))) +
    0.3333333333333333*(exp((x3-sin(6.0+3.0+3.0))*(x3-sin(6.0+3.0+3.0)))) +
    0.25*(exp((x4-sin(6.0+4.0+4.0))*(x4-sin(6.0+4.0+4.0)))) +
    0.2*(exp((x5-sin(6.0+5.0+5.0))*(x5-sin(6.0+5.0+5.0)))) +
    0.1666666666666666*(exp((x6-sin(6.0+6.0+6.0))*(x6-sin(6.0+6.0+6.0)))) +
    0.14285714285714285*(exp((x7-sin(6.0+7.0+7.0))*(x7-sin(6.0+7.0+7.0)))) +
    0.125*(exp((x8-sin(6.0+8.0+8.0))*(x8-sin(6.0+8.0+8.0)))) +
    0.1111111111111111*(exp((x9-sin(6.0+9.0+9.0))*(x9-sin(6.0+9.0+9.0)))) +
    0.1*(exp((x10-sin(6.0+10.0+10.0))*(x10-sin(6.0+10.0+10.0)))) +
    0.09090909090909091*(exp((x11-sin(6.0+11.0+11.0))*(x11-sin(6.0+11.0+11.0)))) -
    u;

subject to f8:
    0 >= (exp((x1-sin(7.0+1.0+1.0))*(x1-sin(7.0+1.0+1.0)))) +
    0.5*(exp((x2-sin(7.0+2.0+2.0))*(x2-sin(7.0+2.0+2.0)))) +
    0.3333333333333333*(exp((x3-sin(7.0+3.0+3.0))*(x3-sin(7.0+3.0+3.0)))) +
    0.25*(exp((x4-sin(7.0+4.0+4.0))*(x4-sin(7.0+4.0+4.0)))) +
    0.2*(exp((x5-sin(7.0+5.0+5.0))*(x5-sin(7.0+5.0+5.0)))) +
    0.1666666666666666*(exp((x6-sin(7.0+6.0+6.0))*(x6-sin(7.0+6.0+6.0)))) +
    0.14285714285714285*(exp((x7-sin(7.0+7.0+7.0))*(x7-sin(7.0+7.0+7.0)))) +
    0.125*(exp((x8-sin(7.0+8.0+8.0))*(x8-sin(7.0+8.0+8.0)))) +
    0.1111111111111111*(exp((x9-sin(7.0+9.0+9.0))*(x9-sin(7.0+9.0+9.0)))) +
    0.1*(exp((x10-sin(7.0+10.0+10.0))*(x10-sin(7.0+10.0+10.0)))) +
    0.09090909090909091*(exp((x11-sin(7.0+11.0+11.0))*(x11-sin(7.0+11.0+11.0)))) -
    u;

subject to f9:
    0 >= (exp((x1-sin(8.0+1.0+1.0))*(x1-sin(8.0+1.0+1.0)))) +
    0.5*(exp((x2-sin(8.0+2.0+2.0))*(x2-sin(8.0+2.0+2.0)))) +
    0.3333333333333333*(exp((x3-sin(8.0+3.0+3.0))*(x3-sin(8.0+3.0+3.0)))) +
    0.25*(exp((x4-sin(8.0+4.0+4.0))*(x4-sin(8.0+4.0+4.0)))) +
    0.2*(exp((x5-sin(8.0+5.0+5.0))*(x5-sin(8.0+5.0+5.0)))) +
    0.1666666666666666*(exp((x6-sin(8.0+6.0+6.0))*(x6-sin(8.0+6.0+6.0)))) +
    0.14285714285714285*(exp((x7-sin(8.0+7.0+7.0))*(x7-sin(8.0+7.0+7.0)))) +
    0.125*(exp((x8-sin(8.0+8.0+8.0))*(x8-sin(8.0+8.0+8.0)))) +
    0.1111111111111111*(exp((x9-sin(8.0+9.0+9.0))*(x9-sin(8.0+9.0+9.0)))) +
    0.1*(exp((x10-sin(8.0+10.0+10.0))*(x10-sin(8.0+10.0+10.0)))) +
    0.09090909090909091*(exp((x11-sin(8.0+11.0+11.0))*(x11-sin(8.0+11.0+11.0)))) -
    u;

subject to f10:
    0 >= (exp((x1-sin(9.0+1.0+1.0))*(x1-sin(9.0+1.0+1.0)))) +
    0.5*(exp((x2-sin(9.0+2.0+2.0))*(x2-sin(9.0+2.0+2.0)))) +
    0.3333333333333333*(exp((x3-sin(9.0+3.0+3.0))*(x3-sin(9.0+3.0+3.0)))) +
    0.25*(exp((x4-sin(9.0+4.0+4.0))*(x4-sin(9.0+4.0+4.0)))) +
    0.2*(exp((x5-sin(9.0+5.0+5.0))*(x5-sin(9.0+5.0+5.0)))) +
    0.1666666666666666*(exp((x6-sin(9.0+6.0+6.0))*(x6-sin(9.0+6.0+6.0)))) +
    0.14285714285714285*(exp((x7-sin(9.0+7.0+7.0))*(x7-sin(9.0+7.0+7.0)))) +
    0.125*(exp((x8-sin(9.0+8.0+8.0))*(x8-sin(9.0+8.0+8.0)))) +
    0.1111111111111111*(exp((x9-sin(9.0+9.0+9.0))*(x9-sin(9.0+9.0+9.0)))) +
    0.1*(exp((x10-sin(9.0+10.0+10.0))*(x10-sin(9.0+10.0+10.0)))) +
    0.09090909090909091*(exp((x11-sin(9.0+11.0+11.0))*(x11-sin(9.0+11.0+11.0)))) -
    u;

```

3.10 Using Neural Network Compression With CNMA

Two possible challenges with CNMA are having to choose a good neural network architecture and being able to solve the MILP problem quickly. To address these challenges, we tried using SCANN, a method that automatically selects an architecture to generate a compact, yet accurate neural network. The chosen architecture is always feed-forward, but does not necessarily follow a multilayer perceptron structure. We tried using CNMA with SCANN's Scheme A algorithm, where it starts with a small neural network and gradually increases the number of hidden neurons and connections. To evaluate CNMA's performance using SCANN, we ran CNMA multiple times with different chosen architectures and with SCANN. Our experiments showed that the number of iterations and function calls needed actually increased slightly when using SCANN. Additionally, due to the increased training time when using SCANN, the time taken to solve a query significantly increased. On average, it took 4159 seconds to solve a query with SCANN, while without SCANN, it took 38.57 seconds. We also did not see any improvement in MILP solve times (~.1 seconds when manually choosing the architecture vs ~.33 seconds with SCANN). We believe this is because SCANN may produce neural networks with multiple hidden layers, which can slow down the MILP solver. See Figure 20.

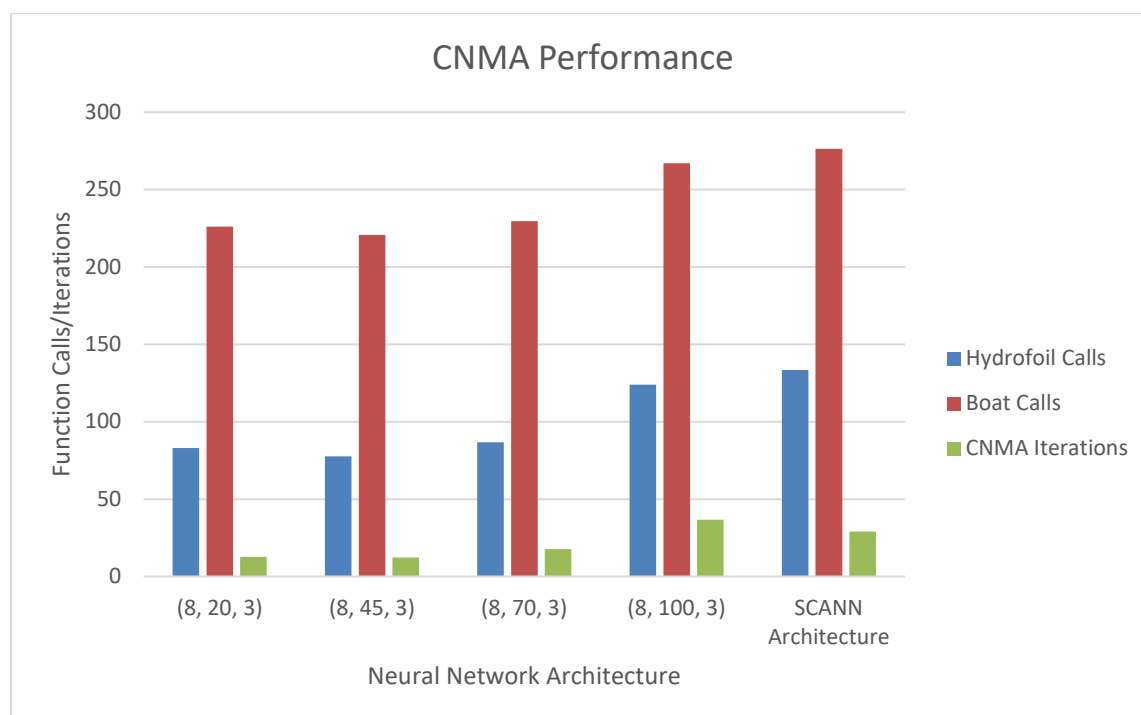


Figure 20. CNMA Performance With and Without Compression

Because using SCANN significantly increases the time it takes to train the neural network (~2 minutes vs <1 second), we also tried incremental training. During each iteration of CNMA, rather than train a neural network from scratch, the training of the neural network from the previous iteration is continued, with an additional data point. To further decrease training times, we also tried tuning a couple of SCANN's hyper parameters that affect how much training is done. For each of the four different ways we used SCANN, we ran CNMA three times. CNMA was stopped if no solution was found within two hours. Solutions were always found

when the neural networks were retrained from scratch, but this was not always the case for incremental training. The average number of hydrofoil calls, boat calls, CNMA iterations taken, and training times are plotted below. While both incrementally training and tuning the hyper parameters improved the neural network training time, the number of iterations and the number of function calls increased. See Figure 21 and Figure 22.

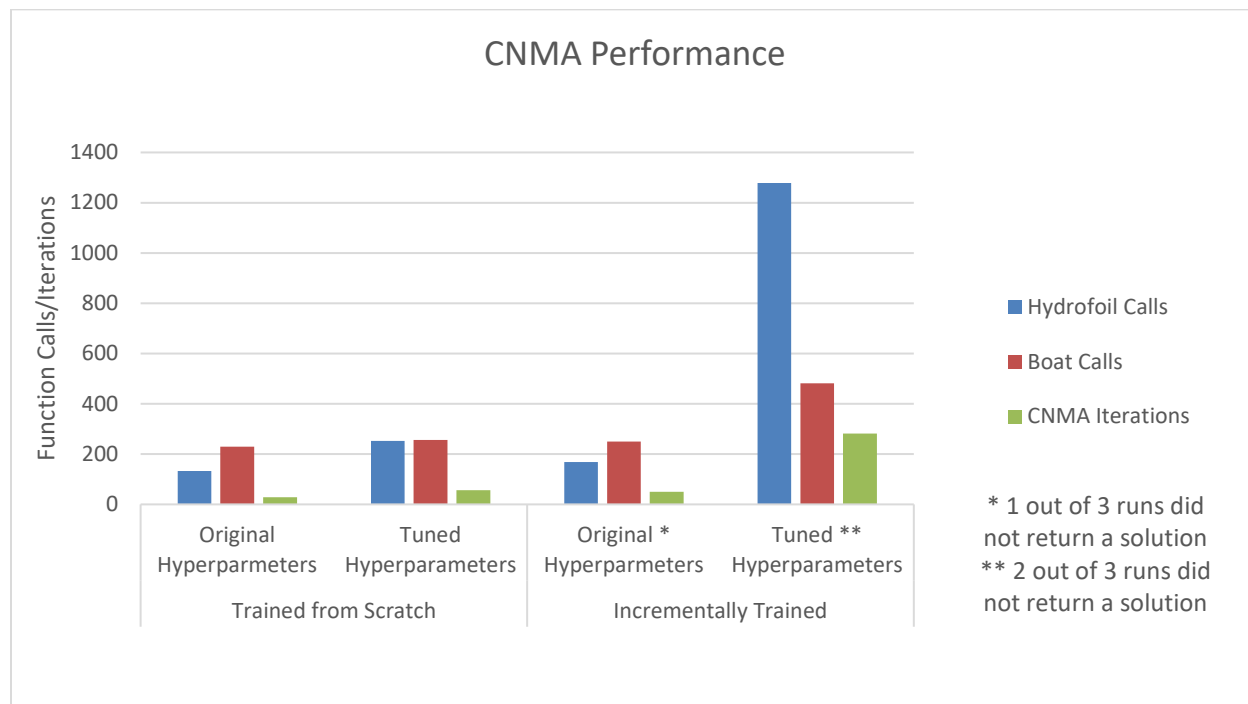


Figure 21. Number of CNMA Iterations With Different Types of SCANN Tuning

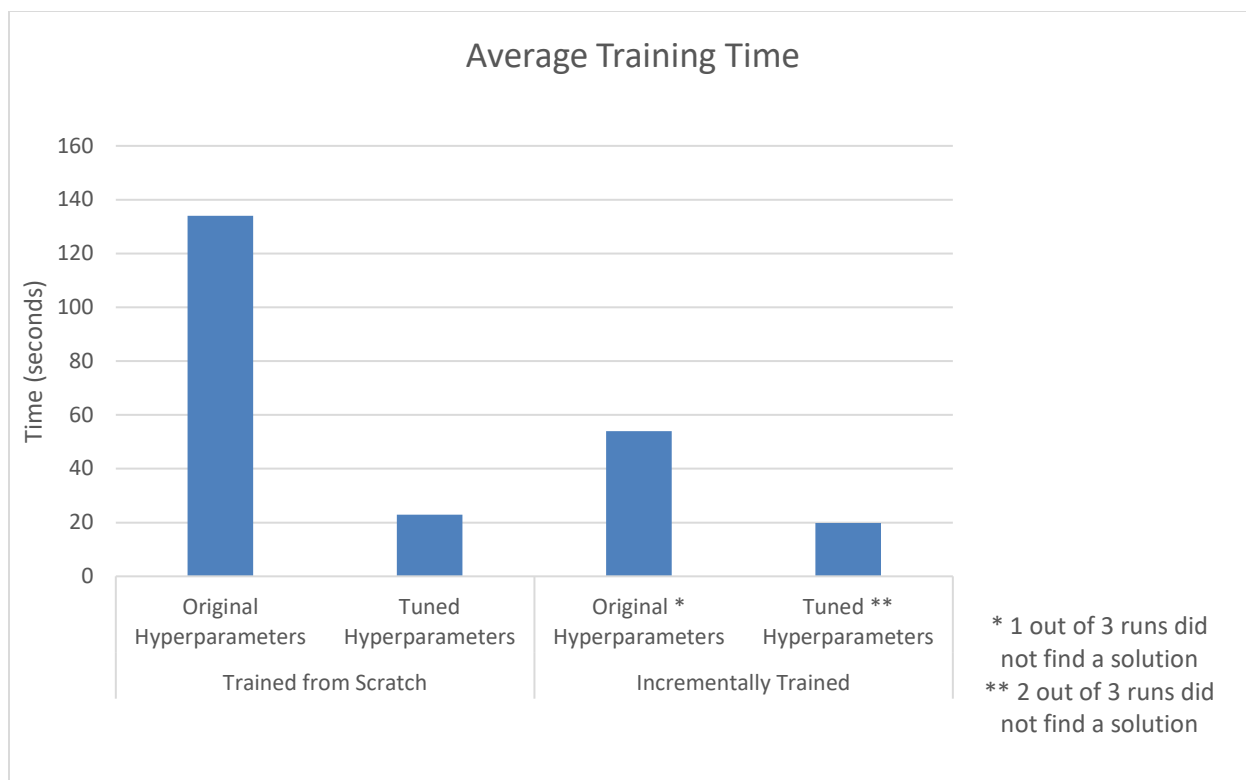


Figure 22. Average Training Time With and Without SCANN Tuning

The average times it takes to solve a query are also plotted in Figure 23. Even though training from scratch and using the tuned hyperparameters took more iterations than using the original hyperparameters, due to the much quicker training times, training from scratch using the tuned hyperparameters finds a solution the quickest. However, using CNMA without SCANN is still much quicker (38.57 seconds vs. 1551.28 seconds).

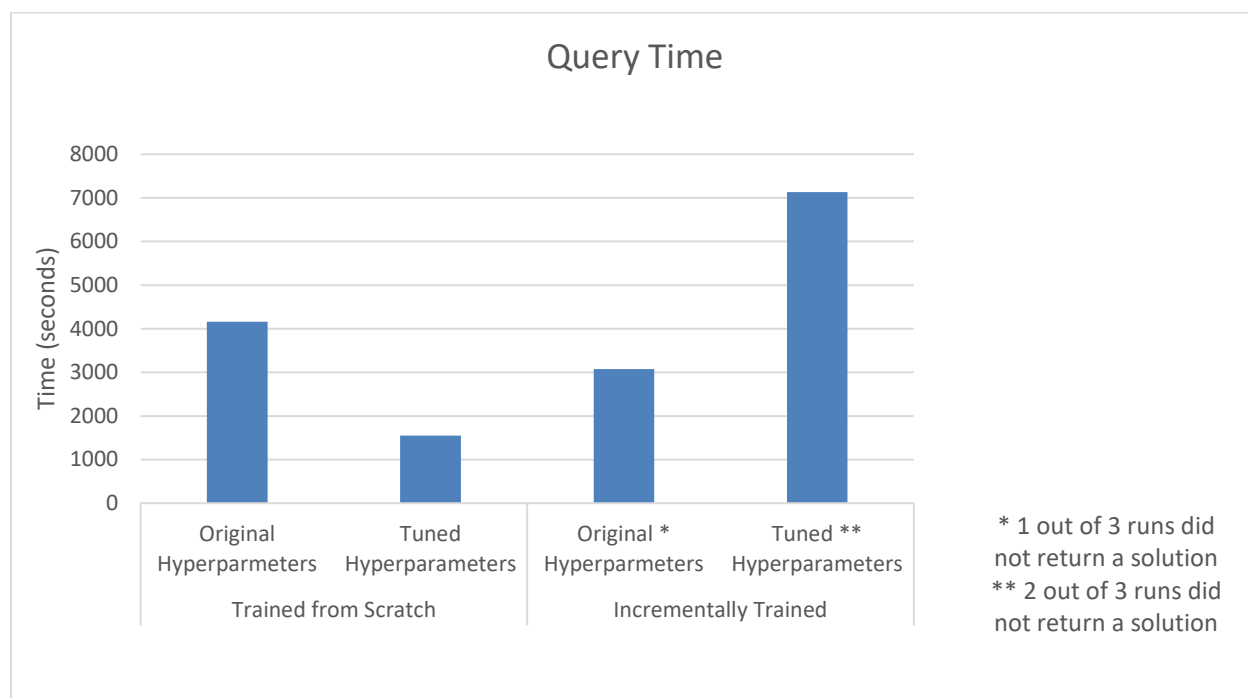


Figure 23. Times Taken To Solve Water Craft Design Query With Compression

Overall, compression did not yield the benefits to CNMA we had hoped for. However, this may change as we deal with much larger systems that need much larger neural networks to model.

4 Conclusions

The goal of this effort was to evaluate the feasibility of creating a toolkit for design-space exploration of cyber physical systems and identify metrics for evaluating these. CPS are marked by the presence of discrete and continuous inputs and outputs usually governed by complex nonlinear relationships. CPS need not be just physical. They can also be logical e.g., parallel parking or obstacle avoiding robot maneuvers. Such a toolkit would allow a designer to specify requirements on design and performance. It would then automatically suggest designs satisfying requirements. It would also allow exploration of design tradeoffs. Tasks that such a toolkit needs to accomplish include:

- Synthesis, a.k.a. “inverse design”
- Composition of heterogeneous CPS subsystem models
- Scalability a.k.a. alleviating curse of dimensionality
- Minimizing number of calls to expensive design/blackbox tools
- Resilience to design tool failure
- Non-parametric synthesis
- Solving nonlinear constraints
- Computing global optima

Metrics for evaluating CPS design-space exploration toolkits can be based on how well they accomplish these tasks.

Most relevant to building such a toolkit are the mature nonlinear optimization tools. These optimize an objective function subject to constraints. However, these have several shortcomings:

- Derivative-based methods [28, 20] are infeasible when the cost of evaluating functions is high. Derivative computation can dramatically increase the number of function evaluations.
- Derivative-free methods [29, 31, 24] are conservative in the number of times they evaluate functions. They build surrogate functions based on current samples and use those to predict the best point to sample next. But, their complexity is high, for example that of Gaussian process is $O(n^3)$ where n is the sample size [24].
- Derivative-free methods are not resilient to design tool failure. The information gaps caused by such failures disrupt surrogate construction. Such methods are also inadequate as constraint solvers.
- Constrained optimization accomplishes constraint solving only indirectly by reducing it to unconstrained optimization. This is done by use of penalty or barrier functions encoding the cost of violating constraints. However, construction of such functions requires considerable creativity on the part of the user [17, 27, 20].
- Optimization methods that require starting points are highly sensitive to their choice. It is hard to find good starting points in high-dimensional spaces.
- Interior point optimization methods require the starting point to be within the feasible region – in other words, they need to solve constraints before even beginning.

- There is no obvious way of specifying recursive relationships. Such specification can be avoided by flattening the design space into a high-dimensional one, but that gives rise to the curse of dimensionality: we have to solve a single optimization problem, rather than being able to break it up into simpler ones and combining their solutions.

This research effort has proposed CNMA, a new approach for building such a toolkit, that overcomes the above shortcomings. CNMA solves the fundamental problem of finding x, y such that $F(x) = y \wedge P(x, y)$ where F is a possibly nonlinear function modeling a CPS, x is a design vector of discrete and continuous values, y is the associated output vector of discrete and continuous values and P is a possibly nonlinear design constraint on both x, y . CNMA is based on the idea that the constraint can be solved by learning F as a neural network, converting it into an equivalent mixed integer linear program, and solving it along with P with industrial-strength MILP solvers. However, since learning is always approximate, the solver can return incorrect results. Adding a new error-correction step focusses the learning on that part of F 's domain that is relevant to solving the constraint. Thus, the learning is reduced by orders of magnitude.

CNMA has been evaluated as follows:

- We used it to explore designs of a water craft propelled solely with wave energy. It is a moderately high-dimensional problem with 13 design variables. Such design requires solutions to all of the problems listed above. We have shown how CNMA can efficiently accomplish those tasks. In particular, a design was obtained in about 30 seconds with ~ 100 calls to the water craft simulators. To create a sufficiently accurate neural network model of the water craft without CNMA would require $\sim 10^6$ calls to these simulators. The water craft is inspired by the Wave Glider [6, 8].
- We formulated the water craft design problem with two mature nonlinear, derivative-free optimization methods: Gaussian Processes [24] and COBYLA [31], and have shown that CNMA's performance in terms of several metrics, time to compute design, number of function calls and resilience to design tool failure, is far superior. Gaussian Processes did not even terminate. COBYLA did but took about 1348 calls to the simulators.
- We did *not* evaluate derivative-based methods as they greatly increase the number of function evaluations.
- We further evaluated CNMA against traditional nonlinear optimization benchmarks. One of these has constraints on transcendental functions of quadratic expressions [30]. When functions to be optimized are amenable to mathematical analysis, current methods significantly outperform CNMA. However, because they are also sensitive to starting points, they can sometimes fail to produce answers, whereas CNMA will produce an answer if it exists.
- We have shown how, despite using an MILP solver (where L stands for Linear), CNMA allows P to be nonlinear. This is accomplished by adding, for each nonlinear expression in P , an extra output to the neural network for F and then expressing P as a linear constraint on these outputs.
- We have shown how CNMA can be used to synthesize logical CPS such as parallel parking and obstacle avoiding robot maneuvers. Such problems are traditionally solved

with temporal-logic based synthesis engines. These examples show we might reduce them to non-temporal constraint solving by abstracting away time.

- We have shown how CNMA can be used to solve the synthesis problem for a CPS with both discrete and continuous variables. This problem is that of identifying a failure pattern in a smart grid that is consistent with sensor readings at a limited number of points in the smart-grid.

Based on this study, we believe CNMA is a viable approach for creating a toolkit for design-space exploration of cyber physical systems.

5 Recommendations

We recommend exploring the following new research directions:

- **Scalability.** While CNMA can efficiently design a CPS with 13 design variables, what innovations, if any, are needed to scale the size by at least an order of magnitude to e.g., 100 variables?
- **Combining strengths of existing optimization tools with CNMA.** For example, CNMA could be used to find feasible starting points for interior-point methods. CNMA could also be integrated into SMT solvers and temporal-logic based synthesis tools, both of which need nonlinear constraint solving.
- **Extending CNMA to neural networks with general activation functions.** At present, CNMA uses neural networks with the ReLU activation function to permit their transformation into equivalent MILPs. When other activation functions such as tanh or sigmoid are more appropriate, we could use traditional nonlinear solvers while retaining the benefits of CNMA.
- **Extending CNMA to new neural network architectures.** At present, only fully connected, static neural networks are used. What new CPS designs are possible if we could use Convolutional Neural Networks, Long Short-Term Memory, or Recurrent Neural Networks e.g., [21]?
- **Automatically identifying hyper parameters.** At present, neural network hyper parameters, as well as the size of the initial training set are manually selected. Can we automatically identify these?
- **CNMA with Diversity.** Can we use an ensemble of neural networks with different hyper parameters to accelerate solution convergence? Each neural network may learn a different aspect of the function.
- **UNSAT-Core.** At present, CNMA does not terminate if the constraint is unsatisfiable. It keeps learning new points in the domain of the function in the hopes that a better approximation would let it solve the constraint. How do we overcome this significant limitation of CNMA?
- **Fluent user interfaces.** Can we make the design-space experience fluent e.g., in MIT's Northstar system [26]?
- **Identifying new application domains.** We are talking to experts in satellite design and propulsion. All we need are simulators for a domain that map designs into outputs. Then, CNMA can assist with design-space exploration.
- **Using neural network compression.** Compression of a neural network produces another whose size is much smaller than that of the original, but whose accuracy is the same. The MILP equivalents of smaller neural networks are much easier to solve. We have tried to use one such compression method (SCANN) [1] but found that while the compressed neural network was solved in about the same time as the original, the overhead of compression at each CNMA iteration was considerable. Thus, the current implementation of CNMA does not use compression. This may change as we deal with much larger neural network for realistic CPS. For example, NeST system compresses images [11].

6 References

1. Z. Wang, N. Jha. SCANN: Synthesis of Compact and Accurate Neural Networks. Princeton University
2. B. Englot. Solidworks Modeling of Wave Glider-Like System. Stevens-Tech
3. K. Pochiraju. Model Based Systems Engineering with Digital Twins and Physical Prototypes. Steven-Tech
4. Guy Katz, Clark Barrett, David Dill, Kyle Julian, Mykel Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. <https://arxiv.org/abs/1702.01135>
5. Lomuscio, L. Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. <https://arxiv.org/abs/1706.07351>.
6. Liquid Robotics. Energy harvesting ocean robot. <https://www.liquid-robotics.com/wave-glider/how-it-works/>
7. N. Bjørner, A.-D. Phan, L. Fleckenstein. vZ - An Optimizing SMT Solver <https://www.microsoft.com/en-us/research/wpcontent/uploads/2016/02/nbjorner-nuz.pdf>.
8. Nicholas Kraus. Wave glider dynamic modeling, parameter identification and simulation. M.S. Thesis, University of Hawaii, Manoa, 2012. <https://scholarspace.manoa.hawaii.edu/handle/10125/101307>
9. R. Kuhn, R. Kacker, Y. Lei. Combinatorial testing. https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=910001
10. Sanjai Narain. Learning And Reasoning About Cyber Physical System Configuration. Internal report, Vencore Labs, 2018.
11. Xiaoliang Dai, Hongxu Yin, Niraj K. Jha. NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm. <https://arxiv.org/abs/1711.02017>
12. Josh Bongard and Hod Lipson. Automated reverse-engineering of non-linear dynamical systems. <http://www.pnas.org/content/104/24/9943>
13. Xfoil. Subsonic Airfoil Development System. <https://web.mit.edu/drela/Public/web/xfoil/>
14. Combinatorial Interaction Testing Portal. <http://cse.unl.edu/~citportal/>
15. Sobol Sequence. https://en.wikipedia.org/wiki/Sobol_sequence
16. Ptolemy Project. Heterogeneous Modeling and Design. <http://ptolemy.berkeley.edu/ptolemyII/>
17. Julia Roberts and Mykel Kochenderfer. Mathematical Optimization. <https://web.stanford.edu/group/sisl/k12/optimization/#!index.md>, 2014. Penalty Methods are discussed at <https://web.stanford.edu/group/sisl/k12/optimization/MO-unit5-pdfs/5.7interiorpenalty.pdf>
18. Luis Miguel Rios · Nikolaos V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. J Glob Optim (2013) 56: 1247. <https://link.springer.com/content/pdf/10.1007%2Fs10898-012-9951-y.pdf>
19. Juliane Mueller. Optimizing Expensive Blackbox Simulations. <https://cs.lbl.gov/assets/CSSSP-Slides/20180719-Mueller.pdf>
20. John Chinnek. Practical Optimization: A Gentle Introduction. <http://www.sce.carleton.ca/faculty/chinneck/po.html>. Nonlinear programming is treated at <https://www.sce.carleton.ca/faculty/chinneck/po/Chapter16.pdf>.

21. Pattarawat Chormai, Felix Sattler, Raphael Holca-Lammare. Blackbox Optimization Using Recurrent Neural Networks. <https://www.ni.tu-berlin.de/fileadmin/fg215/teaching/nnproject/zation-final.pdf>. 2017
22. Olof Skogby Steinholtz. A Comparative Study of Black-box Optimization Algorithms for Tuning of Hyper-parameters in Deep Neural Networks. <https://itu.diva-portal.org/smash/get/diva2:1223709/FULLTEXT01.pdf>. 2018
23. Daniel Golovin, Benjamin Solnik, Subhdeep Moitra, Greg Kochanski, John Karro, D. Sculley. Google Vizier: A Service for Black-Box Optimization. Proceedings of KDD, 2017. https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019/papers/golovin_KDD_2017.pdf
24. Daniele Calandriello, Luigi Carratino, Alessandro Lazaric, Michal Valko, Lorenzo Rosasco. Gaussian Process Optimization with Adaptive Sketching: Scalable and No Regret. March, 2019. <https://arxiv.org/abs/1903.05594>
25. Zeyuan Shang, Emanuel Zraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, Tim Kraska. Democratizing Data Science through Interactive Curation of ML Pipelines. <https://www.shangzeyuan.com/static/papers/2019/sigmod2019-automl.pdf>
26. Northstar: Making Data Science More Interactive. <http://dsail.csail.mit.edu/index.php/vizdom/>
27. Interior Penalty Functions. <https://web.stanford.edu/group/sisl/k12/optimization/MO-unit5-pdfs/5.7interiorpenalty.pdf>
28. S. Rao. Engineering Optimization: Theory and Practice / Edition 4, Wiley. 2009.
29. L. Rio, N. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. <http://thales.cheme.cmu.edu/dfo/comparison/dfo.pdf>
30. Nonlinear Minmax Problem in 11 variables. <https://vanderbei.princeton.edu/ampl/nlmodels/cute/polak3.mod>
31. M.J.D. Powell. A view of algorithms for optimization without derivatives. http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2007_03.pdf
32. Aneesh Chand, Michihiro Kawanishi, Tatsuo Narikiyo. Application of Sigmoidal Gompertz Curves in Reverse Parallel Parking for Autonomous Vehicles. September 2015. International Journal of Advanced Robotic Systems 12(9):1. https://www.researchgate.net/publication/283979807_Application_of_Sigmoidal_Gompertz_Curves_in_Reverse_Parallel_Parking_for_Autonomous_Vehicles
33. Zachary T. Wilson¹ and Nikolaos V. Sahinidis. The ALAMO approach to machine learning. <https://arxiv.org/pdf/1705.10918.pdf>

7 List of Symbols, Abbreviations, and Acronyms

ALAMO	Automatic Learning of Algebraic Models
AMPL	A Mathematical Programming Language
CFD	Computational Fluid Dynamics
CNMA	Constraint Satisfaction with Neural Networks, Mixed Integer Linear Programming and Active Learning
COBYLA	Constrained Optimization by Linear Approximation
CPS	Cyber Physical Systems
EMS	Error Maximization Sampling
LGO	Lipschitz-continuous Global Optimizer
MILP	Mixed Integer Linear Program
MIT	Massachusetts Institute of Technology
NEOS	Network-Enabled Optimization System
NEST	Neural network Synthesis Tool
RBF	Radial Basis Function
ReLU	Rectified Linear Units
SCANN	Synthesis of Compact and Accurate Neural Networks
SMT	Satisfiability Modulo Theories
SUMT	Sequential Unconstrained Minimization Technique
UNSAT	Unsatisfiability

8 Glossary

Active Learning – A form of Machine Learning where new labeled data is obtained during the learning process

CNMA – A new constraint solver that uses neural networks, MILP solvers, and active learning

COBYLA – A derivative free constrained optimization method which uses simplexes to create linear approximations of the objective and constraints

Combinatorial Interaction Testing – A sampling technique that was originally designed to test software

CPLEX – Simplex Method as Implemented in the C Programming Language

Curse of Dimensionality – The problem that when the number of dimensions increase, the amount of data needed to learn the space may increase exponentially

GUROBI – GU, Rothberg, Bixby

Kriging – An interpolation technique using Gaussian Processes that can be used as a surrogate function in Bayesian Optimization

Mixed Integer Linear Programming – Linear programming where some of the variables may be integers

Sobol Sampling – A quasi-random sampling method designed to have better coverage of the input space than random sampling