INL/JOU-21-62519-Revision-0

An Agent-Based Blackboard System for Multi-Objective Optimization

March 2022

Ryan Hunter Stewart, Samuel E Bays, Todd S Palmer

INL/JOU-21-62519-Revision-0

An Agent-Based Blackboard System for Multi-Objective Optimization

Ryan Hunter Stewart, Samuel E Bays, Todd S Palmer

March 2022

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Journal of Computational Design and Engineering, 2022, 9(2), 480–506

https://doi.org/10.1093/jcde/qwac009
Journal homepage: www.jcde.org

RESEARCH ARTICLE

An agent-based blackboard system for multi-objective optimization

Ryan Stewart[1,2,*], Todd S. Palmer[2] and Samuel Bays[1]

[1]Idaho National Laboratory, 2525 N Freemont Avenue, Idaho Falls, ID 83415, USA and [2]Oregon State University, School of Nuclear Science and Engineering, 1500 SW Jefferson Street, Corvallis, OR 97331, USA

*Corresponding author. E-mail: ryan.stewart@inl.gov http://orcid.org/0000-0003-4867-6555

Abstract

In the field of multi-objective optimization, there are a multitude of algorithms from which to choose. Each algorithm has strengths and weaknesses associated with the mechanics for finding the Pareto front. Recently, researchers have begun to examine how multi-agent environments can be used to help solve multi-objective optimization problems. In this work, we propose a multi-objective optimization algorithm based on a multi-agent blackboard system (MABS). The MABS framework allows for multiple agents to read and write pertinent optimization problem data to a central blackboard agent. Agents can stochastically search the design space, use previously discovered solutions to explore local optima, or update and prune the Pareto front. A centralized blackboard framework allows the optimization problem to be solved in a cohesive manner and permits stopping, restarting, or updating the optimization problem. The MABS framework is tested against three alternative optimization algorithms across a suite of engineering design problems and typically outperforms the other algorithms in discovering the Pareto front. A parallelizability study is performed where we find that the MABS is able to evaluate a set number of designs, which require an evaluation time ranging from 0 to 300 seconds, quicker than a traditional optimization algorithm: this fact becomes more apparent the longer it takes to evaluate a design. To provide context for the benefits provided by MABS, a real-world nuclear engineering design problem is examined. MABS is used to examine the placement of experiments in a nuclear reactor, where we are able to evaluate hundreds of configurations for experimental placement while maintaining a strict set of safety constraints.

Keywords: blackboard architecture; agent-based modeling; multi-objective optimization; nuclear reactor design; high-performance computing

1 Introduction

Multi-objective optimization algorithms are employed in multiple fields of science and engineering to determine ideal solutions based on a set of objectives and constraints. Research in this field has generated hundreds of distinct algorithms, variations, and architectures for solving multi-objective optimization problems. Metaheuristic methods tend to dominate the field of multi-objective optimization, as they are typically easy to implement, efficient, and highly parallelizable. These methods can be divided into three major categories: nature-, physics-, and human-inspired algorithms (Stewart *et al.*, 2021b). Nature-inspired algorithms draw inspiration from the living world, and seek to find optimality by mimicking natural processes. Genetic algorithms (GAs; Murata, 1995; Melanie, 1996) follow the processes of natural selection to find and promote optimal solutions. Particle swarm optimization (PSO; Kennedy & Eberhart, 1995; Mostaghim & Teich, 2003) examines the social behavior of animal groups for finding resources. Ant colony optimization (ACO; Dorigo *et al.*, 2006; Alaya *et al.*, 2007) relies on the methods ants use for finding paths to food sources by placing pheromones on the ground. Physics-inspired algorithms rely on physics-based rules or theories to find optimal solutions.

Received: 25 August 2021; **Revised:** 17 November 2021; **Accepted:** 3 December 2021

Simulated annealing (SA; Kirkpatrick *et al.*, 1983; Serafini, 1994) emulates the process of metallurgic annealing to determine optimal solutions. The gravitational search algorithm (GAS; Rashedi *et al.*, 2009; Hassanzadeh & Rouhani, 2010) relies on gravity to attract ideal solutions, where more optimal solutions have a greater gravitational pull. Multiverse optimization (MO; Mirjalili *et al.*, 2015, 2017) utilizes concepts from cosmology such as white holes, black holes, and worm holes to effectively search global and local areas in a problem. Human-inspired algorithms examine human behavior such as social interactions, learning, and competition. The harmony search algorithm (HSA; Lee & Geem, 2004; Sivasubramani & Swarup, 2011) is derived from the ability of jazz musicians to improvise during jam sessions. The imperialistic competition algorithm (ICA; Atashpaz-Gargari & Lucas, 2007; Sherinov & Unveren, 2017) examines how empires compete for dominance over resources and other colonial powers. Tabu search (TS; Glover, 1990; Hansen, 1997) creates sets of rules for selecting future solutions, based on previous steps to ensure repetition is minimized and future movements are ideal.

While numerous optimization algorithms have been developed and tested, the use of multi-agent systems to aid the optimization process is a relatively new field. Hulse *et al.* applied a multi-agent approach for the design of complex and multidisciplinary design problems. The test case examined the optimization of a quadrotor (drone with four propellers) design, where agents were assigned different components to design and optimize (Hulse *et al.*, 2017, 2019). Communication between the agents allowed global design goals to be assessed, where agents were rewarded for producing viable components locally and for the ability of the component to contribute to the overarching objectives of the design. This approach found that the use of a multi-agent optimization methodology was able to outperform a centralized genetic or SA algorithm. Other approaches have examined the use of multiple agents to cooperatively search for an optimal solution in a single objective problem. Gebreslassie *et al.* developed a heterogeneous multi-agent optimization framework for solving process system engineering problems (Gebreslassie & Diwekar, 2018). The multi-agent system allowed agents to use global information in conjunction with a unique optimization algorithm to explore the objective space. The authors examined the ability of this multi-agent system to determine radioactive waste blending for storage. Utilizing a multi-agent system for this process provided a decrease in both the computational times required to solve the problem and increased the quality of optimal solutions. While the work of these two articles is small, compared with the breadth of optimization methods, they indicate that the use of multi-agent environments can aid the optimization process.

While multi-agent environments have begun to diffuse into the field of optimization, another area of interest for computational frameworks is the use of blackboard systems. Blackboard systems are frameworks for storing problems, where agents can interact with the framework to provide help in solving an overarching problem; a more detailed explanation is provided in Section 1.1. Given the use of agents, blackboard systems lend themselves naturally to a multi-agent environment. Szymanski *et al.* examine using a multi-agent blackboard architecture as a method for combing legal precedent to help support legal decision making, with a current focus on traffic violations (Szymanski *et al.*, 2018). The users provide agents with the facts of case (e.g. who was at fault, where they inebriated, what did they hit). Lower levels of the blackboard make up the current facts of the case, where the top level contains a statutory decision based on the facts. Between these layers are intermediate notions that help connect facts to statutory concepts. These intermediate steps can be assisted by precedents, legal rules, and common tips that agents find when examining similar cases. This results in presenting the likely outcome of the case in question, and poses additional questions that may alter the outcome. De Silva *et al.* and Brzykcy *et al.* have also explored the ability of a blackboard framework to maintain information about an unknown terrain that is being explored by various agents (Brzykcy *et al.*, 2001; Silva *et al.*, 2019). Blackboard systems were determined to be a stationary lander, where mobile robots would explore the surrounding areas. Information could be relayed back to the blackboard, where all other agents would gain access to this information upon communicating with the lander. This process allowed agents to have a better representation of the surrounding area to ensure agents did not explore the same area multiple times. Blackboard systems have been applied to various problems, where this environment can help organize data and present a uniform method for approaching problems.

Multi-agent environments have successfully been integrated into both optimization algorithms and blackboard systems. Utilizing these hybrid methods, researchers have found advantages over utilizing either approach independently. We follow these approaches and create a multi-objective optimization algorithm that utilizes a blackboard system in conjunction with a multi-agent environment. This paper examines a human-inspired optimization algorithm known as the multi-agent blackboard system (MABS). MABS imitates the ability of researchers to collectively work toward the optimal solution of an overarching problem. Each researcher has the ability to contribute to the problem; however, they are likely not able to solve the entirety of the problem on their own. These researchers instead write portions of solutions to a blackboard, where other researchers can utilize these results to help find optimal solutions. MABS seeks to leverage the benefits of multi-agent and blackboard systems for distributed problem solving. Similar to exploring an unknown environment, as proposed in Brzykcy *et al.* (2001) and Silva *et al.* (2019), this work seeks to utilize multiple agents to search an unknown terrain, in this case the objective space of an optimization problem. All of this information is stored on the blackboard, where agents can filter the acquired designs and determine the Pareto front (PF).

This approach is in contrast to many currently implemented multi-objective optimization algorithms. Typically, a multi-objective optimization algorithm focuses on two aspects: the ability to discover the true PF and the speed at which the PF can be discovered. The ability to discover the true PF is often examined in the context of benchmark problems (Zitzler, 1999) that are highly complex and can have multiple false PFs. Determining the PF in a timely manner is also important for these algorithms to be used in real-world scenarios. These two aspects are considered for the MABS; however, we seek instead to focus on developing a flexible optimization algorithm that excels at examining engineering-based optimization problems. This framework concentrates on the ability to interact with high-performance computing clusters (HPCs) for evaluating objective functions and constraints by solving high-fidelity simulations that can take on the order of minutes to hours to complete. Solving high-fidelity simulations on HPCs introduces the possibility that simulations may fail; to account for this, the multi-agent system must be robust to ensure that if a single agent fails, the problem overall does not fail. If high-fidelity simulations were to be incorporated into a typical optimization algorithm, a failure in a single evaluation would likely cause the entire problem to fail.

Given the nature of utilizing an HPC to evaluated various designs, it is also important that the optimization algorithm is highly parallelizable to fully leverage the computational resources available. Evaluation of high-fidelity simulations that can take minutes to hours to complete ensures that we want to maximize the number of designs that can be evaluated at any given time, given the available computational resources. Using multiple agents, who are each performing one or more design evaluations, allows MABS to maximize the evaluation rate. MABS can utilizing increasing numbers of agents to increase the parallelizability of the algorithm; compared with a traditional algorithm that would be limited by mechanics in the algorithm (such as the population size for a GA or swarm size in the PSO algorithm).

Introducing a multi-agent environment provides an optimization algorithm that is robust and can easily be incorporated within an HPC; however, additional complexity is also added to the algorithm. This includes the development of the multiple agents, a communication pattern, and a method for keeping track of problem progression. While the use of a blackboard system provides a framework for handling many of these complexities, there are inherently more components than are required for a typical optimization algorithm. MABS is not meant as a generic catch all optimization algorithms, and the justification for adding complexity to the optimization algorithm is to ensure performance when examining engineering-based problems that require full simulations.

To demonstrate the effectiveness of the MABS, we examine its ability to discover the PF for a set of engineering-focused multi-objective optimization benchmark problems and compare the ability to discover the PF with other multi-objective optimization algorithms. This is extended to analyse the parallelizability of the MABS over a variety of simulation times. We also apply the MABS framework to a real-world nuclear engineering design problem to determine the optimal placement of experiments in a test reactor. To the authors' knowledge, there has been no previous research investigating a blackboard system for performing multi-objective optimization.

1.1 Blackboard framework

A blackboard system is defined as a problem-solving algorithm that relies on multiple knowledge agents (KAs) to contribute to the solution of an overarching problem. To gain a general understanding of the system, a common metaphor is used:

> Imagine a group of human specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution. Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialist finds sufficient information to make a contribution, they record their contribution on the blackboard, hopefully enabling other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved (Corkill, 1991).

Each specialist typically cannot solve the problem independently and requires the knowledge and resources of others. The blackboard can be used to solve large, complex, or ill-posed problems as it creates a central knowledge repository and allows KAs access to this information. A blackboard system consists of three components: the blackboard, KAs, and a controller. The blackboard acts as a global database and creates abstract levels for storing problem specific information (partial results, alternatives, input data, etc.) for KAs to utilize (Craig, 1995). Abstract levels are defined by an entry system, where each entry is unique to the abstract level and acts as a template for KAs. The entry system creates a common format from which an agent can read/write. The advent of multiple abstract levels allows KAs to rapidly read and write information to their portion of the blackboard without requiring access to the remaining abstract levels.

The KAs (called knowledge sources in literature; Corkill *et al.*, 1988) provide information or modules required to solve various aspects of the problem. KAs can provide unique approaches to the same problem, solve different portions of an overarching problem, or move information within the blackboard. KAs write their portion of the solution to the blackboard and wait to see whether their method is required again. Once a KA contributes to the blackboard, other KAs may find that they have the required information for them to act.

The controller organizes the problem, dictates the problem flow, and allocates resources for a KA to perform an action. Organizing and controlling the problem flow require the controller to assess how each KA can contribute to the problem. This process is performed via a triggering system, where KAs respond to the trigger with a trigger value (TV; typically some numerical value) to indicate their ability to contribute to the problem. Based on this information, the controller selects the KA that can provide the most information to the problem (indicated by a larger TV) and tells the corresponding KA to perform its action. This provides a structure for solving the problem and allows for a centralized unit to determine the best KAs to be executed at a given stage in the process.

2 Methods

2.1 Multi-agent blackboard system optimization algorithm overview

The MABS framework (Stewart, 2019) is dependent on the Python module `osbrain` for generating a multi-agent environment (OpenSistemas, 2019). `osbrain` allows for the creation of remote agents who perform tasks independently while relying on message passing to advance the problem. The nomenclature for MABS diverges slightly from the definitions in Section 1.1. Each component of the blackboard system is a unique agent, denoted as a blackboard agent (BA), knowledge agent (KA), and controller agent (CA). The remaining sections are laid out as follows: Section 2.2 describes each of these agents in detail, Section 2.3 describes the mechanics of the MABS framework, Section 2.4 describes how the MABS environment is generated, Section 2.5 describes how the MABS solves an optimization problem, and Section 2.6 describes the versatility and benefits of the MABS for solving optimization problems.

2.2 Agent descriptions

2.2.1 *Blackboard agents*

The BA serves two primary functions: storing the blackboard and communicating with the KAs. For a basic multi-objective optimization problem, the blackboard consists of three abstract levels: *Data Level*, *Viable Design Level*, and *Pareto Front Level*. The blackboard abstract levels and data stored on each level are illustrated in Fig. 1.

The Data Level consists of specific design information including the values for the design variables, objective functions, and constraints. The Viable Design Level is an intermediate abstract level that contains design names that meet all of the objective

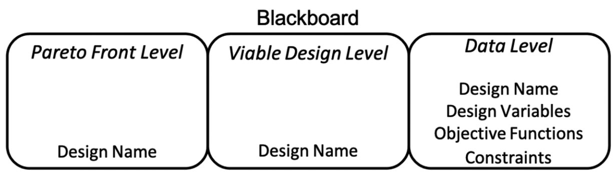


Figure 1: Data stored in blackboard abstract levels.

functions and constraints placed on the problem. The Pareto Front Level consists of the PF and contains optimal designs. The Data Level and Viable Design Level are further divided into two panels: *new* and *old*. In general, a KA writes to the new panel. Once that entry is processed by a different KA, it is moved to the old panel. The panel system prevents KAs from continually searching the same information. The Pareto Front Level does not contain a panel system, as KAs are constantly refining the entries in this level. During the process of solving the optimization problem, the blackboard is stored as a series of nested dictionaries. This storage system allows KAs to quickly search for specific designs or information regarding a design. The BA also periodically writes the blackboard to an archive in the form of an HDF5 file (The HDF Group, 2000–2020). The archived blackboard allows the user to store the PF and associated design data, restart a problem, utilize the data for larger optimization problems, or analyse the results.

2.2.2 *Knowledge agents*

KAs are the problem solvers in the MABS, and advance the problem by discovering and analysing the PF. Two types of KAs are present in the MABS: Search (KA-S) and Blackboard Reader (KA-BR). KA-S agents analyse the design space and determine objective functions based on a set of design variables. To search the design space, six types of KA-S agents are available, where their name, search type, and methodology are briefly described in Table 1. A detailed description for each KA-S agent type can be found in Section 2.6. In the MABS, the search process can involve either a single or multiple KA-S agent types, depending on the problem description and available computational resources. KA-S agents act as a black-box, allowing the user to couple in a surrogate model or physics-based code to determine the values of objective functions or constraints based on designs selected by the KA-S. We note that some agent's actions can be parallelized; this is further described in Section 2.6.

When designs are present on the blackboard, the KA-BR agents move designs among the three abstract levels to promote viable and optimal core designs. Four KA-BR variants have been developed for the MABS and can be seen in Table 2, along with their methods for promoting designs. The KA-BR agents are the driving force in selecting designs on the PF. A detailed description of each KA-BR agent type can be found in Section 2.6.

2.2.3 *Controller agents*

The CA initializes the problem and drives problem progression. This is accomplished by setting up the problem, BA, and all KAs required. The CA then begins the optimization process by informing the BA when to send messages to the KAs. The CA also allows for multiple optimization problems (i.e. a multitiered optimization problem) to be solved simultaneously using multiple BAs. A unique CA is required to initialize a primary problem and any number of subproblems. Each problem is directed by a BA and can be solved in serial or simultaneously. Figure 2 illustrates an example with three subproblems and a primary problem. The three BAs solve subproblems and pass information to the primary problem. The BA-0 solves the overarching problem using data gained from the multiple subproblems. The only limitation is that there must exist a way to pass consistent design variables, objective functions, and constraints to the primary problem.

Table 1: KA-S agent, search type, and action.

KA name	Search type	Action
Stochastic	Global	Uses a uniform distribution to randomly select each design variable from a range or set of available options.
Latin Hypercube[a]	Global	Divides the design space into a grid and samples combinations of design variables to span the entire design space.
Genetic Algorithm[a]	Local	Performs cross-over and mutation techniques on a set of designs.
Neighborhood Search[a]	Local	Perturbs each design variable, or a set of design variables, for a design.
Hill-Climb[a]	Local	Applies incremental changes to a design. If the design is better, another incremental change is applied to this new design. This process continues until no further improvements are possible.
Pymoo[a]	Local	Allows the use of the NSGA-II algorithm from the `pymoo` module (Blank & Deb, 2020).

[a]Agent process can be parallelized.

Table 2: KA-BR agent action.

KA name	Action
Data reader	Reads the Data Level and promotes designs within objective function and constraint ranges to the Viable Design Level.
Promoter	Reads the Viable Design Level, compares designs with designs in the Pareto Front Level and promotes designs that are optimal to the Pareto Front Level.
PF Maintainer	Reads the Pareto Front Level and prunes the PF based on design domination, HVI, DCI grid environment, or a hybrid DCI-HVI.
Inter-BB	In multiblackboard problem, reads the level of one blackboard and writes this information to another blackboard.

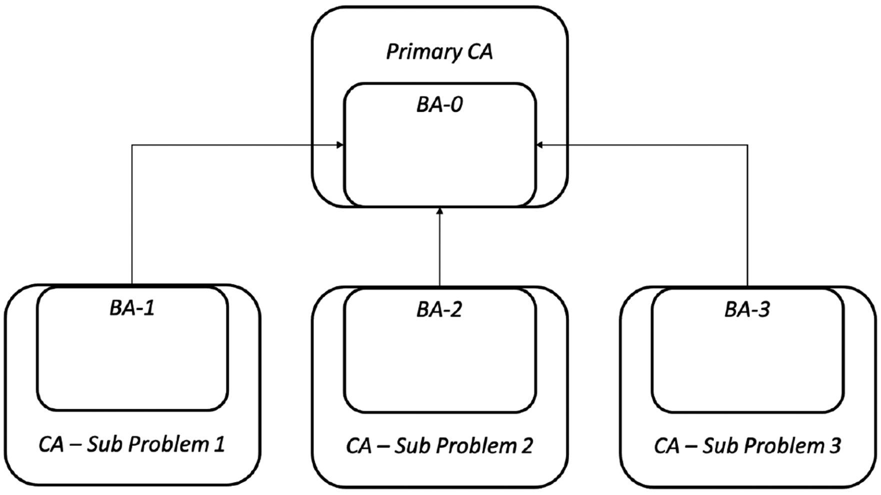


Figure 2: An example of a multitiered MABS problem.

Table 3: BA communication channels.

Name	Type	Description
Trigger	*Publish-Subscribe*	BA asks KAs for TV
Trigger	*Push-Pull*	KA informs BA of its TV
Executor	*Push-Pull*	BA tells KA to perform their action
Writer	*Request-Reply*	KA asks BA permission to write
Action	*Push-Pull*	KA tells BA its action is complete
Shutdown	*Push-Pull*	BA tells agents to shut down

2.3 MABS framework

The MABS framework utilizes a message passing system for communication between the BA and KAs. KAs can only communicate with the BA and can influence the actions of other KAs by reading and writing to the blackboard; no other inter-KA communication is allowed. KAs are connected to the BA via five communication channels, found in Table 3. *Publish-subscribe* allows the BA to simultaneously send the same message to all agents who are subscribed. *Push-pull* allows for one-way communication, where an agent can send a message to another agent. *Request-reply* allows for two-way communication, where an agent can send a message and waits for a reply from the other agent. While the agent who sent the message is waiting for a reply, it can still perform other tasks.

Messages are passed via unique channels to elicit a defined response, denoted as a handler, from each KA. Figure 3 provides an overview for the communication channels presented in Table 3. We follow Fig. 3 through the general process in agent selection. The BA initially sends a *trigger message*, where each KA has a *trigger handler* that is used to determine its TV and send the TV back to the BA. The BA uses this information to select the KA that can most contribute to the problem by sending an *executor message* to the KA with the highest TV (KA-S in Fig. 3). The KA-S agent then begins its action. While the KA-S agent is performing its action, if it is able to add information to the blackboard, it sends a *writer message* to the BA requesting permission to write. If the BA is currently not being written to, the BA's *writer handler* responds to the KA granting it permission to write. Once the KA-S agent has completed its action, it sends an *action message* to the BA to inform it that it is finished.

During the time the KA-S agent is performing its action, the BA is able to send out another trigger message to determine whether the KA-BR agent can perform an action. If the KA-BR agent is already preforming an action or is not able to contribute, the BA then waits for a designated amount of time before sending another trigger message. This process continues until a termination criterion is achieved; the BA then sends a *shutdown message* (not shown in Fig. 3) to each agent that removes them from the multi-agent environment.

We conclude the discussion of the MABS framework with a brief overview of how each agent interacts with the blackboard, as seen in Fig. 4. Dashed lines indicate that the KA can read the level, and solid lines indicate that a KA can write to the level. We see that due to the inter-connectivity of the KAs, each KA relies on the other KAs to populate an abstract level. Once a KA is able to contribute, there is likely another KA that can use this information to either place the design on a different abstract level or perform a perturbation around an already optimal design.

2.4 Initialization

To initialize a problem, the CA is passed a problem statement, a BA, and a list of KAs. The problem statement sets the design variables, objective functions, and constraints that define the problem. Along with this, the problem statement is how a user ties in their surrogate model or high-fidelity simulation. The first agent generated is the BA, where the abstract levels are initialized and a termination criterion is set. The CA passes the KA names and types to the BA, where KAs are initialized as separate system processes. The BA ensures that all channels of communication are available, and any additional attributes are passed

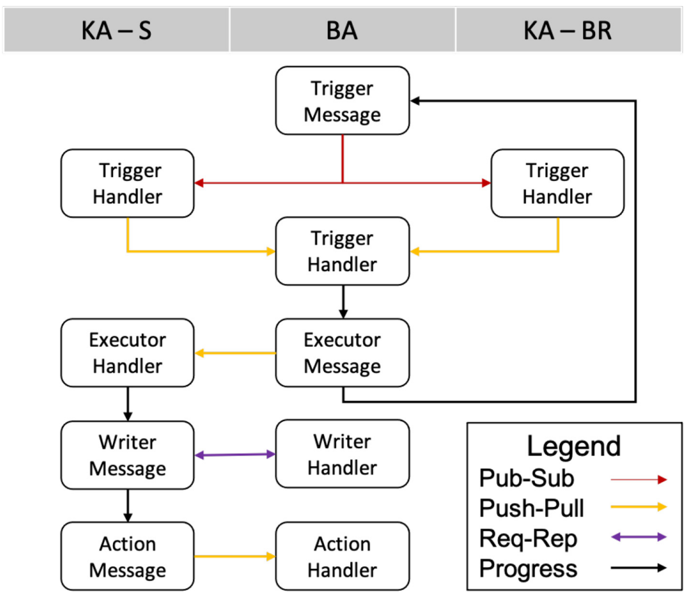


Figure 3: Communication channels of KAs with the BA.

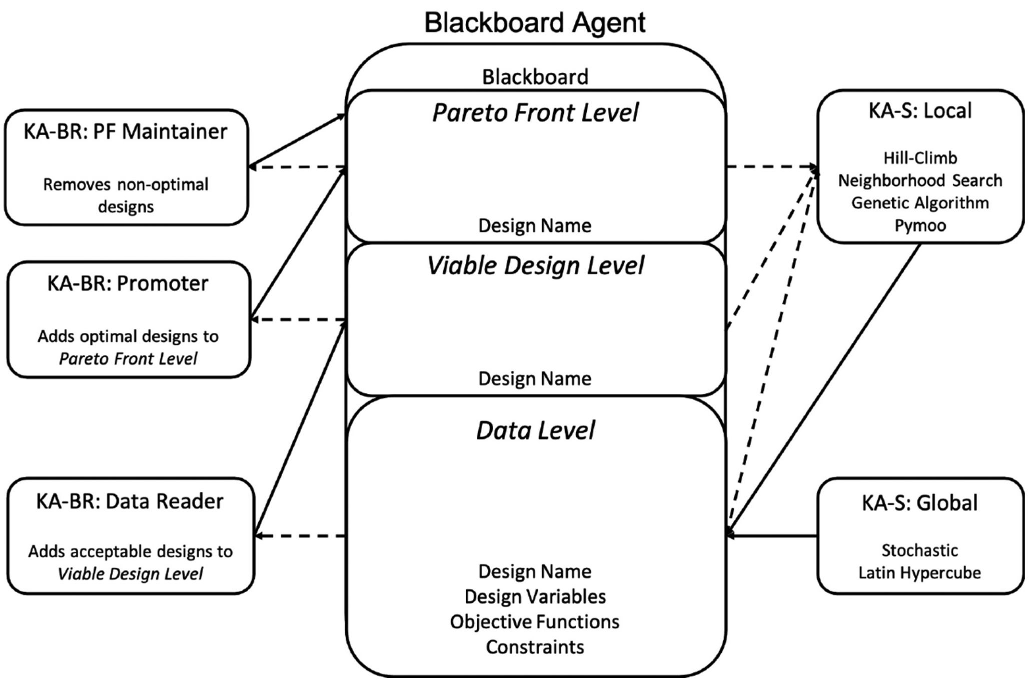


Figure 4: Interactions of KAs with the BA's blackboard.

to the KAs. Once the BA and all KAs are initialized, the problem can be executed in either multi-agent or single-agent mode. This paper describes the multi-agent problem progression; a detailed description of the single-agent approach can be found in Stewart *et al*. (2021a).

2.5 Problem progression

Once the problem has been initialized, the CA begins the optimization process. This process is described by Algorithm 1, where some termination criterion dictates when the problem is complete. The BA starts by publishing a trigger event that is sent to all of the KAs. The BA then waits until a KA sends back a TV greater than 0, indicating that the KA can contribute to the problem. If multiple KAs have responded, the BA selects the KA with the highest TV; if multiple KAs are tied, one of these KAs is randomly selected. The BA then sends the selected agent an executor message that instructs the KA to perform its action. While the KA is performing its actions, the BA repeats this process until there are either no KAs responding to the trigger event or no KAs that can contribute. During this time, the BA can perform other tasks such as write the blackboard in memory to an archive or run diagnostics on the KAs to determine whether any KA has failed and needs to be restarted. For example, if a KA submits a physics model to an HPC and the job fails, this would likely cause the agent to fail.

Algorithm 1: CA loop for BA in MABS

```
1: Result: Pareto Front, HDF5 File
2: while Termination Criteria not met do
3:   Publish trigger
4:   Receive TVs
5:   if No TVs then
6:     Archive blackboard
7:     Run agent diagnostics
8:   end if
9:   Determine KA to select
10:    Send KA executor
11:    if Check Progress then
12:      Check complete
13:      Archive blackboard
14:      Run agent diagnostics
15:    end if
16: end while
17: Shut KAs down
```

After a KA has performed its action, it requests permission from the BA to write this information to an abstract level. KA-S agents write information to the Data Level. When a viable design has been obtained, the KA-BR agents promote these designs to the Viable Design Level and Pareto Front Level. If designs are present on the Pareto Front Level, local KA-S agents use these designs to search the area for additional optimal designs.

The BA will continually send trigger events to request whether KAs can perform their tasks. During this process, the BA will occasionally check the progress of the problem to determine whether the termination criterion has been met. Upon reaching the termination criterion, the BA sends a shutdown message to each KA; this shuts down the agent to prevent it from performing any additional tasks. The blackboard (or corresponding HDF5 archive file) can then be used to examine the PF.

Generational algorithms (such as GAs, ICAs, etc.) require multiple designs to be present in order to explore various regions of the problem. Utilizing a multi-agent approach for solving an optimization problem allows for the exploration of regions of interest in real time as they are discovered by other KAs. For example, if a stochastic KA-S agent finds a viable design, and this design is promoted to the Pareto front level, all of the local KA-S agents can use this information to further explore this area. The stochastic KA-S agent can begin searching other areas of the design space. Utilizing multiple KA-S agents can be applied in two synergistic ways. If duplicate KA-S agents are used, multiple optimal points can be examined simultaneously, leading to an increase in parallel performance. Duplicate KA-S agents can also be used with tuned hyperparameters to help expedite the search process by providing slight variations to their algorithmic approach. If multiple types of KA-S agents are used, the advantages of each search algorithm can be leveraged to thoroughly explore areas of interest. Combining multiple and duplicate KA-S agents gives the user the ability to create a highly parallelizable optimization algorithm that can rapidly search the design space for optimal designs.

2.6 Built-in versatility

The MABS architecture was built to provide a diverse set of algorithms to solve a variety of optimization problems. This section describes many of the built-in algorithms and the methods/hyperparameters that can be enabled. The MABS can handle any number of design variables, objective functions, and constraints for a multi-objective optimization problem. Due to this, MABS is applicable to most engineering-type optimization problems. Both continuous and discrete design variables are allowed, and continuous, discrete, or array-like types are allowed for objective functions and constraints. Array-like objective functions and constraints require the user to specify whether they are looking for an average, maximum, minimum, absolute maximum, or absolute minimum value in the array. The absolute maximum/minimum takes the absolute value of each element in the list and then finds the maximum/minimum. Reducing an array-like objective to a single value allows it to be used by the various KA-S/KA-BR agents for determining optimality. Holding an array of values allows users to store additional information that may be pertinent to a design problem, but not necessary to the optimization process. The ability to store additional information for an objective function or constraint provides the user with a database after the optimization process that contains far more relevant physics information than is typical in most optimization algorithms.

Objective functions (f) can be minimized, maximized, or optimized to a specific value within a bounding region, as seen in equation (1). Users set upper and lower bounds on each objective function to define the objective space. For a minimization problem, we seek to find the smallest value within our bounds. For a maximization problem, we transform the bounds by taking the negative of the objective function and minimize this value. If an objective function is optimized to a specific value, we minimize the absolute difference between the target value (f^*) and the objective function value. This results in the ability to mix and match how objective functions are defined for real-world problems.

$$\begin{aligned} \text{Minimize:}\ & \min \quad f \\ \text{Maximize:}\ & \min \quad -f \\ \text{Equal to:}\ & \min \quad |f^* - f| \end{aligned} \tag{1}$$

The ability of agents to perform tasks independently provides a framework that can be highly parallelizable. For a typical problem, three KA-BR agents (Data Reader, Promoter, and PF Maintainer) are required to sort designs and maintain the PF. The KA-S agents require the most computational resources, especially if they are running high-fidelity physics codes to determine the objective functions and constraints. For HPCs, the number of KA-S agents is theoretically limited by the number of processors required to determine the objective functions and constraints. Typically, a combination of local and global KA-S agents helps promote a balance of exploration and exploitation. Hyperparameters for each of the local KA-S can be tuned to provide a greater diversity in the way the PF is determined. The design variable or objective function ranges can also be adjusted for each agent; this would force these agents to search different regions of the design space. Manipulating the design variables of objective function ranges allows for the injection of expert knowledge into the optimization problem.

The multi-agent approach allows users to write new optimization algorithms to help discover the PF, while simultaneously allowing for agents to be based on other optimization modules. This encourages collaboration between various optimization algorithms from multiple sources, similar in manner to the "experts" from the initial blackboard analogy. For example, pyDOE is used in the Latin Hypercube agent to generate the

Latin hypercube sampling pattern (Pydoe, 2013). The pymoo KA-S agent can utilize the Non-Sorted Genetic Algorithm-II [NSGA-II – originally described in Deb *et al.* (2002)] from the pymoo module, where each design that is evaluated in the NSGA-II algorithm is written on the blackboard (Blank & Deb, 2020). With the ability to easily incorporate innovative optimization modules from other sources, MABS can utilize the strengths of these algorithms while accounting for weaknesses with other algorithms.

MABS can also handle solving multitiered optimization problems, where multiple blackboards are present simultaneously. This can help solve large problems that can naturally be broken into subproblems. Multitiered problems can be solved in serial or in tandem, depending on the problem specifications. Solving a multitiered problem in tandem allows a primary BA to solve the overarching problem, while multiple sub-BAs are used to solve each subproblem. A separate CA is assigned to each BA to control the loop found in Algorithm 1. Once data are presented on the PF of a subblackboard, these data are translated into a the correct format for the primary blackboard and are written to the Data Level. The primary BA uses its own set of KAs to examine these data and optimize the overarching problem. During this time, the sub-BA(s) continues to find optimal designs and promote these designs to the primary blackboard.

2.6.1 *Search agents*

Most KA-S agents have various hyperparameters that can be tuned to examine the design space more effectively. We will begin by describing in greater detail each agent's methodology for searching the design space, followed by the various hyperparameters associated with each KA-S. Hyperparameter names and default values are given by (name=<value>). All KA-S agents can be assigned varying ranges for the design variables, objective functions, and constraints. Along with this, the base TVs can also be updated to increase or decrease the probability that they will be selected.

The local search methods select designs present in one of the three abstract levels; typically, we select designs from the Pareto Front Level, and perform a local search to further explore the PF. The KA-S agents can either select a design on the PF by random, by utilizing a fitness function, or select a design that is optimal in one or more objective functions. The fitness function for each objective function is scaled to a value between 0 and 1 and the resulting values are summed; a larger fitness function indicates a better design. When a single objective is selected, the design with the most optimal value of that objective is selected. If two or more objectives are desired, a fitness function based on the desired objectives is created and the design with the largest fitness function is selected.

Four KA-S agents (highlighted in Table 1) are able to parallelize their search method. This allows for multiple simulations to be executed simultaneously using a series of subagents. The subagents are given a set of design variables and evaluate the objective functions and constraints based on the problem. Once each of the subagents has completed its task, the subagent is removed and the KA writes this information to the blackboard. The number of subagents created is dependent on the method and the number of designs to analyse, with the exception of the Latin Hypercube KA-S.

Stochastic KA-S agents search the design space by randomly selecting a value for each design variable within the bounding limits. Currently, only a uniform distribution is allowed for random sampling. There are no hyperparameters associated with Stochastic KA-S agents.

Latin Hypercube KA-S agents generate a set of designs based on a hypergrid, which adequately samples the entirety of the design space (McKay *et al.*, 1979). Hyperparameters for the Latin Hypercube agent include the number of samples to take within the design space (samples=50), and if running in parallel, the maximum number of subagents deployed can be set (max_sub_agents=50). The criterion for generating the hypergrid can also be selected as either simple and correlated (lhc_criterion="correlated"). A simple criterion randomizes the points within the hypergrid intervals, while a correlated criterion tries to minimize the maximum correlation coefficient associated with the designs selected (Pydoe, 2013).

Neighborhood Search KA-S agents select a design from one of the abstract levels and apply a small perturbation to each design variable, separately, to generate new designs (Lones, 2014). Hyperparameters include the size of the perturbation (perturbation_size=0.05), the ability to perturb multiple design variables simultaneously (additional_perturbations=0), and the ability to fix the perturbation size or randomly select a perturbation within the designated perturbation (neighborhood_search="fixed"). For a discrete design variable, a value is randomly selected among the options.

Hill-Climb KA-S agents select a design from one of the abstract levels and apply an incremental change (called a step size) to each design variable independently (Lones, 2014). The objective functions are analysed and the KA-S selects a design that is more optimal than the initial design. Two methods for determining the next design are available: simple ascent and steepest-ascent. The simple ascent method analyses each step individually and selects the first step that increases the fitness function. The steepest-ascent method examines all possible steps and selects the step that has the greatest change in fitness function for a change in the design variable. If no step creates a more optimal design, the step size is decreased and the process is repeated. This process stops if a set number of steps has been reached or if the step size is smaller than some convergence criteria. Hyperparameters include the algorithm type (hc_type="simple"), initial step size (step_size=0.1), the rate at which the step size decreases (step_rate=0.1), the total number of steps allowed (step_limit=100), and the convergence criteria (convergence_criteria=0.001).

Genetic Algorithm KA-S agents select designs from one of the abstract levels and applies cross-over and mutation techniques to generate new designs. Single-point, linear, and batch cross-over are available for performing design cross-over (Sorsa *et al.*, 2008; Zameer *et al.*, 2014), while random and nonlinear mutations are available for performing mutation (Sorsa *et al.*, 2008). Hyperparameters include the cross-over type (crossover_type="single point") and mutation type (mutation_type="random"), the cross-over rate (crossover_rate=0.8) and mutation rate (mutation_rate=0.1), population size (pf_size=40), and the offspring produced (offspring_per_generation=20). Three additional hyperparameters ($b = 2$, $k = 5$, and $T = 100$) are available for the nonuniform mutation that drives the size of the mutation.

Pymoo KA-S agents utilize algorithms from the pymoo module (Blank & Deb, 2020). The PF from the blackboard can be used as the initial population, and the NSGA-II module undergoes a series of cross-over and mutations to discover the PF. Hyperparameters include the algorithm (pymoo_algorithm_name="NSGA2"), cross-over type (crossover_type="real_sbx"), mutation type (mutation_type="real_pm"), population size (pop_size=25),

Table 4: Parallelizability for KA-S agents.

KA name	Parallelizability
Stochastic	N
Latin Hypercube	$N *$ `max_sub_agents`
Genetic Algorithm	$N *$ `offspring_per_generation`
Neighborhood Search	$N*2*DVs$
Hill-Climb (Simple)	N
Hill-Climb (Steepest-Ascent)	$N*2*DVs$
Pymoo	$N *$ `n_offspring`

number of offspring (n_offspring=10), termination type (termination_type="n_eval"), and termination criteria (termination_criteria=250). The hyperparameters are used to set up and run a pymoo problem.

Many of the KA-S agents have the ability to be parallelized, meaning that the evaluations of designs can be performed concurrently. This, along with the ability to add KA-S agents, allows for a highly parallelizable optimization algorithm. Table 4 shows how each of the KA-S agents can be parallelized: *N* is the number of the particular KA-S agents in the system, and *DVs* is the number of design variables present in the problem. For some KA-S agent, such as Latin Hypercube, GA, and pymoo, the user can dictate the number of subagents that can be spawned to evaluate multiple designs. Other agents, such as Neighborhood Search and Hill-Climb (Steepest-Ascent), are dependent on the number of design variables: the multiple of 2 is based on a continuous variable where an increase and decrease to the design variable is examined. Utilizing multiple KA-S agents and parallelizing each agent can create a large multi-agent system, where the user must take care to ensure the operating system can handle the number of processes required for these problems.

2.6.2 Blackboard reader agents

Of the four KA-BR agents, only the PF maintainer agent has unique hyperparameters. The PF Maintainer agent is tasked with maintaining the PF; this includes examining and removing designs on the PF. To prune the PF, four options are available: nondominated sorting, HV, DCI, and DCI-HV. Nondominated sorting removes any dominated designs by comparing each design against all other designs on the PF. The nondominated sorting method does not limit the number of designs on the PF. HV examines the contribution each design makes to the HV and removes the designs that contribute the least to the HV until the total number of designs on the PF reaches a user-defined value. The Diversity Comparison Indicator (DCI) places the designs in a hypergrid, which is used to assess the diversity of the designs on the PF. If multiple designs are present in the same hyperbox, the design with the highest fitness function is kept and the other designs are discarded. The DCI method does not explicitly limit the number of designs on the PF. DCI-HV utilizes the DCI method to remove an initial number of designs that are crowded together, and then relies on the HV method to prune the number of designs to a user-defined value. This method was implemented to reduce the time required in calculating the HV contribution for each design by preemptively pruning designs that are close together.

2.6.3 Blackboard

There are three methods for terminating the MABS: a set number of function evaluations, a set number of TVs, or the hypervolume (HV) convergence. Terminating the problem based on function evaluations is determined by the number of designs found on the Data level. Terminating the problem based on the number of TVs simply examines how many agents have been tasked with performing an action. Setting a fixed number of function evaluations or number of TVs can also be used in conjunction with the HV convergence to bound the problem. The HV examines the change in the HV over a TV interval, with a default convergence criterion of 10^{-5}. The TV interval is used to average the HV and compare it with the previously averaged HV to determine the convergence rate; if this value is less than the convergence criteria, the problem is determined converged. While the HV can be used to determine if the algorithm has converged on the PF, care must be taken to ensure that the problem does not prematurely converge. Calculating the HV can be extremely time consuming for problems with large numbers of design variables, especially if the PF is large. Allowing KAs to perform tasks during the time required for the BA to calculate the HV mitigates some of this time requirement, and the potential to create subagents to calculate the HV as a function of function evaluations has the promise to increase the viability of using the HV as a convergence metric. While this is currently not implemented, parallelizing agent abilities has shown promise for reducing computational time (see Section 3.4).

Terminating the problem can be done in a soft or hard manner. In the soft method, the BA no longer publishes a request for TVs and instead waits for any remaining agents to finish their tasks. The hard method forces each KA agent to shut down immediately upon completing its most recent design evaluation. While shutting down via the hard method takes considerably less time than the soft method, the time for shutdown will still be on the order of time required for performing a function evaluation.

3 Results

3.1 Benchmark problems

To assess the advantages and disadvantages of using the MABS as an optimization algorithm, it is advantageous to examine its ability to solve a series of benchmark optimization design problems. These benchmarks have known PFs and can be expected to represent the types of problems most experienced in science and engineering (Tanabe & Ishibuchi, 2020). Some benchmarks, such as the ZDT or DTLZ suites, provide varying numbers of design variables, objective functions, and PF types (Zitzler, 1999; Deb *et al.*, 2005). While these benchmarks are extremely useful for determining how optimization algorithms will handle multiple false PFs or PFs with exotic shapes, we have instead decided to focus on determining how the MABS will be able to solve real-world engineering benchmarks. The engineering benchmark suite provides problems using two to nine objectives, two to seven design variables, and both continuous and discrete design variables. This provides a diverse set of problems that are envisioned to be representative of problems that a typical scientist, researcher, or engineer will encounter in their work. As such, it is pertinent to show that the MABS will be able to adequately discover the PF for these types of problems. Table 5 gives a brief description for each benchmark problem; mixed indicates a problem with both continuous and discrete variables.

Given the diversity of optimization algorithms presented in literature, it is prudent to compare MABS against some well-known modern algorithms from the pymoo module (Blank & Deb, 2020): Non-Sorted Genetic Algorithm-III [NSGA–III – originally described in Deb and Jain (2014b) and Jain and Deb (2014)],

Table 5: Benchmark properties for engineering benchmarks.

Name	Original name	Objs	DVs	Variables
RE2-4-1	Four bar truss design	2	4	Continuous
RE2-3-2	Reinforced concrete beam design	2	3	Mixed
RE2-4-3	Pressure vessel design	2	4	Mixed
RE2-2-4	Hatch cover design	2	2	Continuous
RE2-3-5	Coil compression spring design	2	3	Mixed
RE3-3-1	Two bar truss design	3	3	Continuous
RE3-4-2	Welded beam design	3	4	Continuous
RE3-4-3	Disk break design	3	4	Continuous
RE3-5-4	Vehicle crash worthiness design	3	5	Continuous
RE3-7-5	Speed reducer design	3	7	Mixed
RE3-4-6	Gear train design	3	4	Discrete
RE3-4-7	Rocket injector design	3	4	Continuous
RE4-7-1	Car side impact design	4	7	Continuous
RE4-6-2	Car side impact design	4	6	Continuous
RE6-3-1	Water resource planning	6	3	Continuous
RE9-7-1	Car cab design	9	7	Continuous

Adaptive Geometry Estimation-based Multi Objective Evolutionary Algorithm [AGE–MOEA – originally described in Panichella (2019)], and Two Archive Evolutionary Algorithm [C–TAEA – originally described in Li *et al.* (2019)].

3.2 Performance metrics

Three performance metrics are used to quantitatively assess the performance of each algorithm. The PF is known for each benchmark case that allows for the examination of the generation distance (GD), inverted generational distance, and the HV.

3.2.1 *Generation distance*

The GD examines the distance between the nondominated PF (PF) and the optimal PF (PF*; Veldhuizen & Lamont, 1998), and is defined by

$$\mathrm{GD}(\mathrm{PF}, \mathrm{PF}^*) = \frac{\left[\sum_{v\in \mathrm{PF}} d(v, \mathrm{PF}^*)\right]^{\frac{1}{2}}}{|\mathrm{PF}|}, \tag{2}$$

where $d(v, \mathrm{PF}^*)$ is the Euclidean distance between the point v and the nearest point in PF*, and |PF| is the total number of solutions on the PF. The GD indicates how close the solutions on the PF are to the Pareto-optimal solutions. A PF that is closer to the optimal PF (i.e. is more converged) has a smaller GD.

3.2.2 *Inverted generation distance*

The inverted generation distance (IGD) is similar to the GD in that it also measures the distance between the nondominated PF and the optimal PF (PF*; Coello *et al.*, 2002). It is defined by

$$\mathrm{IGD}(\mathrm{PF}, \mathrm{PF}^*) = \frac{\left[\sum_{v\in PF^*} d(v, \mathrm{PF})\right]^{\frac{1}{2}}}{|\mathrm{PF}^*|}, \tag{3}$$

where $d(v, \mathrm{PF}^*)$ is the minimum Euclidean distance between the point v and the nearest point in the PF, and |PF* | is the total number of solutions on the PF*. The IGD uses the solution in the PF* to determine the distance, rather than the solution on the nondominated PF. If a large number of solutions are present in PF*, the IGD can also be used as a measure of diversity of the nondominated PF. A PF that is closer to PF* (i.e. is more converged and diverse) has a smaller value for the IGD.

3.2.3 *Hypervolume*

The HV gives the volume of the objective space that is dominated by the set of solutions on the PF. This is given by

$$\mathrm{HV}(\mathrm{PF}, R) = \bigcup_i \mathrm{vol}_i \mid v_i \in \mathrm{PF}, \tag{4}$$

where v_i is a solution on the PF, and vol_i is the volume between v_i and the reference point R. The HV can be used to evaluate an algorithm's ability to converge and maintain a diverse PF.

3.3 Benchmark results

To allow for a fair comparison, each algorithm was allowed to perform 10 000 function evaluations. Each algorithm was executed 20 times to provide an average for GD, IGD, and HV. The NSGA-III, AGE-MOEA, and C-TAEA algorithms were all evaluated using the pymoo module. Most of these algorithms require a reference direction to initialize the algorithm. While a detailed discussion for reference directions is not suited for this paper, this places limits on the population size. As such, a population of at least 100 was required, but a specific number was not required. For each algorithm, the default cross-over (0.9) and mutation (0.2) probabilities were used. Simulated binary cross-over was used as the cross-over technique, and random mutation was used to mutate individuals; all other parameters for each algorithm were set according to any predefined value. No attempt was made to optimize the parameters for any algorithm.

The MABS was set up using the KA-S agents presented in Table 6. Hyperparameter values are presented as an array, where each index of the array represents the value for a unique agent. Along with the 10 KA-S agents, 3 KA-BR agents were used: Data Reader, Promoter, and PF Maintainer. The PF Maintainer used nondominated sorting to maintain the PF. No attempt was made to optimize the number of agents or agent hyperparameters for the MABS.

Results are presented for each benchmark problem in Tables 7–9, where the mean and standard deviation of each performance metric are displayed. The last row in each table tallies the number of times the algorithm has the best value (*w*), the worst value (*l*), or ties the best value (*t*). Tables 7–9 show that MABS performs statistically well when compared with other algorithms. While these attributes are associated with the MABS as a whole, these values are based on the KAs selected in addition to their

Table 6: KA-S agents and hyperparameters for MABS.

Agent type	Num. agents	Hyperparameters
Stochastic	1	None
Latin Hypercube	1	`samples`: [100∗N^a] `criterion`: [correlated]
Neighborhood Search	3	`perturbation_size`: [0.025, 0.01, 0.01] `neighborhood_search`: [random, fixed, random] `additional_perturbation`: [0, 0, $N^a - 1$]
Genetic Algorithm	2	`cross_over`: [linear, random] `mutation_type`: [nonuniform, random] `offspring_per_generation`: [20, 20] `mutation_rate`: [0.1, 0.1]
Hill-Climb	2	`hc_type`: [simple, steepest-ascent] `step_size`: [0.01, 0.01] `step_limit`: [10, 50] `convergence_criteria`: [0.001, 0.001]
Pymoo	1	`pop_size`: [50] `n_offspring`: [50] `termination_criteria`: [2500] `termination_type`: [n_eval] `crossover_type`: [real_sbx] `mutation_type`: [real_pm]

[a]N is the number of design variables.

Table 7: Mean (standard deviation) of GD for algorithm comparison for engineering-based benchmarks.

Problem	NSGA-III	AGE-MOEA	C-TAEA	MABS
RE2-4-1	4.504e-01 (3.729e-02)	4.760e-01 (3.829e-02)	**1.932e-01** (1.097e-01)	4.684e-01 (**2.272e-02**)
RE2-3-2	5.928e+01 (2.437e+02)	8.523e+04 (3.540e+05)	3.516e-01 (7.987e-02)	**2.735e-01** (**4.148e-02**)
RE2-4-3	3.926e+02 (1.696e+01)	**2.516e+02** (**1.645e+01**)	3.449e+02 (9.830e+01)	3.899e+02 (3.790e+01)
RE2-2-4	1.524e-01 (2.061e-02)	1.306e-01 (1.303e-02)	**1.141e-01** (**4.029e-03**)	1.218e-01 (2.121e-02)
RE2-3-5	6.064e-04 (3.732e-04)	**2.645e-04** (**1.108e-05**)	4.803e-02 (8.493e-02)	3.204e-04 (8.868e-05)
RE3-3-1	**3.605e+03** (**1.143e+03**)	3.689e+04 (1.014e+04)	9.628e+03 (6.835e+03)	2.931e+04 (8.715e+03)
RE3-4-2	7.410e+04 (1.199e+04)	4.244e+04 (3.947e+03)	**7.148e+03** (**2.486e+03**)	1.773e+04 (5.178e+03)
RE3-4-3	**1.690e-01** (2.254e-02)	3.400e+00 (2.822e+00)	2.268e-01 (**8.983e-03**)	1.869e-01 (2.682e-02)
RE3-5-4	7.470e-02 (1.469e-02)	7.486e-02 (6.813e-03)	6.591e-02 (8.826e-03)	**6.309e-02** (**4.581e-03**)
RE3-7-5	**7.264e+00** (2.211e+00)	9.566e+00 (8.266e-01)	1.092e+01 (**7.728e-01**)	9.901e+00 (1.629e+00)
RE3-4-6	5.609e-02 (8.224e-02)	5.571e-02 (7.964e-02)	6.522e-02 (**4.087e-02**)	**5.239e-02** (1.530e-01)
RE3-4-7	2.285e-02 (3.038e-03)	2.611e-02 (2.859e-03)	2.003e-02 (1.643e-03)	**1.482e-02** (**1.579e-03**)[a]
RE4-7-1	2.882e-01 (5.437e-02)	3.219e-01 (**3.783e-02**)	3.639e-01 (5.766e-02)	**2.449e-01** (2.754e-02)
RE4-6-2	1.419e+02 (9.409e+01)	8.815e+01 (1.739e+01)	1.569e+02 (4.578e+01)	**4.790e+01** (**7.756e+00**)
RE6-3-1	**3.280e+04** (**4.197e+03**)	5.172e+04 (3.573e+03)	6.256e+04 (8.302e+03)	4.496e+04 (5.533e+03)
RE9-7-1	**2.438e-01** (3.678e-02)	5.114E-01 (**2.268E-02**)	5.056E-01 (3.590E-02)	4.469e-01 (4.435e-02)
w/l/t	5/4/0	2/6/0	3/6/0	6/0/0

[a]Using a Welch T-test, the MABS statistically outperforms all three algorithms for this metric.

Table 8: Mean (standard deviation) of IGD for algorithm comparison for engineering-based benchmarks.

Problem	NSGA-III	AGE-MOEA	C-TAEA	MABS
RE2-4-1	4.940e+00 (1.608e-01)	5.004e+00 (3.106e-01)	5.060e+00 (6.728e-01)	**6.157e-01** (**7.639e-02**)
RE2-3-2	1.790e+00 (5.195e-01)	1.388e+00 (8.831e-02)	2.167e+00 (1.566e-01)	**4.394e-01** (**5.872e-02**)[a]
RE2-4-3	1.199e+04 (8.789e+02)	5.929e+03 (**2.550e+02**)	4.265e+04 (4.257e+04)	**7.383e+02** (3.172e+02)
RE2-2-4	8.608e+00 (3.691e-01)	1.607e+00 (7.241e-02)	2.454e+01 (**1.068e-01**)	**2.136e-01** (2.291e-01)
RE2-3-5	5.134e+02 (1.320e-04)	**2.122e-03** (**6.649e-05**)	6.258e+03 (6.794e-02)	7.418e+02 (4.885e-04)
RE3-3-1	3.902e+06 (1.781e+06)	6.282e+05 (1.290e+05)	3.330e+06 (1.963e+06)	**1.536e+05** (**8.609e+04**)
RE3-4-2	5.736e+07 (2.383e+07)	5.381e+06 (**4.515e+05**)	3.206e+07 (2.390e+07)	**1.886e+06** (2.830e+06)
RE3-4-3	**3.520e+07** (2.536e+00)	**3.520e+07** (3.094e+02)	**3.520e+07** (**1.684e-01**)	**3.520e+07** (2.442e+01)
RE3-5-4	6.980e-01 (5.775e-02)	**3.081e-01** (**1.862e-02**)	5.725e-01 (7.699e-02)	3.349e-01 (2.489e-01)
RE3-7-5	2.351e+02 (1.912e+01)	3.596e+01 (**2.919e+00**)	1.071e+02 (4.774e+01)	**2.075e+01** (1.468e+01)
RE3-4-6	1.331e+00 (**2.176e-01**)	**5.804e-01** (2.207e-01)	7.351e-01 (2.191e-01)	8.694e-01 (2.220e-01)
RE3-4-7	7.182e-02 (1.881e-03)	5.344e-02 (**1.049e-03**)	5.839e-02 (1.281e-03)	**2.555e-02** (1.993e-03)
RE4-7-1	1.148e+00 (5.229e-02)	6.278e-01 (2.338e-02)	6.996e-01 (**1.966e-02**)	**2.965e-01** (3.353e-02)[a]
RE4-6-2	4.363e+02 (3.271e+01)	1.428e+02 (**6.139e+00**)	2.499e+02 (3.866e+01)	**9.118e+01** (2.618e+01)[a]
RE6-3-1	3.839e+05 (9.968e+04)	**1.131e+05** (**3.843e+03**)	1.530e+05 (7.887e+03)	3.432e+05 (8.869e+04)[a]
RE9-7-1	2.845e+01 (7.985e+00)	**4.531E+00** (**9.425E-01**)	7.862E+00 (2.250E+00)	6.498e+00 (1.870e+00)
w/l/t	0/10/1	5/0/1	0/5/1	10/0/1

[a]Using a Welch T-test, the MABS statistically outperforms all three algorithms for this metric.

Table 9: Mean (standard deviation) of HV for algorithm comparison for engineering-based benchmarks.

Problem	NSGA-III	AGE-MOEA	C-TAEA	MABS
RE2-4-1	6.723e-01 (2.898e-04)	6.708e-01 (3.713e-04)	4.936e-01 (1.314e-01)	**6.778e-01** (**1.321e-04**)
RE2-3-2	5.458e-01 (2.254e-03)	5.455e-01 (6.107e-04)	5.461e-01 (**2.029e-04**)	**5.507e-01** (4.171e-04)[a]
RE2-4-3	9.509e-01 (2.144e-04)	9.526e-01 (**5.059e-05**)	9.480e-01 (4.687e-03)	**9.535e-01** (8.444e-05)
RE2-2-4	9.574e-01 (1.157e-04)	9.598e-01 (1.015e-04)	9.470e-01 (8.400e-05)	**9.612e-01** (**5.899e-05**)[a]
RE2-3-5	8.513e-01 (5.048e-10)	**8.712e-01** (**4.599e-11**)	4.871e-01 (5.284e-05)	**8.712e-01** (1.176e-10)
RE3-3-1	**1.000e+00** (3.326e-06)	**1.000e+00** (**9.964e-08**)	**1.000e+00** (7.716e-05)	**1.000e+00** (5.521e-08)
RE3-4-2	9.986e-01 (2.699e-04)	**9.996e-01** (**3.759e-06**)	9.955e-01 (2.693e-03)	**9.996e-01** (2.404e-05)
RE3-4-3	9.754e-01 (4.014e-03)	9.806e-01 (1.383e-03)	9.699e-01 (8.903e-03)	**9.836e-01** (**1.067e-03**)
RE3-5-4	7.170e-01 (**4.003e-03**)	7.399e-01 (8.560e-04)	6.967e-01 (1.528e-02)	**7.432e-01** (9.011e-03)
RE3-7-5	9.631e-01 (5.216e-03)	9.744e-01 (1.215e-03)	9.685e-01 (1.948e-03)	**9.752e-01** (**9.428e-04**)
RE3-4-6	6.449e-01 (4.811e-03)	**6.618e-01** (**1.224e-04**)	6.551e-01 (4.170e-03)	6.613e-01 (5.849e-04)
RE3-4-7	5.756e-01 (2.722e-03)	5.344e-02 (**1.049e-03**)	5.839e-02 (1.281e-03)	**6.151e-01** (2.523e-03)[a]
RE4-7-1	4.450e-01 (**2.982e-03**)	4.733e-01 (3.707e-03)	4.149e-01 (7.212e-03)	**5.060e-01** (8.656e-03)[a]
RE4-6-2	4.889e-01 (5.868e-03)	5.412e-01 (**1.545e-03**)	4.946e-01 (9.970e-03)	**5.484e-01** (5.308e-03)
RE6-3-1	7.684e-01 (5.093e-03)	**8.040e-01** (**1.635e-03**)	6.804e-01 (5.032e-02)	7.220e-01 (4.473e-03)
RE9-7-1	3.291e-02 (2.679e-03)	**4.276E-02** (2.109E-03)	1.740E-02 (2.439E-03)	4.007E-02 (**1.835E-03**)
w/l/t	0/3/1	3/2/3	0/10/1	10/0/3

[a]Using a Welch T-test, the MABS statistically outperforms all three algorithms for this metric.

Table 10: Number and type of KA-S agents for parallelizability study.

Case	Total agents	Agent breakdown
1	6	S: 1, LHC: 1, NS: 1, HC: 1, GA: 1, pymoo: 1
2	12	S: 2, LHC: 2, NS: 2, HC: 2, GA: 2, pymoo: 2
3	24	S: 4, LHC: 4, NS: 4, HC: 4, GA: 4, pymoo: 4
4	48	S: 4, LHC: 4, NS: 12, HC: 12, GA: 12, pymoo: 4
5	96	S: 4, LHC: 4, NS: 28, HC: 28, GA: 28, pymoo: 4

hyperparameters. However, given that the number and hyperparameters of KAs were not optimized, it appears that using a combination of KA-S types leads to a thoroughly developed PF. As such, different combinations of KAs and their associated hyperparameters may yield better or worse results.

Overall, we find that the MABS performs very well when compared with the three optimization algorithms. MABS has the largest number of wins, but more importantly, we find that MABS does not score the worst value in any metric over all the benchmarks. This provides confidences that in a real-world problem, MABS will be able to both find and converge to the PF.

Much of the success of MABS is based on the ability to initially sample the design space using the Latin hypercube method, further drive exploration using pymoo and the GA, and thoroughly search optimal areas using the hill climb and neighborhood search algorithms. Using a traditional optimization algorithm, like NSGA-II or the built in GA, allows MABS to continually explore large portions of the PF, which in turn helps drive the diversity of solutions present. MABS capitalizes on this and is able to outperform the compared optimization algorithms by the use of other local search methods (neighborhood search, hill-climb) to effectively search surrounding areas. While mutation in evolutionary algorithms can provide this type of exploration, the rate at which mutations are added is much slower than utilizing other KA-S agents explicitly for this function. In conjunction with this, since the MABS is not constrained to a specific population size, the PF has no limit on the number of solutions present. This helps ensure that optimal solutions remain present on the PF to allow for further exploration.

3.4 Parallelization study

To examine the parallelizability of the MABS, we considered the RE4-7-1 benchmark problem with three cases corresponding to various evaluation times required to calculate the objective functions/constraints. 10 000 function evaluations were performed and the time requirements of the MABS were observed and compared with the NSGA-III algorithm. The three cases correspond to 0, 30, and 300 second evaluation times. A 0 second evaluation time corresponded to either the evaluation of a surrogate model or some type of analytical relationship between the design variables and objective functions/constraints. The 30 and 300 second evaluation times correspond to some type of simulation being used for the calculations. To implement this, we forced each function evaluation to wait for the evaluation time. For the NSGA-III algorithm, we fixed the population size to 56 and 120 members and assume we can solve all members in parallel. This same approach was used for the NSGA-II algorithm within the MABS as well. For the MABS, we examine the use of base parallelization (i.e. adding agents) and subagent parallelization (i.e. adding agents and allowing them to utilize subagents as well) when comparing with the NSGA-III algorithm.

For each time evaluation, five different cases, each using different numbers of agents, are completed with the MABS. The number and types of agents for each case are shown in Table 10. KA-S agents in Table 10 are described as stochastic (S), Latin-hypercube (LHC), neighborhood search (NS), hill-climb (HC), GA, or pymoo's NSGA-II (pymoo). The hyperparameters are not described in detail, but are randomly selected for each agent; hyperparameters between runs are maintained for consistency. For example, the hyperparameters for run 1 are used in run 2, where six new sets of hyperparameters are also initialized for the remaining six agents not previously maintained. The exception to the randomly selected hyperparameters is the pop_size for pymoo. The pop_size starts with a population of 20, and was increased by 10 for each pymoo agent added (for example, run 4 has four pymoo agents with population sizes of 20, 30, 40, and 50). For cases using all four pymoo agents, up to 20% of the solutions will be generated using the NSGA-II algorithm from pymoo. Table 11 shows the KA-S agents and all hyperparameters available to be selected along with any hyperparameter ranges [shown in (...)]

Table 11: Hyperparameters and ranges for KA-S agents.

Agent type	Hyperparameters (Name: Values)
Stochastic	N/A
Latin Hypercube	`lhc_criterion`: [simple, corr]
Hill-Climb	`hc_type`: [simple, steepest-ascent]; `convergence_criteria`: (0.0001, 0.1); `step_size`: (0.01, 0.5); `step_rate`: (0.01, 0.5); `step_limit`: (5, 100)
Neighborhood Search	`neighborhood_search`: [fixed, random]; `perturbation_size`: (0.005, 0.5); `additional_perturbations`: (0, $N^a - 1$)
Genetic Algorithm	`crossover_type`: [linear, single point]; `mutation_type`: [random, nonuniform]; `offspring_per_generation`: (10, 100); `mutation_rate`: (0.05, 0.25), `k`: (1, 10), `T`: (50, 150)
pymoo	`pop_size`: (20, 50); `n_offspring`; (pop_size∗0.5, pop_size); `function_evals`: pop_size ∗ 20)

[a]N is the number of design variables.

or hyperparameter options (shown in [...]). For cases 4 and 5, only the numbers of local KA-S agents are increased (NS, HC, and GA). Both the stochastic and Latin-hypercube agents are meant for initial exploration of the problem, where further increasing the stochastic and Latin-hypercube method would provide diminishing returns for developing the PF. For cases using four Latin hypercube agents, roughly 15% of solutions will be generated by this method, which should provide an adequately sampled design space. To gain adequate statistics, each run for each evaluation time is executed 10 times.

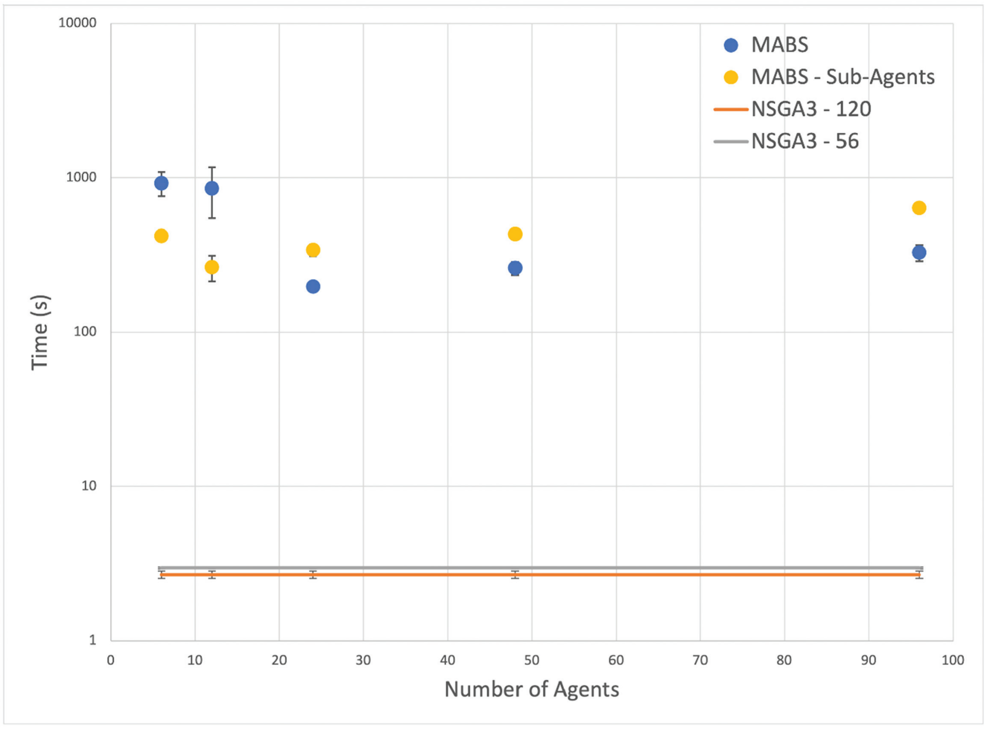
Figures 5–7 show the time required for both the MABS and the NSGA-III algorithm to examine 10 000 function evaluations. The time for the NSGA-III algorithm is constant as it does not rely on the agent-based system. We find two interesting trends in the data shown in Figs 5 and 7: the failure of the MABS to compete with the NSGA-III with a 0 second simulation time, and the time reduction of the MABS when we increase the number of agents present for larger simulation times.

For a 0 second simulation time, the computational overhead associated with executing the MABS outweighs the potential benefit of using additional agents. While there is a minimal decrease in computational time with increasing numbers of agents initially, the trend does not continue for larger numbers of agents. This failure is due to the multi-agent system becoming saturated with agents. The BA sends an executor message to a KA, and by the time the BA is ready to determine the next round of TVs, the initial KA has already completed its action and is ready to perform a new action. This prevents multiple agents from being able to act concurrently, and instead, agents must wait. The HV also suffers when utilizing a 0 second simulation time. For the first two data points (i.e. using six and twelve agents), the HV is relatively close to the NSGA-III algorithms; however, the later data points, representing additional agents present, are subpar. The lower HV score is due to the rate at which solutions are added to the PF, which can lead to a single area being rapidly explored; however, the PF as a whole is not adequately explored. Along with this, because KAs must constantly wait to be selected, it is highly likely that the advantages of using the various hyperparameters for each agent were not able to manifest. This would likely lower the ability to explore the PF in a uniform manner.

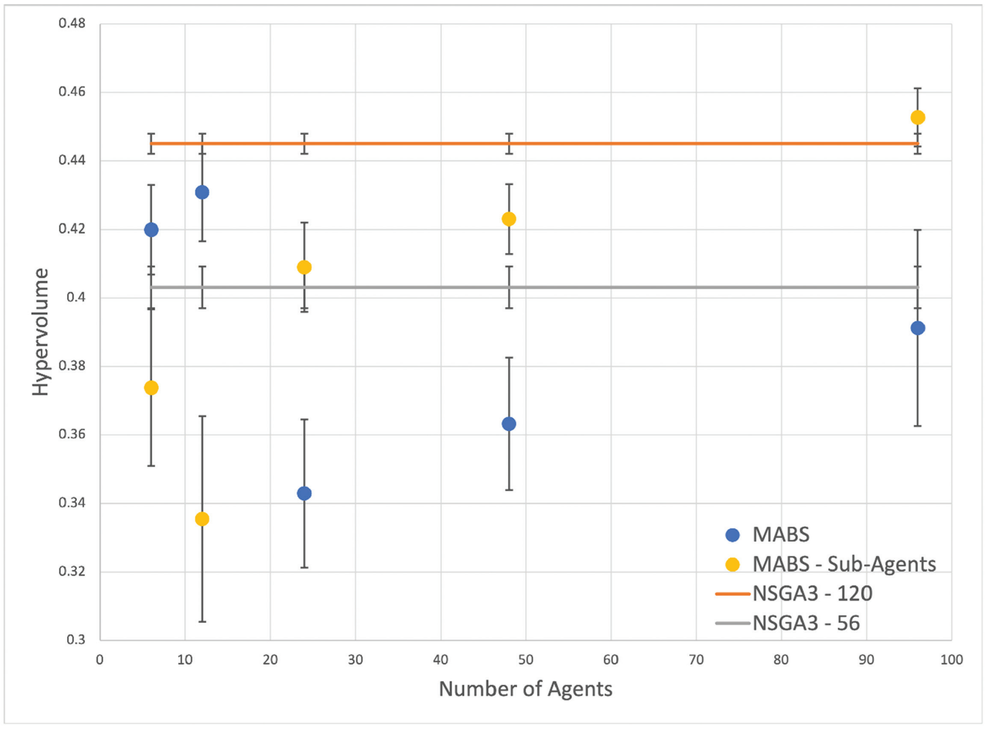
For simulation times of 30 and 300 seconds (Figs 6a and 7a), there is a negative correlation between the computational time and number of agents that roughly follows a power law. This boon in the computation time is due to the number of simulations that can be run concurrently, roughly based on the number of agents (A) for a base parallel and $A_s{*}P_s$ with the use of subagents, where A_s is a KA-S agent of type (s) and P_s is the parallelizability of the agent (see Table 4). It follows that utilizing subagents provides an added benefit to reducing the computational time, as we are able to calculate more simulations in a given time than the NSGA-III algorithm. For the 300 second simulation, both the base parallel and the use of subagents, there is reduction in the computational time by roughly a factor for 2 when doubling the number of agents, indicating a strong ability to parallelize the problem. For the HV, Figs 6b and 7b show that the number of agents only weakly affects the HV score; however, both outperform the NSGA-III algorithm. This is again due to the fact that the simulation time allows multiple agents to each perform their search methods to effectively discover the PF.

If high-fidelity simulations are evaluated rather than computationally cheap surrogate models, the MABS is able to outperform the NSGA-III algorithm. Accounting for the time required to run a simulation allows much of the computational overhead to be performed while agents are performing their actions. To further examine this, a breakdown of the time requirements for the problem is considered. Creating the multi-agent environment is performed by the CA and BA, where each KA must be substantiated: it takes approximately 0.25 second per agent created (2 seconds for 8 agents, 24 seconds for 96 agents, etc.). For the MABS to examine a problem with 10 000 function evaluations, 10 000 designs must be written to an HDF5 file: On average, this process consumes 100–250 seconds, which is currently based on writing to an HDF5 file every 25 TVs. Finally, the multi-agent environment must be shut down: This process takes about 5 seconds plus the amount of time required for a function evaluation (as KA-S agents complete their current function evaluation before shutting down). In all, the overhead associated for using the MABS can range from 110 to 275 seconds plus the time associated with performing a function evaluation. Examining Fig. 5a, the overhead for MABS already surpasses the time required for the NSGA-III algorithm, indicating that using the MABS in conjunction with surrogate models will never be as efficient as a generic NSGA-III algorithm.

We conclude by noting a couple of important aspects with the current MABS framework. The ability to parallelize agents can greatly reduce the time required for developing the PF; however, the exact time can vary greatly. The variation in time requirements resides with the use of various agent types, where agents that are easier to parallelize will in turn reduce the overall time requirements of the problem. For example, creating a large number of offspring with the GA KA-S agent would allow for a large number of design evaluations to happen simultaneously. While the KA-S agents have been implemented in parallel, there is additional room for both adding new agent types to reduce the computational time required. Calculating the HV parameter during a simulation can require considerable time, and



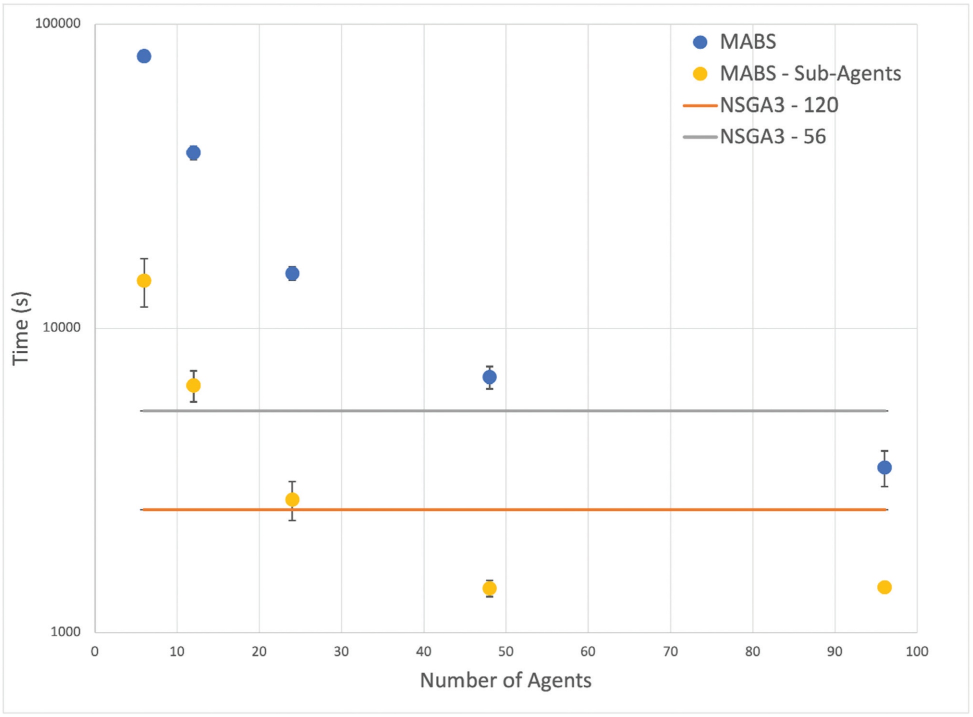
(a) Algorithm Run Time.



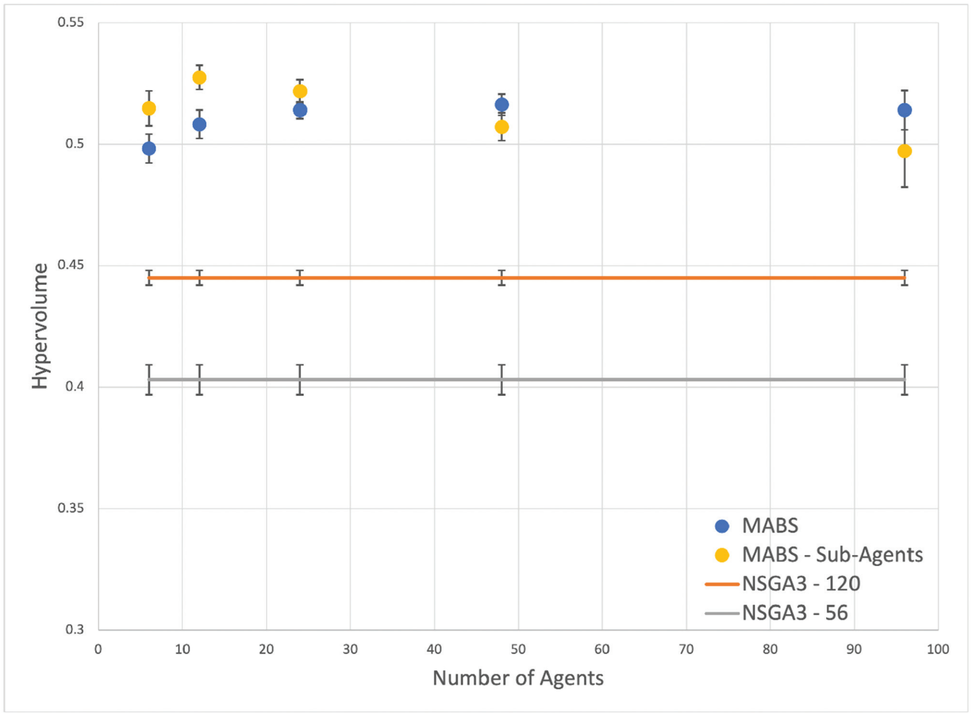
(b) Hypervolume

Figure 5: Results for 0 second function evaluation time.

as such, creating a KA to calculate the HV for the BA could significantly reduce the time the BA must take. Similarly, for many of the problems in the parallelization study, the BA took on the order of 100–250 seconds to write information to the HDF5 file: If this task was passed to a KA, the overall time requirements could shrink. While this may not be important when examining problem with large design evaluation times, it would ensure that the BA is able to constantly respond to KA requests.



(a) Algorithm Run Time.

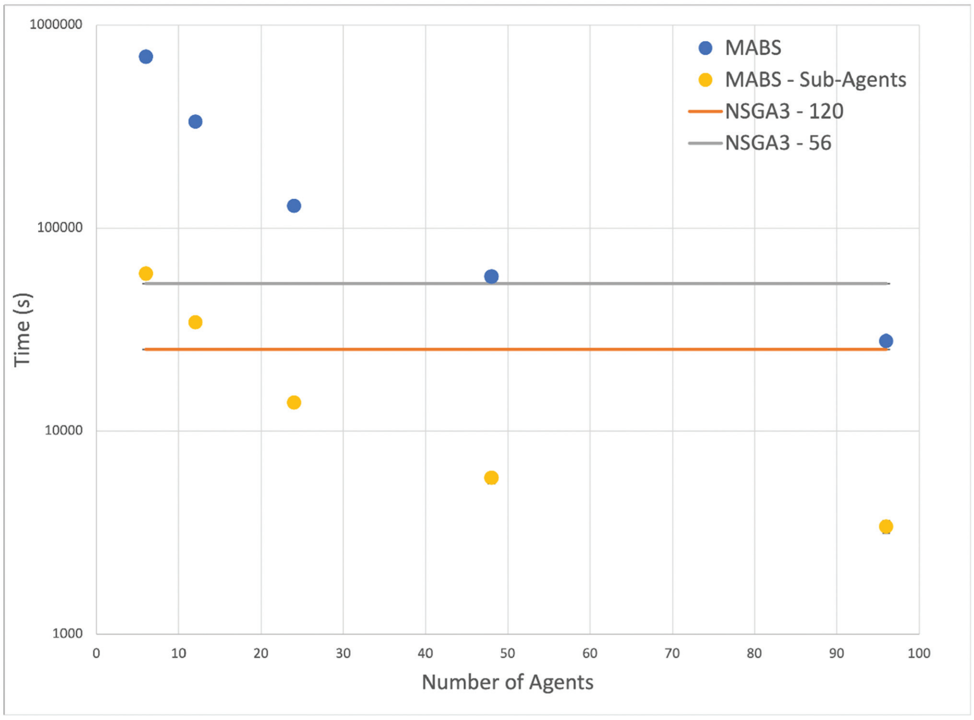


(b) Hypervolume

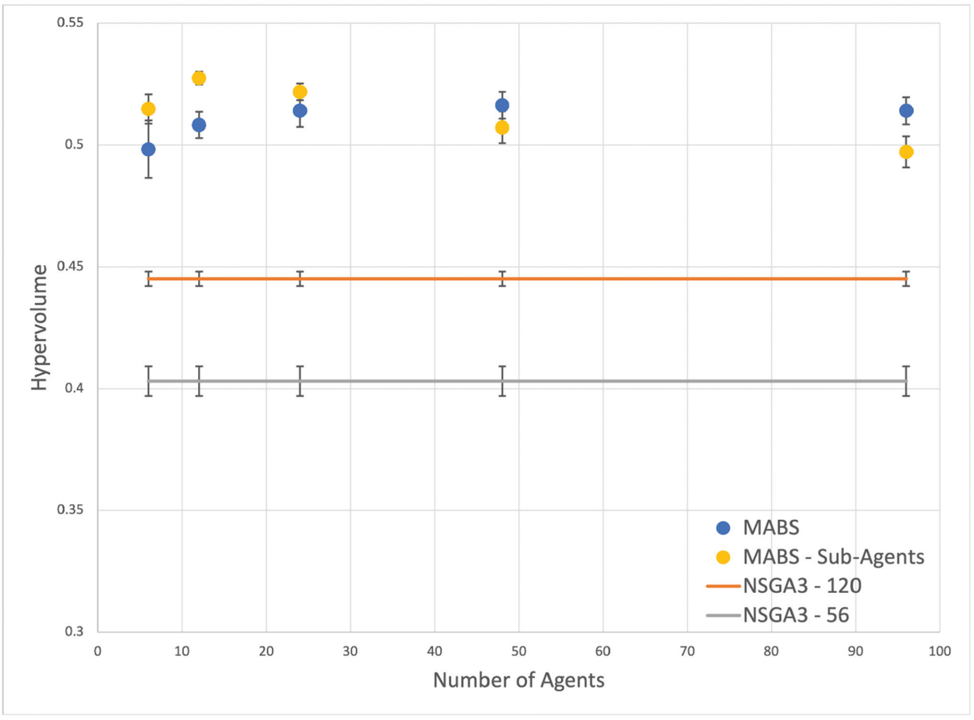
Figure 6: Results for 30 second function evaluation time.

3.5 Hypervolume convergence study

The last study performed for the MABS was the use of the HV as a convergence criterion, again looking at problem RE4-7-1. The MABS was compared against the NSGA-III algorithm, where the same population constraints were used. A simulation time of 30 seconds was selected, where the simulation time and HV were examined as figures of merit. For the MABS, the six cases with varying numbers of agents (shown in Table 10) were used again. The convergence interval was 10 TVs, and a convergence



(a) Algorithm Run Time.

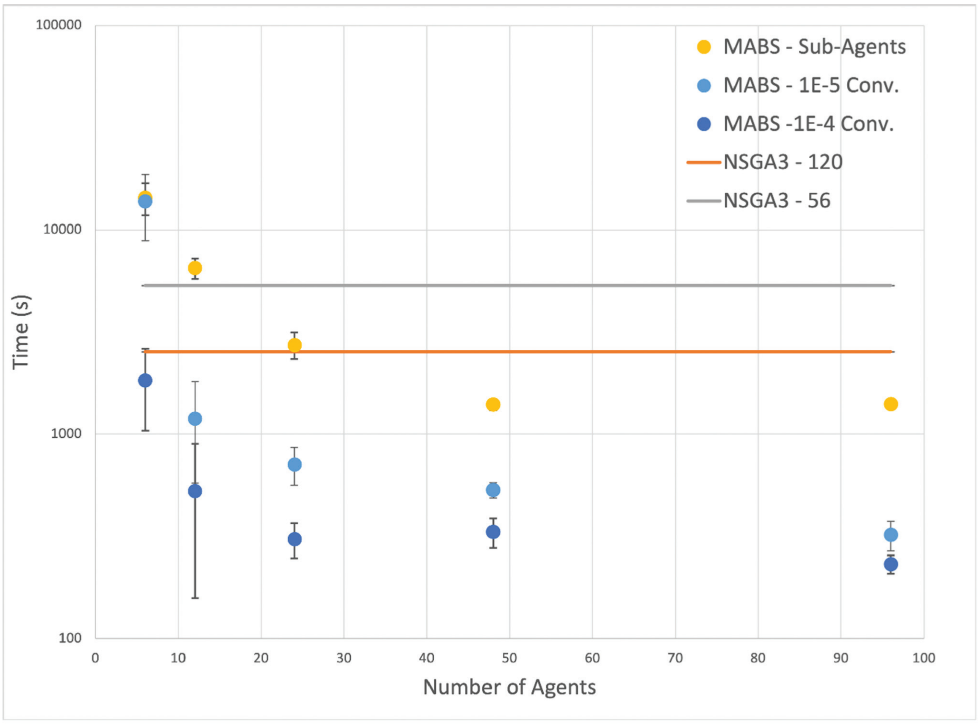


(b) Hypervolume

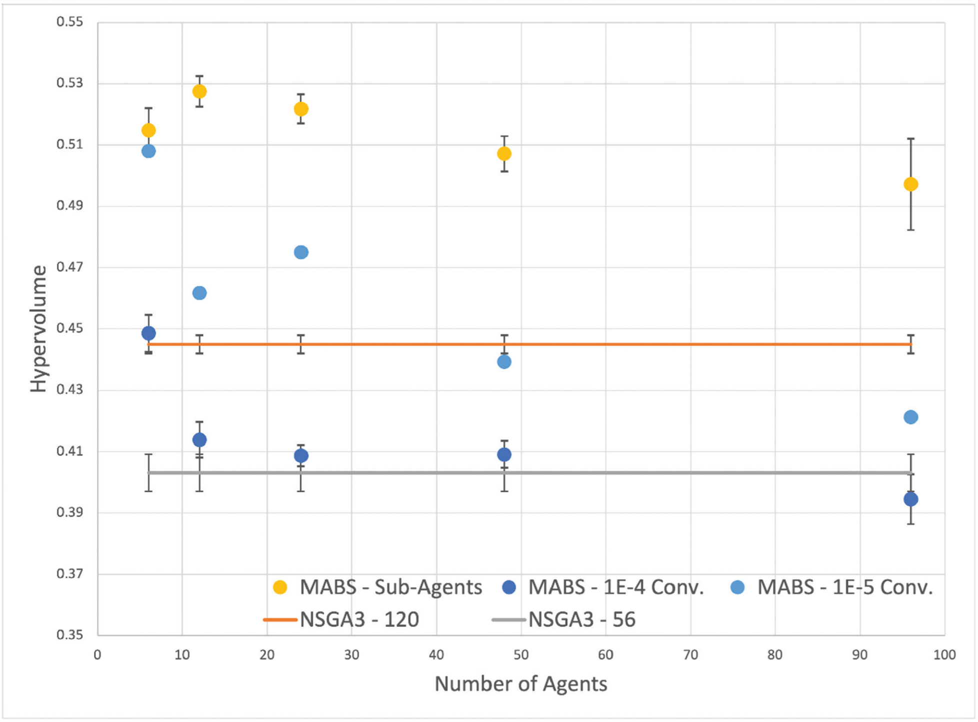
Figure 7: Results for 300 second function evaluation time.

criterion of 10^{-4} and 10^{-5} was used. A maximum of 10 000 function evaluations were allowed for MABS if the HV did not converge. Figure 8a and b shows the results for the two convergence criteria compared with the NSGA-III algorithm.

Using the HV as a convergence criterion can provide a significant decrease in the amount of time required to reach an acceptable PF. There is between a fourfold to twelvefold decrease in the time requirements when using a convergence criterion



(a) Algorithm Run Time.



(b) Hypervolume

Figure 8: Results for 30 second function evaluation time using HV convergence.

of 10^{-4}; this is due to the fact that only 1500–2500 simulations were examined. When using a convergence criterion of 10^{-5}, the time requirements are lessened to between a twofold to fourfold decrease. Also of interest is the fact that simulations using greater agents require similar time-scales. This is likely due to using multiple agents, with varying hyperparameters, which allows for a large PF to be discovered and sampled simultaneously. For the cases with 6 and 12 agents, fewer agents means that certain areas of the PF are more thoroughly explored before the remaining PF was developed and explored. This is further explained when using six agents with a convergence criterion of 10^{-5}, which has a time requirement (and HV value) that is within the uncertainty of the base MABS case. This arises naturally, as the number of function evaluations for this cases approaches 10 000, indicating that the HV does not converge well with a limited number of agents.

Using the hypervolume as a convergence criterion plays a strong role in the time requirements for converging to the PF; however, there is a tradeoff as the PF is not as thoroughly explored. Figure 8b shows that the HV is on par with the NSGA-III algorithm using a populations size of 56. Overall, the HV when using a convergence criterion of 10^{-4} is roughly 75–80%, of the value obtained by using MABS without the convergence criteria. This value improves to 85–90% when using a convergence criterion of 10^{-5}; however, the consequence of improving the HV coverage comes at the cost of increased run-time. There is an inherent tradeoff to using the HV as a convergence criterion; the smaller we make the convergence criterion, the more time it will take to reach convergence. This gives the user the ability to determine whether speed or accuracy in developing the PF is of greater importance. Also, using the HV convergence criteria would provide little to no benefit when examining problems with little to no evaluation time.

Using the HV as a convergence criterion can be beneficial to reducing the number of function evaluations; however, care must be taken to ensure that the problem converges to the true PF. As such, introducing a convergence criterion can provide an initial PF for users to examine in a fraction of the time that would be required for the MABS to complete a fixed number of function evaluations. While this introduces an added level of complexity for determining the PF, if the user is not satisfied with the PF, the convergence criteria can be updated and the problem can be restarted with little difficulty due to the HDF5 file.

3.6 Application for experiment placement in a nuclear test reactor

Nuclear test reactors are mainly used for the irradiation of novel fuel or structural materials for post-irradiation examination. Irradiating materials require placing experiments into dedicated facilities in the reactor core. There is a limit on the number of locations for placing experiments, and the presence of experiments must not diminish the safe operation of the reactor. We examine the placement of experiments in a proposed test sodium fast reactor (SFR).

Three experimental assemblies will be considered for examination; fertile-free fuel (fissile exp.), fertile fuel (fertile exp.), and nonfuel (steel exp.). The majority of experiments envisioned for an SFR will be enveloped by one of these three experiment types. For a fissile exp., minor characteristics will change when replacing the fuel matrix or fissile material, but it is envisioned that using plutonium as the fissile material will help provide a bounding case for fueled experiments. The fissile exp. is modeled as an advanced fuel material that is comprised of 60 wt% plutonium and 40 wt% zirconium based on the AFC-1B (serial number AE-1A) experiment presented in Hilton *et al.* (2006). The plutonium isotopic vector is weapon grade (i.e. 96 wt% Pu-239 and 4 wt% Pu-240); while this may not be indicative of all nonfertile experiments, it will provide a bounding case. The fissile exp. assembly geometry resembles the standard fuel assembly seen in Table 13. Fertile experiments will behave similarly to a neutron poison in the SFR core, and utilizing a uranium-based fuel matrix will provide characteristics similar to most fertile system. The fertile experiment represents a material that is 90 wt% natural uranium and 10 wt% zirconium. The assembly for a fertile experiment contains 91 fuel pins with a pin diameter of 1.25 cm, a fuel density of 80%, and a pitch-to-diameter ratio of 1.078. Multiple types of metal alloys will likely be placed in the SFR core to determine alloy degradation, though they are not expected to drastically affect core neutronics. The steel exp. is representative of a standard assembly (from Table 13) with steel pins.

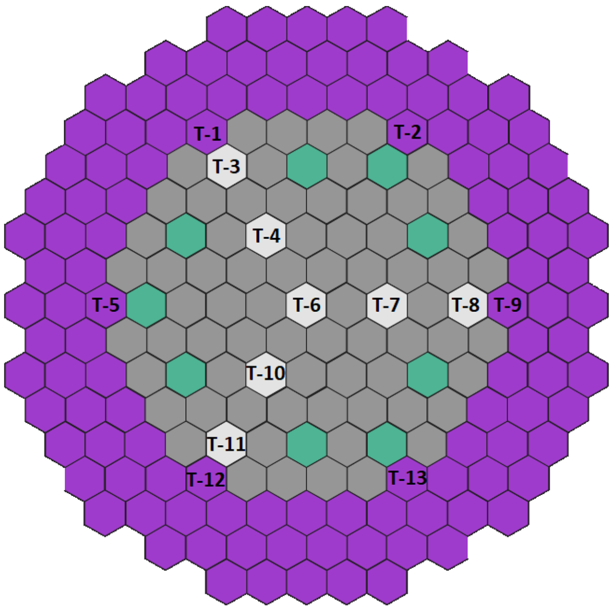


Figure 9: Plan view of the SFR core with experiment locations.

The SFR core layout is shown in Fig. 9 and contains 69 fuel assemblies (grey), 9 safety/control assemblies (green), and 7 empty test locations filled with sodium (white); these initial 6 rings comprise the core region. The core region is surrounded by three rings of stainless steel reflectors (purple). In addition to the seven dedicated in-core experiment locations, six additional locations are allowed near the reflector regions. Allowable experiment locations are denoted with a “T” in Fig. 9. The design variables are the 13 experiment locations T-1 to T-13, and each of these locations can exist in one of the four states, as seen in Table 12. This problem is denoted as Experiment Placement I.

The nominal fuel assembly design specification and remaining details for the core design are provided in Tables 13 and 14. Fuel assemblies remain in the same position for three cycles (1200 days) and are then removed from the core. We assume that these assemblies have been placed in a repeating pattern of fresh, once-, and twice-burned assemblies. The beginning of cycle (BOC) and end of cycle excess reactivity for this core are 3772 and 1350 pcm with a 3σ uncertainty of 55 and 75 pcm, respectively. The notation of “pcm” (per cent milli) is used to denote the difference in reactivity compared to a critical core; i.e. pcm = $\frac{k_{\mathrm{eff}}-1.0}{k_{\mathrm{eff}}} * 10^5$. The multiplication factor (k_{eff}) is the change in the neutron population from one generation to another, where k_{eff} = 1 indicates that the neutron population is self-sustaining. A large amount of excess reactivity is required for a test reactor to accommodate the negative reactivity burden associated with neutron-absorbing materials. This core does not represent an optimized design for any one particular purpose; however, it serves as a basis for comparing the effects of adding experiments with some degree of physical realism.

Table 12: Design variables for Experiment Placement I.

Design variable	Options
T-1, T-2, T-5, T-9, T-12, T-13	[Fissile Exp., Fertile Exp., Steel Exp., Reflector]
T-3, T-4, T6, T-7, T-8, T-10, T-11	[Fissile Exp., Fertile Exp., Steel Exp., Empty]

Table 13: Nominal fuel design description (Stewart *et al.*, 2021c).

Assembly parameter	Value
Assembly pitch (cm)	14.5
Duct thickness (cm)	0.394
Assembly gap (cm)	0.432
Pins per assembly	217
Pin diameter	0.808
Clad thickness (cm)	0.0599
Wire wrap diameter (cm)	0.0808
Fuel pin pitch-to-diameter ratio	1.1
Smear density	75%
Fuel height (cm)	101.6

Table 14: Core design description (Stewart *et al.*, 2021c).

Core parameter	Value
Core power (MW)	300
Fuel assemblies	69
Primary CR assemblies	6
Safety CR assemblies	3
Peak LHGR (kW/m)	35.0
Peak burnup (MWD/kg)	105
Fuel residency (days)	1200
Cycle length	400

Table 15: Objective functions and constraints for Experiment Placement I.

Constraint	Range	Objective	Range	Goal
Cycle length (EFPD)	>365	Fissile Exp.	0–13	Max.
MLHGR (kW/m)	25.0–40.0	Fertile Exp.	0–13	Max.
CR insertion (cm)	10–90	Steel Exp.	0–13	Max.
		Total Exp.	0–13	Max.

The objective functions and constraints placed on the problem are listed in Table 15. For objective functions, we are maximizing the number of each experimental assembly type individually, along with the total number of experiments placed in the core. The three constraints represent basic safety and operating parameters that must be met to operate the SFR. A limit of 365 days ensures that the core will have enough fissile material to operate to within approximately 10% of the 400 day cycle length. The maximum linear heat generation rate (MLHGR) ensures that the fuel is operating within safety parameters. The MLHGR is calculated for each of the six assemblies in row 2 of the core. The BOC control rod (CR) insertion depth is used to assess the ability to shut the reactor down. The CR insertion depth represents the amount of negative reactivity that must be inserted into the core to counteract the positive reactivity associated with fresh fuel. This position corresponds to a critical core, i.e. the nuetron multiplication factor $k_{\mathrm{eff}} = 1$. The range presented ensures that the reactor will be able to shut down during operations. Each of the objective functions and constraints are stored as a single value with the exception of the MLHGR. The MLHGR stores each of the six (there are six fuel assemblies in row 2) LHGRs in a list, where the maximum value in the list is used to determine whether the constraint is violated.

MABS was executed on Idaho National Laboratory's (INL) HPCs. To calculate the objective functions and constraints, a high-fidelity Serpent2 (Leppanen, 2015) Monte Carlo neutronics model was used. Each simulation takes approximately 20 minutes to complete, with up to *N* simulations executing simultaneously, where *N* is the number of KA-S agents. For this simulation, 12 KA-S agents were used (2 stochastic, 1 Latin hypercube, 3 neighborhood search, 3 hill-climb, and 3 GAs). Agents are executed as separate processes on a single HPC node. KA-S agents submit jobs to the HPC, wait for the simulation to complete, read the results from an output file, and write this information to the blackboard. 1000 function evaluations were used to terminate the problem.

The PF discovered by the MABS is shown in Fig. 10. Over 200 combinations of the multiple experiment types are found, with a majority of solutions utilizing all 13 experimental assembly locations. The number of fissile experiments is limited to six or fewer. This is likely because adding more than six fissile experiments breaches our limits for CR insertion depth given the addition of fuel to the core. This is confirmed when examining the two heat maps for experiment placement in Figs 11 and 12. We find that the fissile experiments are typically placed in the exterior of the core (ring 6), with a small percentage of core designs placing fissile experiments in position T-4 and T-8. Placing the fissile experiments in the core periphery reduces the amount of reactivity added to the core, which increases the number of experimental assemblies that can be incorporated in the design.

Due to the large number of solutions present in the PF, the problem is expanded in an attempt to further utilize the core. To do this, the number of design variables is increased by allowing each fresh fuel assembly to be replaced by an experiment; this problem is denoted as Experiment Placement II. This process is typical in SFRs and has been employed in previous SFR test reactors such as the Experimental Breeder Reactor II and the Fast Flux Test Facility (Koch *et al.*, 1958; Steele, 1967). One-third of the core is replaced each cycle with fresh fuel, where the remaining fuel remains in place. As such, the fresh fuel can be replaced with an experiment during the refueling process, as seen in Fig. 13. The original 13 test positions from Fig. 9 are also available, but not shown in the figure. Three fresh assembly locations are not considered for experiment placement as these assemblies are adjacent to the center of the core and are used to determine the MLHGR. The design variable options for T-1 to T-13 are consistent with Table 12; the design variable options for T-14 to T-33 are provided in Table 16. The objective functions and constraints are modified slightly to include the addition of new experiment locations and are captured in Table 17.

Experiment Placement I was a subset of the Experiment Placement II problem, and as such, data stored in the HDF5 file from the first optimization problem can be used as a starting point. The HDF5 file from Experiment Placement I was

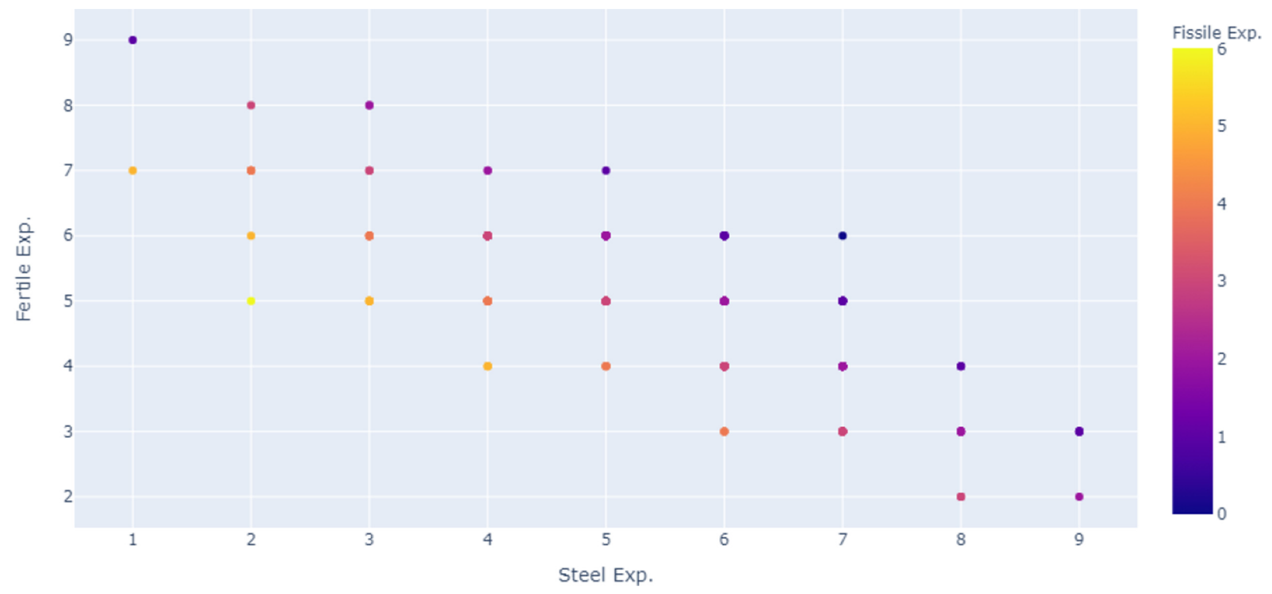


Figure 10: PF for Experiment Placement I in SFR core.

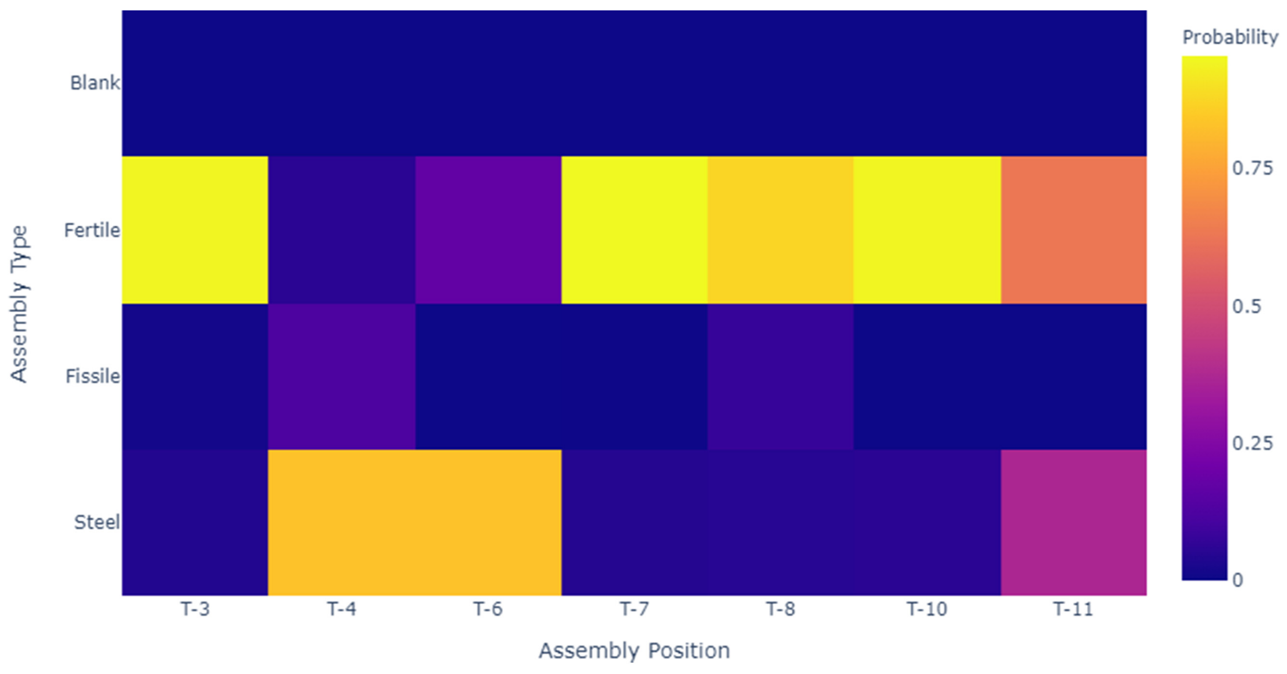


Figure 11: Heat map for Experiment Placement I in inner locations.

loaded, the design variables are updated, and a new optimization problem encompassing Experiment Placement II was executed. The ability to update a problem and start a new optimization process highlights one of the major advantages of the MABS.

Figure 14 shows the PF for Experiment Placement II. Expanding the design space allows for additional experiments to be placed in the core while still maintaining the constraints placed on the design. Similar to the initial experiment placement problem, a majority of the optimal designs found maximize the total number of experimental assemblies. Figure 14 shows a roughly linear relationship between the number of fertile and steel experimental assemblies, arising from the removal of fresh fuel to place a nonfueled experiment. This drives the requirement for a minimum number of fissile experiments to maintain the cycle length.

Figure 15 shows a 2D representation of the PF, including the relationships between the objective functions and the constraints. A few major relationships are important to point out in this figure. The number of fissile experiments drives the range of acceptable core configurations. Beginning with the BOC CR insertion depth, a limit of 12 fissile assemblies is reached before the maximum CR insertion depth is violated. Conversely, when the number of fissile assemblies is reduced, the cycle length trends toward the lower limit of 365 days. There is significant variation in these trends, as the placement of the fissile

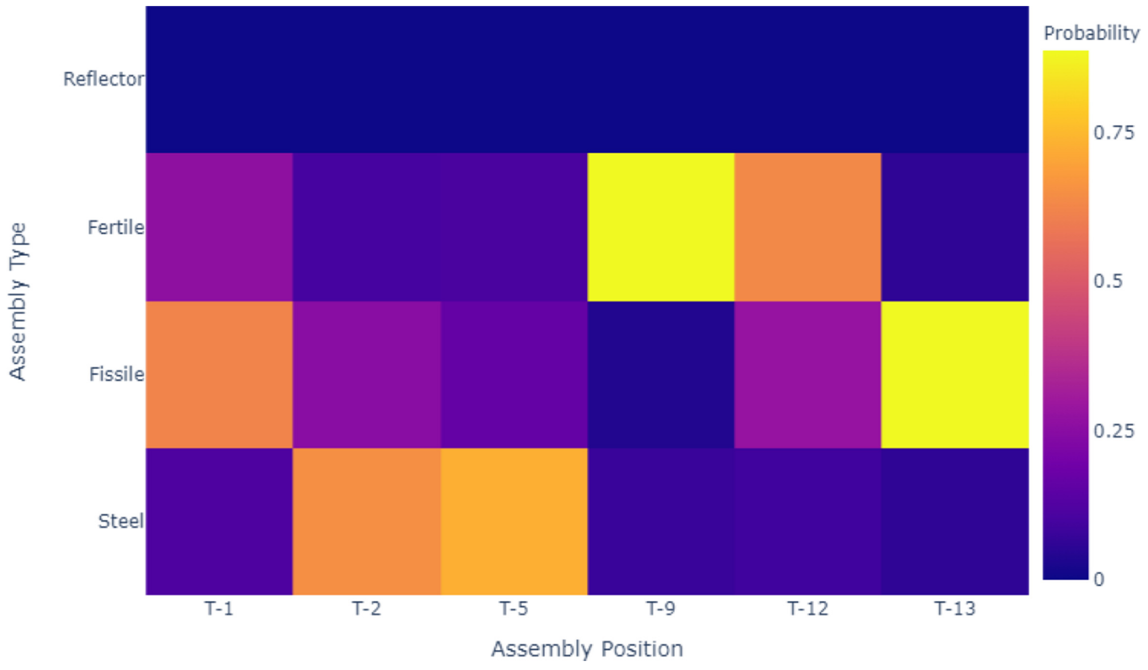


Figure 12: Heat map for Experiment Placement I in outer locations.

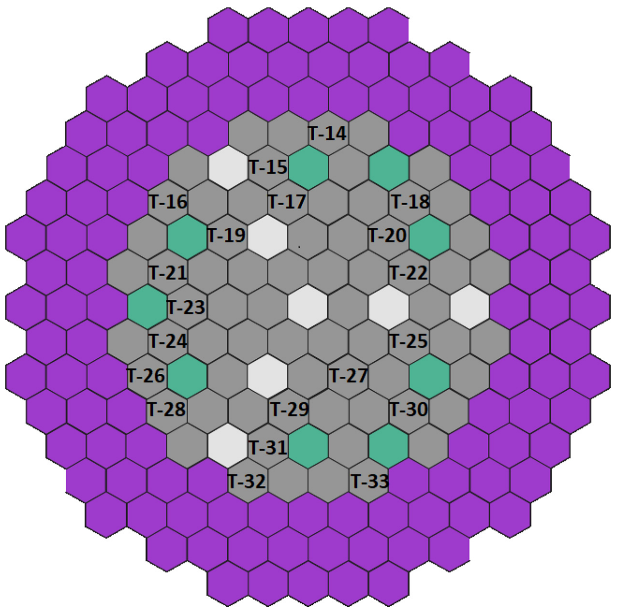


Figure 13: Additional experiment locations for Experiment Placement II.

experiments plays a large role in determining the amount of reactivity added to the core.

We conclude by examining Figs 16–19, which provide heat maps of the 33 experiment locations, and the probability of finding a particular experimental type in each location. Despite removing fresh fuel from the core, the fissile experiments are still placed in the periphery of the core (rows 5 and 6). There are a few locations in rows 1–4 (namely, T-4, T-8, T-20, and T-22) that have a very low possibility of a fissile experiment. The fissile experiments are typically placed in the periphery of the core to ensure that the reactivity addition does not violate the 90 cm CR insertion depth, as seen in Fig. 15. Similar to Experiment Placement I, with 33 experiment locations, there are multiple ways to load the core and ensure we meet the constraints placed on the system.

Table 16: Additional design variables for Experiment Placement II.

Design variable	Options
T-14 to T-33	[Fissile Exp., Fertile Exp., Steel Exp., Fresh Fuel]

Table 17: Objective functions and constraints for Experiment Placement II.

Constraint	Range	Objective	Range	Goal
Cycle length (EFPD)	>365	Fissile Exp.	0–33	Max.
MLHGR (kW/m)	25.0–40.0	Fertile Exp.	0–33	Max.
CR insertion (cm)	10–90	Steel Exp.	0–33	Max.
		Total Exp.	0–33	Max.

4 Discussion and Conclusions

4.1 Discussion

The genesis of MABS was to build an optimization algorithm that could utilize a high degree of parallelizability for optimizing engineering-based problems. Many engineering problems require the use of high-fidelity simulations that can take on the order of minutes to hours. If hundreds or thousands of simulations are required for an optimization problem, the time requirements would be intractable. However, researchers analysing engineering design type problems frequently have access to HPCs, in which thousands of computational nodes are available at a given time. It is advantageous then to utilize HPCs to run high-fidelity simulations, and the optimization algorithm, in parallel. The MABS framework has been built to utilize parallelizability

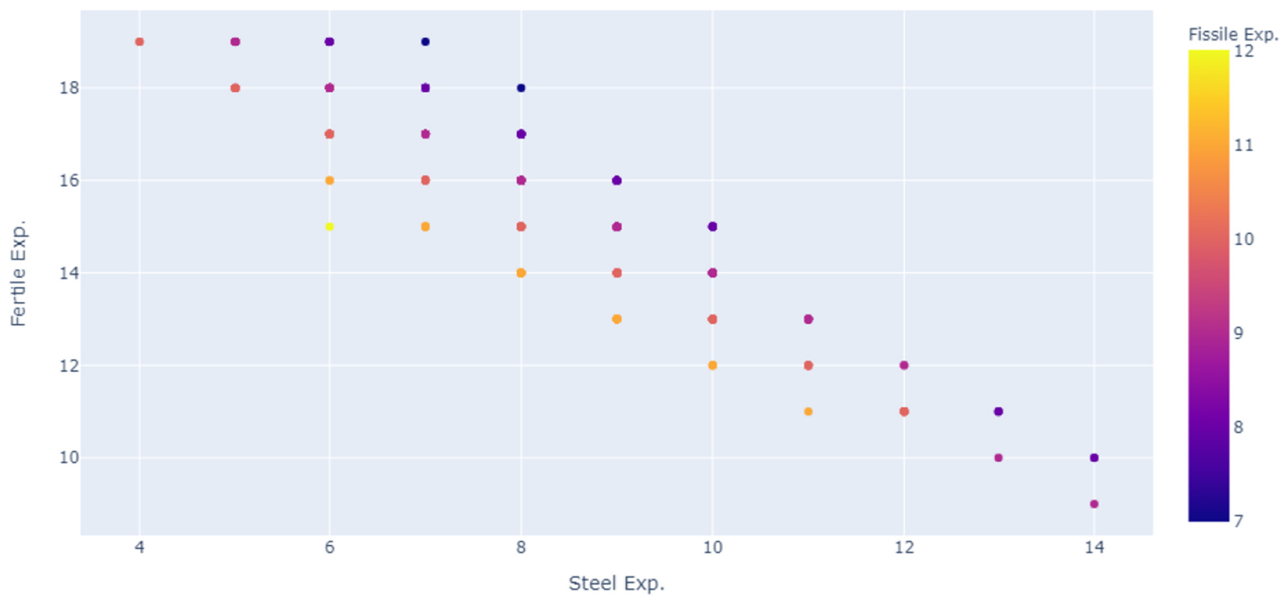


Figure 14: PF for Experiment Placement II in SFR core.

in two major ways: the use of multiple agents to carry out tasks and the use of subagents to help perform these tasks.

Many optimization algorithms, especially evolutionary algorithms, require the use of generations to propagate the PF. Each generation represents a set number of designs that are deemed to be optimal. The use of generations has inherent consequences for the parallelizability for an optimization algorithm, as information from the previous generation is required for determining where the next set of designs is examined. To this extent, the maximum number of concurrent design calculations that can be performed is set by the size of each generation. However, MABS attempts to capitalize on the available parallelizability present in HPCs. Currently, MABS can utilize tens to hundreds of agents for an optimization problem, which allows for a highly scalable parallel optimization algorithm that can maximize the available computational architecture.

While utilizing multiple agents can be advantageous, many of the optimization algorithms present in MABS can be themselves parallelized. To increase the number of design simulations performed, subagents can be utilized to parallelize an individual optimization algorithm. Each subagent is given a design to evaluate, where multiple subagents can be spawned by a single KA. This process allows optimization algorithms to be broken into components that can be computed simultaneously, where appropriate. For the *Genetic Algorithm* KA, the number of subagents spawned will be equal to the number of offspring allowed per generation (N). If a high-fidelity simulation took time t to solve and the *Genetic Algorithm* KA were run in serial, it would take $N*t$; when parallelized it would only take t. Depending on the simulation time, this reduction by a factor of N can significantly improve the rate at which designs are analysed.

Optimization algorithms require some approach to remove undesirable solutions from the PF. Some algorithms, such as the Non-dominated Sorting Genetic Algorithm II (NSGA-II), rely on crowding distance sorting to continually produce the most diverse set of designs possible (Deb *et al.*, 2002). The crowding distance measures the similarity between designs, and uses this information to remove designs that are clustered together in favor of designs that are at a greater distance from each other. Other algorithms, such as NSGA-III, require a set of reference directions for sorting and maintaining the PF (Deb & Jain, 2014a). The reference directions help prune the PF by determining solutions that are nearest to an underrepresented reference direction. This approach is highly successful in ordering and maintaining the PF; however, this approach is only updated after each generation (or set of examined designs). Conversely, the multiagent environment allows for near real-time updating of the PF to remove dominated or crowded solutions. Measures similar to the crowding distance can be implemented; however, they are actuated each time the PF is updated, allowing the most up to date PF possible for any future KA. In this approach, we expedite the process of finding a valid PF by quickly removing undesirable designs.

Given the emphasis placed on developing an optimization algorithm that is focused on leveraging HPCs, there are multiple inherent drawbacks to MABS. While these drawbacks do not discount the usefulness of MABS, they are important to know, so that researchers interested in using this tool know the limitations of the code suite. MABS was built to be implemented within an HPC architecture, and as such, is less powerful on a desktop machine with a small number of cores. To apply large numbers of agents and subagents, multiple threads and processes must be available for use. On a desktop, this limit can be reached before fully employing all of the desired agents and subagents. Similarly, if high-fidelity simulations are required, a local machine often does not have the computational resources or memory to execute parallel simulations, thereby nullifying many of the benefits provided by MABS. While low-fidelity or surrogate models can be used with MABS, many of the accompanying benefits are dampened. As shown in Section 3.4, when utilizing MABS with a simulation that is executed on the order of seconds, there is a penalty to the time required to obtain the PF. Some benefits of MABS persist with the use of low-fidelity simulations on local machines. For example, the creation of a searchable HDF5 file is advantageous, along with the ability to store large amounts of data associated with a given design, but

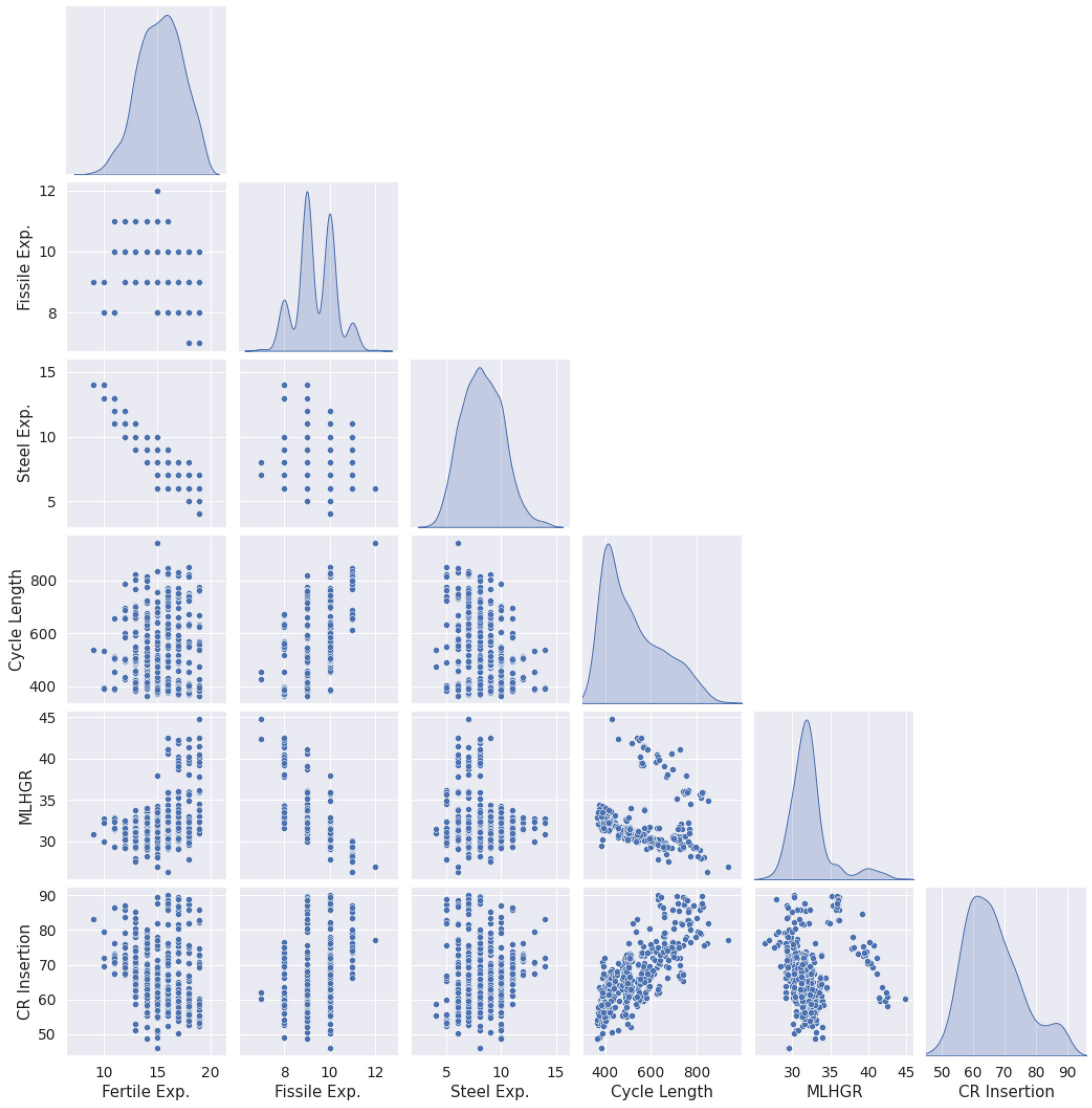


Figure 15: 2D PF for objective functions and constraints for Experiment Placement II in an SFR core.

not explicitly used in the optimization process. These benefits are more advantageous if high-fidelity simulations are used, as it prevents having to re-execute these simulations to obtain this information.

4.2 Conclusions

A multi-objective optimization algorithm based on the MABS was developed to solve engineering-based optimization problems. The MABS framework was tested against three commonly used optimization algorithms on 16 engineering benchmark problems. The MABS was able to search the objective space and discover the PF in an efficient manner, and consistently produced performance metrics that are competitive with common optimization algorithms. A parallelization study was also performed for the MABS, resulting in the observation that the time required to obtain the PF can be impactfully reduced by the addition of more agents. Increasing the number of agents provides greater benefits when the time required to produce a set of objective functions/constraints is large, indicating that the MABS is better suited for performing optimization using high-fidelity simulations. For simulations taking over 300 seconds, we find that MABS using two types of parallelization (number of agents and subagents) can allow for significant time savings to evaluate a set number of function evaluations. Along with this, using the HV convergence method can reduce the time requirements for developing the PF by two to 12 times depending on the convergence criteria and number of agents that are used.

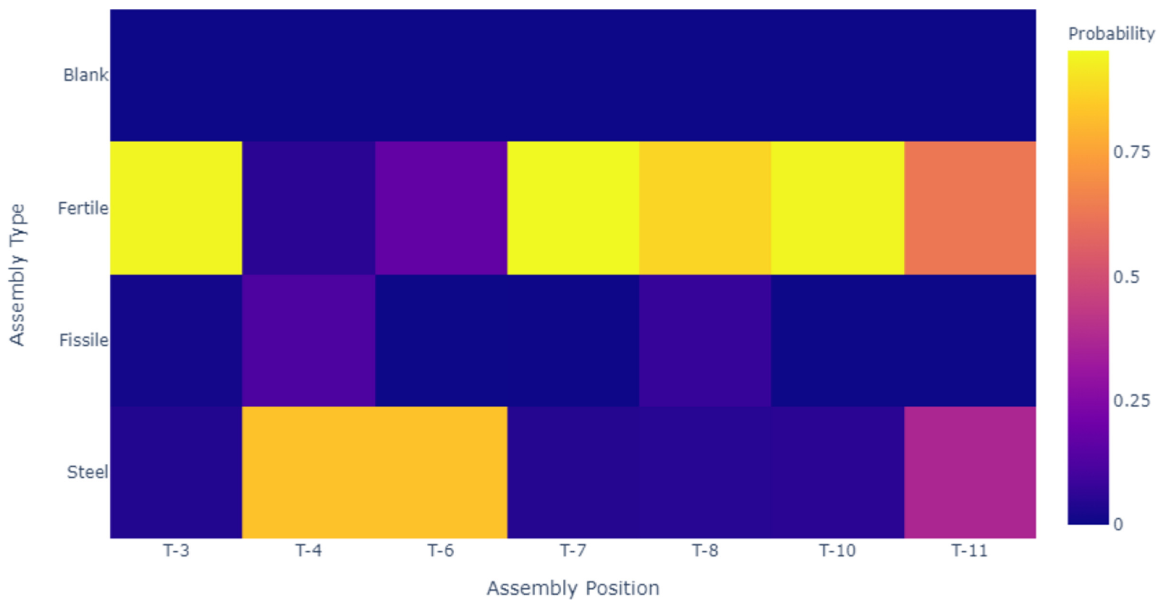


Figure 16: Heat map for inner experiment locations for Experiment Placement II.

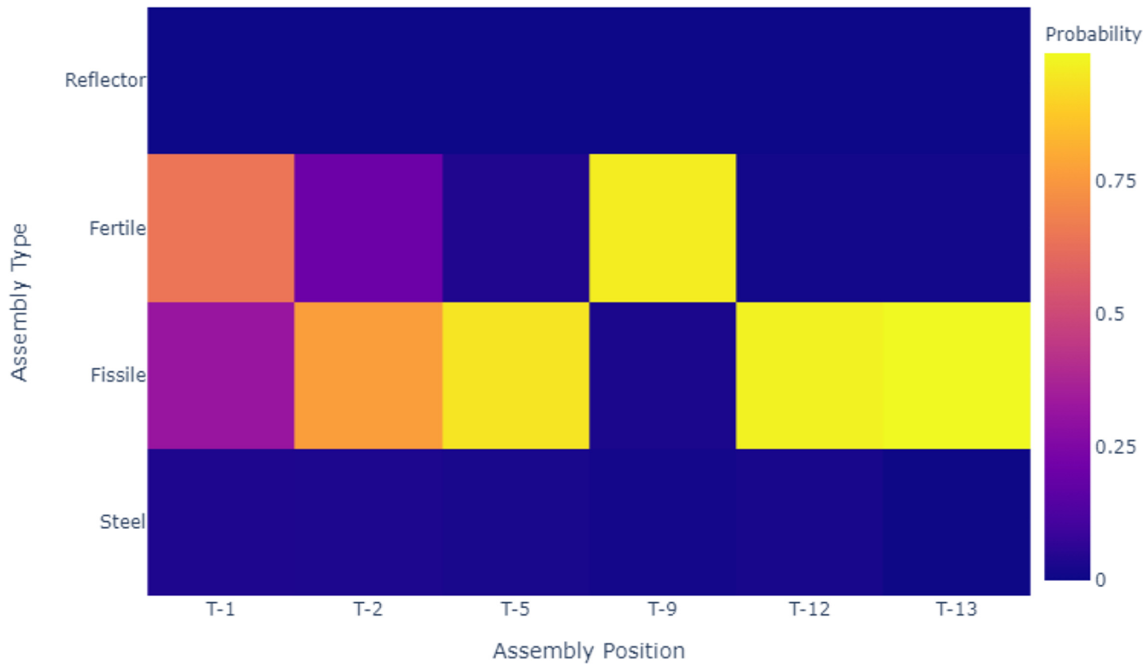


Figure 17: Heat map for outer experiment locations for Experiment Placement II.

The MABS was also applied to a nuclear engineering problem to determine the optimal number of experiments that can be placed in a test reactor while still being able to maintain operational and safety parameters. The initial results provided multiple core configurations that met all of the objectives and constraints placed on the problem. This initial problem formed the basis of a second optimization problem with additional design variables to increase the usability of the test reactor. The second optimization problem expanded the number of viable core designs that utilized all test available locations while balancing the multiple constraints placed on the reactor design.

We have shown that the MABS provides a unique method for solving real-world engineering optimization problems. The versatility presented in this paper highlights this approach, as the blackboard system can store additional information typically not present in other optimization algorithms. Data from a prior optimization problem can readily be used as a starting point for additional problems or in a multitiered optimization problem. Overall, the MABS provides the fundamentals for solving multiobjective optimization problems and leverages an agent-based blackboard system to provide a methodology that can be used to solve real-world engineering problems.

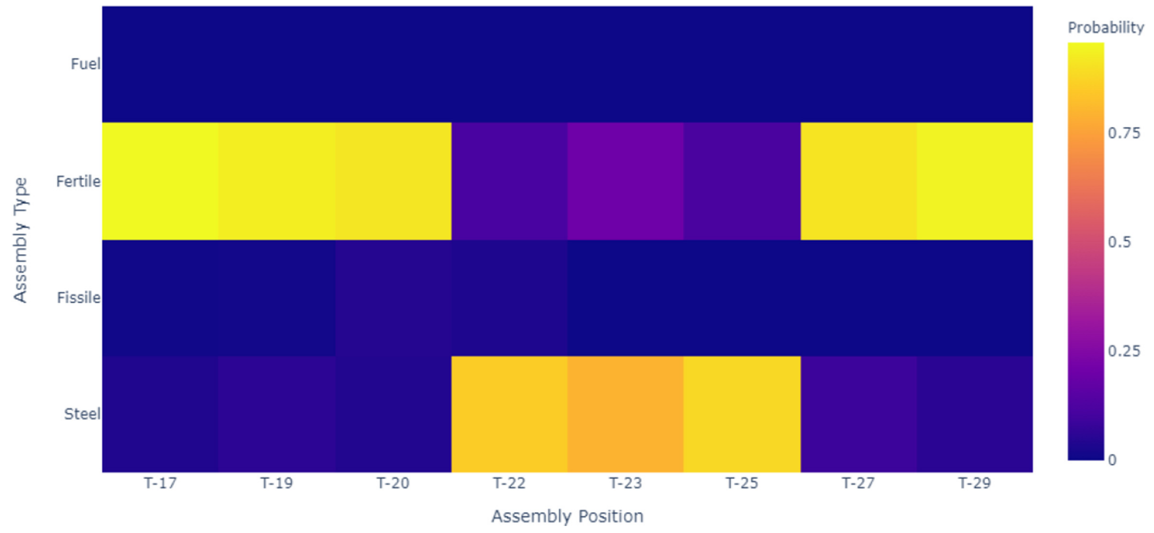


Figure 18: Heat map for rows 3/4 for Experiment Placement II.

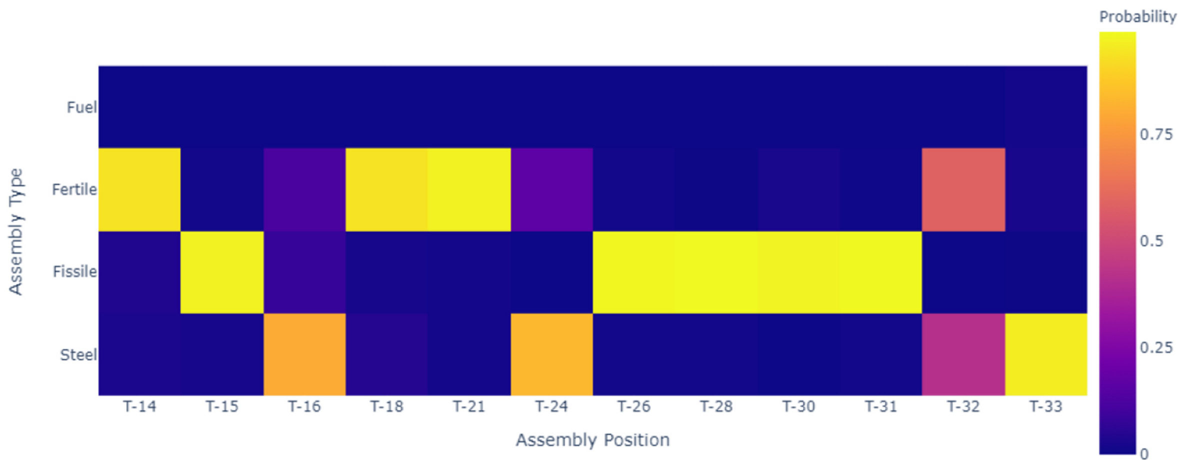


Figure 19: Heat map for rows 5/6 for Experiment Placement II.

Acknowledgments

This research made use of the resources of the High Performance Computing Center at INL, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under contract number DE-AC07-05ID14517.

Conflict of interest statement

None declared.

References

Alaya, I., Solnon, C., & Ghedira, K. (2007). Ant colony optimization for multi-objective optimization problems. In *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*. (Vol. 1, pp. 450–457). https://doi.org/10.1109/ICTAI.2007.108.

Atashpaz-Gargari, E., & Lucas, C. (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In *2007 IEEE Congress on Evolutionary Computation*. (pp. 4661–4667). https://doi.org/10.1109/CEC.2007.4425083.

Blank, J., & Deb, K. (2020). Pymoo: Multi-objective optimization in Python. *IEEE Access*, 8, 89497–89509.

Brzykcy, G., Martinek, J., Meissner, A., & Skrzypczynski, P. (2001). Multi-agent blackboard architecture for a mobile robot. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the Next Millennium (Cat. No. 01CH37180)*. (Vol. 4, pp. 2369–2374). https://doi.org/10.1109/IROS.2001.976424.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2002). *Evolutionary algorithms for solving multi-objective problems*(Vol. 242).

Corkill, D. (1991). Blackboard systems. *AI Expert*, 6(9), 1–7. http://mas.cs.umass.edu/paper/218.

Corkill, D. D., Gallagher, K. Q., & Johnson, P. M. (1988). Achieving flexibility, efficiency, and generality in blackboard architectures. In A. H. Bond, L. Gasser, & Morgan Kaufmann (Eds), *Readings in distributed artificial intelligence*. (pp. 541–546). https://doi.org/10.1016/B978-0-934613-63-7.50053-X.

Craig, I. (1995). *Blackboard systems*. Alex Publishing Corporation.

Deb, K., & Jain, H. (2014a). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, *18*(4), 577–601. https://doi.org/10.1109/TEVC.2013.2281535.

Deb, K., & Jain, H. (2014b). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part II: Handling constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation*, 18(4), 602–622. https://doi.org/10.1109/TEVC.2013.2281534.

Deb, K., Laumanns, M., & Zitzler, E. (2005). *Scalable test problems for evolutionary multiobjective optimization*. Springer. https://doi.org/10.1007/1-84628-137-7_6.

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

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, *1*(4), 28–39. https://doi.org/10.1109/MCI.2006.329691.

Gebreslassie, B. H., & Diwekar, U. M. (2018). Heterogeneous multiagent optimization framework with application to synthesizing optimal nuclear waste blends. *Clean Technologies and Environmental Policy*, 20, 137–157. https://doi.org/10.1007/s10098-017-1464-4.

Glover, F. (1990). Tabu search: A tutorial. *Interfaces*, *20*(4), 74–94.

Hansen, M. P. (1997). Tabu search for multiobjective optimization: MOTS. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making*. Springer-Verlag.

Hassanzadeh, H. R., & Rouhani, M. (2010). A multi-objective gravitational search algorithm. In *2010 2nd International Conference on Computational Intelligence, Communication Systems and Networks*. (pp. 7–12). https://doi.org/10.1109/CICSyN.2010.32.

Hilton, B., Porter, D., & Hayes, S. (2006). *AFC-1 transmutation fuels post-irradiation hot cell examination 4–8 at.% – Final report (irradiation experiments AFC-1B, -1F and -1Æ), Technical report: INL/EXT-05-00785, DOE contract number: DE-AC07-99ID-13727*. https://doi.org/10.2172/911779.

Hulse, D., Gigous, B., Tumer, K., Hoyle, C., & Tumer, I. Y. (2017). Towards a distributed multiagent learning-based design optimization method. In *43rd Design Automation Conference of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, V02AT03A008*. (Vol. 2A). https://doi.org/10.1115/DETC2017-68042.

Hulse, D., Tumer, K., Hoyle, C., & Tumer, I. (2019). Modeling multidisciplinary design with multiagent learning. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 33(1), 85–99. https://doi.org/10.1017/S0890060418000161.

Jain, H., & Deb, K. (2014). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation*, 18(4), 602–622. https://doi.org/10.1109/TEVC.2013.2281534.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 – International Conference on Neural Networks*. (Vol. 4, pp. 1942–1948). https://doi.org/10.1109/ICNN.1995.488968.

Kirkpatrick, S., Gelatt, C. D. Jr, & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.

Koch, L. J., Monson, H. O., Simmons, W. R., Levenson, M., Verber, F., Hutter, E., Jaross, R. A., Spalding, T. R., Simanton, J. R., & Lovoff, A. (1958). *Construction design of EBR-II: An integrated unmoderated nuclear power plant, Technical report: A/CONF.15/P/1782, NSA number: NSA-12-015073*. https://doi.org/10.2172/4325807.

Lee, K. S., & Geem, Z. W. (2004). A new structural optimization method based on the harmony search algorithm. *Computers and Structures*, 82(9), 781–798. https://doi.org/10.1016/j.compstruc.2004.01.002.

Leppanen, J. (2015). *Serpent – A continuous-energy Monte Carlo reactor physics burnup calculation code (user's manual), Technical report*. VTT Technical Research Centre of Finland.

Li, K., Chen, R., Fu, G., & Yao, X. (2019). Two-archive evolutionary algorithm for constrained multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, *23*(2), 303–315. https://doi.org/10.1109/TEVC.2018.2855411.

Lones, M. A. (2014). Metaheuristics in nature-inspired algorithms. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp'14. (pp. 1419–1422). Association for Computing Machinery. https://doi.org/10.1145/2598394.2609841.

McKay, M., Beckman, J., & Conover, W. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239–245.

Melanie, M. (1996). *An introduction to genetic algorithms*. The MIT Press.

Mirjalili, S., Jangir, P., Mirjalili, S. Z., Saremi, S., & Trivedi, I. N. (2017). Optimization of problems with multiple objectives using the multi-verse optimization algorithm. *Knowledge-Based Systems*, *134*, 50–71. https://doi.org/10.1016/j.knosys.2017.07.018.

Mirjalili, S., Mirjalili, S. M., & Hatamlou, A. (2015). Multi-verse optimizer: A nature-inspired algorithm for global optimization. *Neural Computing and Applications*, *27*, 495–513.

Mostaghim, S., & Teich, J. (2003). Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO). In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. (pp. 26–33). https://doi.org/10.1109/SIS.2003.1202243.

Murata, T., & Ishibuchi, H. (1995). MOGA: Multi-objective genetic algorithms. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*.

OpenSistemas (2019). *osBrain: A general-purpose multi-agent system module written in Python*. Available at: https://github.com/opensistemas-hub/osbrain.

Panichella, A. (2019). An adaptive evolutionary algorithm based on non-Euclidean geometry for many-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*. (pp. 595–603). Association for Computing Machinery.

Pydoe (2013). *Pydoe: Design of experiments for Python*. Available at: https://pythonhosted.org/pyDOE/index.html.

Rashedi, E., Nezamabadi-pour, H., & Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information Sciences*, *179*(13), 2232–2248. https://doi.org/10.1016/j.ins.2009.03.004.

Serafini, P. (1994). Simulated annealing for multi objective optimization problems. *Multiple Criteria Decision Making*, 283–292. https://link.springer.com/chapter/10.1007/978-1-4612-2666-6_29#citeas.

Sherinov, Z., & Unveren, A. (2017). Multi-objective imperialistic competitive algorithm with multiple non-dominated sets for the solution of global optimization problems. *Soft Computing*, 22, 8273–8288.

Silva, D. D., Choi, S., & Kim, H. (2019). Multi-agent rover system with blackboard architecture for planetary surface soil exploration. *The Journal of The Institute of Internet, Broadcasting and Communication*, 10(2), 243–253.

Sivasubramani, S., & Swarup, K. (2011). Multi-objective harmony search algorithm for optimal power flow problem. *International Journal of Electrical Power and Energy Systems*, *33*(3), 745–752. https://doi.org/10.1016/j.ijepes.2010.12.031.

Sorsa, A., Peltokangas, R., & Leiviska, K. (2008). Real-coded genetic algorithms and nonlinear parameter identification. In *2008 4th International IEEE Conference Intelligent Systems*. (Vol. 2, pp. 10–42-10-47). https://doi.org/10.1109/IS.2008.4670495.

Steele, C. C. (1967). *Fast flux test facility overall conceptual systems design description, Technical report: BNWL-500, DOE contract number: AT(45-1)-1830*. https://doi.org/10.2172/4555339.

Stewart, R. H., Gleicher, F. N., Martin, N. P., Bays, S. E., Palmer, T. S., Ritter, C., Reyes, G., & Schanfein, M. (2021c). Examination of diversion and misuse detection for a generalized sodium-cooled fast test reactor. *Journal of Nuclear Materials Management*. submitted for review

Stewart, R., Palmer, T., & Bays, S. (2021a). Toward an agent-based blackboard system for reactor design optimization. *Nuclear Technology*, 1–21. https://doi.org/10.1080/00295450.2021.1960783.

Stewart, R., Palmer, T., & DuPont, B. (2021b). A survey of multi-objective optimization methods and their applications for nuclear scientists and engineers. *Progress in Nuclear Energy*, 138, 103830. https://doi.org/10.1016/j.pnucene.2021.103830.

Stewart., R (2019). Multi-Agent Blackboard System. https://github.com/ryanstwrt/multi_agent_blackboard_system.

Szymanski, L., Sniezynski, B., & Indurkhya, B. (2018). A multi-agent blackboard architecture for supporting legal decision-making. *Computer Science*, *19*(4), 457–477. https://doi.org/10.7494/csci.2018.19.4.3007.

Tanabe, R., & Ishibuchi, H. (2020). An easy-to-use real-world multi-objective optimization problem suite. *Applied Soft Computing*, *89*, 106078. https://doi.org/10.1016/j.asoc.2020.106078.

The HDF Group(2000–2020). Hierarchical data format version 5. https://www.hdfgroup.org.

Veldhuizen, D. V., & Lamont, G. (1998). *Multiobjective evolutionary algorithm research: A history and analysis, Technical report TR-98-03*. Air Force Institute of Technology.

Zameer, A., Mirza, S. M., & Mirza, N. M. (2014). Core loading pattern optimization of a typical two-loop 300 MWE PWR using simulated annealing (SA), novel crossover genetic algorithms (GA) and hybrid GA(SA) schemes. *Annals of Nuclear Energy*, 65, 122–131. https://doi.org/10.1016/j.anucene.2013.10.024.

Zitzler, E. (1999). *Evolutionary algorithms for multiobjective optimization: Methods and applications*, Ph.D. thesis. Swiss Federal Institute of Technology Zurich.