Using a Database to Support Interactive Multiobjective Optimization, Visualization, and Analysis

Bhupinder Singh Saini
University of Jyvaskyla
Faculty of Information Technology,
P.O. Box 35 (Agora)
FI-40014 University of Jyvaskyla
Finland
bhupinder.s.saini@jyu.fi

Giomara Lárraga
University of Jyvaskyla
Faculty of Information Technology,
P.O. Box 35 (Agora)
FI-40014 University of Jyvaskyla
Finland
giomara.g.larraga-maldonado@jyu.fi

Kaisa Miettinen
University of Jyvaskyla
Faculty of Information Technology,
P.O. Box 35 (Agora)
FI-40014 University of Jyvaskyla
Finland
kaisa.miettinen@jyu.fi

ABSTRACT

Many libraries of open-source implementations of multiobjective optimization problems (MOPs) and evolutionary algorithms (MOEAs) have been developed in recent years. These libraries enable researchers to solve their MOPs using diverse MOEAs. Some libraries also implement interactive MOEAs, which enable decision-makers (experts in the domain of the MOP) to provide their preferences and guide the optimization process toward their region of interest. These libraries also provide access to visualization methods and benchmarking tools. However, they do not currently implement a database to store and utilize the data generated while running MOEAs.

We propose the creation of SIVA DB, a database designed to be easily incorporated into existing libraries as a modular addition. SIVA DB provides a standard way to archive an MOEA's population and the metadata associated with each population member. Such metadata can include, e.g., the parameters and state of the MOEA and the preferences the decision-maker gives (in the case of interactive MOEAs). The database can store data from multiple runs of any number of MOEAs, and even data from different MOPs. SIVA DB provides easy access to the contained data to analyze the optimization process or create efficient MOEAs. We demonstrate the latter in this paper with experiments.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**.

KEYWORDS

Evolutionary multiobjective optimization, interactive optimization, decision support software

ACM Reference Format:
Bhupinder Singh Saini, Giomara Lárraga, and Kaisa Miettinen. 2023. Using a Database to Support Interactive Multiobjective Optimization, Visualization, and Analysis. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3583133.3596383

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3596383

1 INTRODUCTION AND MOTIVATION

Multiobjective optimization problems (MOPs) are problems where one must simultaneously optimize multiple, often conflicting, objective functions (shortened to objectives). Due to the trade-offs between the objectives, such problems usually do not have a single optimal solution. Instead, a large (or an infinite) number of so-called Pareto optimal solutions exist, each representing different levels of trade-offs among the objectives. Solving MOPs can present a diverse set of challenges. The objectives can be nonlinear or non-differentiable, and the problem can be nonconvex. Besides, one or more objectives of an MOP can be expensive or time-consuming to evaluate. While MOPs may have many Pareto optimal solutions, generally, only one or few are of interest to a decision-maker (DM): a domain expert who wishes to solve the MOP and implement the chosen solution. A DM may need to compare many solutions to find ones that they find interesting. Thus, MOPs with many objectives and Pareto optimal solutions can place a cognitive load on the DM, introducing additional challenges.

Researchers have proposed a large number of so-called multiobjective evolutionary algorithms (MOEAs) to tackle these challenges [7, 9, 12, 19, 23, 27]. MOEAs are population-based heuristics that "evolve" a population of solutions over several generations towards Pareto optimality. Because of the heuristic nature, Pareto optimality cannot usually be guaranteed. Thus, the result is a representation of approximated Pareto optimal solutions.

Various categories of MOEAs exist that tackle specific MOP challenges. Here we give two examples that we refer to in what follows. Surrogate-assisted MOEAs tackle MOPs involving functions that are expensive to evaluate by replacing such functions by cheap-to-evaluate approximations known as surrogate models [6, 21]. Interactive MOEAs take the preferences of the DM into account during the solution process [28]. This allows the MOEA to focus on the region of interest of the DM, making the algorithm more efficient. Moreover, this also lowers the cognitive load on the DM by enabling them to focus on a smaller range of objective values rather than covering the entire set of Pareto optimal solutions at once. Interactive MOEAs allow the DM to learn and change their preferences, enabling them to find the best-suited solution.

While researchers have published many MOEAs, finding their implementations may be challenging. A DM, or even an analyst (an expert in multiobjective optimization), cannot be expected to implement the MOEAs to solve their MOP. To get around the issue of the availability of MOEA implementations, a small number of open-source libraries have been published. Popular libraries of MOEAs

include DEAP [13], DESDEO Framework [25], jMetal/jMetalPy [2, 11], MOEA Framework [14], paGMO/pyGMO [3, 20], Platypus [8] and pymoo [4]. Some of them also provide a graphical user interface [25, 26], making the MOEAs accessible to a larger audience.

As MOEAs are population-based metaheuristics, these libraries deal with a large amount of data while solving MOPs. Using database management systems is a natural way to handle, archive, and analyze this data. However, to the best of our knowledge, no MOEA library makes use of databases to facilitate multiobjective optimization. The DESDEO framework [25] is a minor exception, as it uses a database to add features such as user authentication in its graphical user interface. However, even in this case, the database has a generic design. The database does not provide support for multiobjective optimization in general and interactive or surrogate-assisted multiobjective optimization (for expensive problems) in particular, even though DESDEO implements many interactive and surrogate-assisted MOEAs.

As mentioned earlier, a database management system can help these libraries handle and utilize the data they generate to make the most of the data. The most straightforward implementation of such a database can be used by the libraries to archive the population members. This archive can then later be used for visualization and analysis of solutions [1]. However, additional features can be implemented into the database, independent of the MOEAs or the libraries, to support multiobjective optimization specifically.

For example, in case of interactive MOEAs, the database can store a DM's preferences, and the solutions discovered to satisfy them. The database can then allow the DM, with the help of an analyst, to compare solutions that meet different preferences, making decision-making easier. A database can also allow the DM to save some solutions of interest, which they can return to at a later time.

In this paper, we identify many needs for a database in the context of MOEAs and discuss requirements for a database to facilitate multiobjective optimization. The idea is to demonstrate the added value that a database offers in utilizing the data generated during the evolutionary solution process in a more versatile manner than is currently done. We then design a database for supporting interactive multiobjective optimization, visualization, and analysis and call it SIVA DB. We implement it as an open-source package. Importantly, this package aims not to replace the existing MOEA libraries. Instead, we have designed it to be easy to implement within existing libraries and extend their functionality. Finally, we showcase the usefulness of SIVA DB via two sets of experiments that demonstrate how it can increase the performance of MOEAs and the efficiency in interactive multiobjective optimization.

2 BACKGROUND

We consider MOPs with n decision variables x_j constituting decision (variable) vectors x and k (conflicting) objectives $f_i : S \rightarrow \mathbf{R}$, where the feasible region S is defined by m constraints of the form $g_j(\mathbf{x}) \leq 0$ as well as upper and lower bounds for the variables. A feasible decision vector satisfies all the constraints.

[1]Note that such use cases require additional implementation of, for example, user interfaces and algorithms that assist analysis. A database can act a foundation as well as a common interface of communication for such implementations.

For each feasible decision vector $\mathbf{x}$, we have an *objective vector* $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))^T$. In a corresponding manner, we define a so-called *constraint vector* as $g(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_m(\mathbf{x}))^T$. A feasible decision vector $\mathbf{x}^*$ and the corresponding objective vector $f(\mathbf{x}^*)$ dominate another feasible decision vector $\mathbf{x}$ and the corresponding $f(\mathbf{x})$ if $f_i(\mathbf{x}^*) \leq f_i(\mathbf{x})$ for all $i = 1, \ldots, k$ and $f_j(\mathbf{x}^*) < f_j(\mathbf{x})$ for at least one j. In MOEAs, each solution in a population represents one decision vector. We say that a feasible solution is Pareto optimal if there is no other feasible solution that dominates it. MOEAs can typically only guarantee mutual nondominance, that is, no solution in a final population dominates any of the others. We call them approximated Pareto optimal solutions.

Since Pareto optimal solutions are not comparable without additional information, we need the domain expertise of a DM to identify the most preferred solution for the MOP considered. The DM is not assumed to know details of different methods and, therefore, usually an analyst supports the DM. The analyst knows different methods and takes care of technical matters so that the DM can concentrate on providing preferences and analysing the solutions obtained.

Interactive MOEAs, as mentioned earlier, are one way to utilize DM preferences iteratively. Another class of interactive optimization methods are scalarization functions [24], which are especially popular in the field of multiple criteria decision-making. These functions can use a DM's preferences to convert an MOP into a single objective optimization problem. This "scalarized" problem can then be solved with any appropriate single objective solver. The optimal solution of this scalarized problem is (generally) a Pareto optimal solution of the original MOP, and follows the preferences of the DM.

The most common types of databases are the following: *Relational databases* [18] organize the data into tables with rows and columns. Each table contains a set of records, and each record is made up of fields, which are also called attributes. The columns of a table represent the fields of a record, and the rows represent the records themselves. Relationships between tables are established through common columns, called keys, which serve as links between records. The structure of the tables, including the relationships between tables, forms the so-called schema of the database. The schema of relational databases are required to be defined during initialization. Relational databases are the most commonly used type of database, and examples include MySQL, Oracle, and Microsoft SQL Server.

Non-relational databases [22] are also known as NoSQL databases. They do not use tables to store data. Instead, they use a variety of data models, such as key-value, document, columnar, and graph. Non-relational databases are often used for web applications and big data applications, as they are designed to scale horizontally and can handle large amounts of unstructured data. They are also often used for real-time applications, as they are designed for low latency and high performance. Examples include MongoDB, CouchDB, Cassandra, and Redis.

Among the non-relational databases, document-oriented databases are of interest in this paper. These databases store the data in so-called "collections" of "documents". A collection can be thought of as the equivalent of a table of a relational database. Similarly,

a document can be considered the equivalent of a row in a relational database table. A document stores data of a single entity (for example, the name and email address of a person). A collection groups together similar documents (for example, the multiple documents containing names and email addresses of people can be grouped together to form a Contacts collection). Every document also contains a unique identifier. This identifier can be used to form relationships between documents. These documents can be within the same collection (for example, connecting family members within the Contacts collection), or between documents from different collection (for example, connecting a person in the Contacts collection to perhaps a car in a "Cars" collection to signify ownership). The documents are generally stored as JSON or XML objects.

3 DESIRABLE PROPERTIES FOR THE DATABASE

Different database management systems exhibit different sets of properties, making some more appropriate for usage in MOEA libraries than others. We identify the following desirable properties for database management systems:

- **Schema-less design:** The database management system must be able to support a schema-less design. The reason for this is that we, as the creators of the database, do not have complete knowledge of the MOPs that will be solved by an analyst/DM using the libraries. Nor can we predict how different analyst/DM would solve the associated challenges. For example, we do not know whether surrogate-assisted or interactive MOEAs would be used to solve an MOP. We do not have prior knowledge of the number of objectives or decision variables of the problem. As we lack knowledge about the exact structure of the data that the libraries will save in the database, we cannot enforce a schema on the database. A schema-less database will allow the library to store arbitrary data. This data can include, for example, details about the population members (decision vectors, objective vectors, when the MOEA generated the population members), details about the MOEAs (hyperparameter values), details about the MOPs (MOP names, details about the decision variables and objectives), and preference information of the DM.
  Moreover, the library can make arbitrary connections (i.e., not limited by a schema definition) between the entries of the databases. For example, in the case of interactive MOEAs, the preference information entries in the database can be connected to subsets of population member entries that were generated using that preference information. Designing the exact structure of the entries and connections can be left to the maintainers of the individual libraries. However, we propose a general structure to promote interoperability between various libraries in the next section.
- **Modularity:** Adding the database to any MOEA library should not involve reimplementing the methods already available in the library. Instead, the library's maintainers should be able to add the features provided by the database as modular additions to the MOEAs. This allows the continued usage of the existing MOEA implementations which are database-agnostic. In such cases, the database can act as an archive into which the MOEAs can save solutions and other data. New MOEAs can be implemented with this database as a core feature, creating more efficient MOEAs.
- **Accessibility:** It should be possible to install the database management system in all major desktop operating systems. This allows the analyst/DM who uses the libraries on any operating system to benefit from the inclusion of the database. It should be possible to access the database from major programming languages easily. This allows library maintainers to add the features provided by the database effortlessly.
- **Sharability and mergeability:** Publishing and sharing the database should be easy. Merging databases published by various researchers should also be possible. Sharing such databases will boost open science. A database created by combining such shared databases can enable meta-analysis studies. We expand further on this in Section 4.2.

4 SIVA DB

Now that we have listed desirable properties, we design a database that follows them. SIVA DB has been designed with the needs of interactive (and also surrogate-assisted) MOEAs in mind. In what follows, we introduce SIVA DB and justify the choices made.

Requiring a schema-less design for the database makes it impossible to use relational database management systems. CouchDB and MongoDB are popular alternatives. Both exhibit all the desirable properties mentioned in the previous section. While either is appropriate for managing databases for MOEA libraries, we recommend using MongoDB, as it natively supports many more datatypes. They include "timestamp" and "geospatial" types, which can be helpful in MOPs from certain domains, and a "Javascript" type which can be used to save implementations of the objectives in the JavaScript programming language. Datatypes supported by both CouchDB and MongoDB include various numeric types, arrays, booleans, strings, and key-value pairs. The library can use these datatypes to archive data related to the population members and the data associated with MOEAs.

While MongoDB does not require a schema to be defined, it allows the usage of a schema to validate entries into parts of the database. This validation enables us to propose a general structure for the database, allowing consistency among the various libraries that use this database. The library maintainers are free to expand upon this general structure to fulfill the needs of their MOEAs.

4.1 General structure

As mentioned, MongoDB is a document-oriented database management system that divides the database into collections (instead of tables) which contain documents (in place of rows). With SIVA DB, we support interactive optimization, visualization, and analysis using the database by structuring it into the following collections:

- **Users**: Most MOEA libraries do not support the concept of a "user", who may be an analyst or a DM. However, we consider the Users collection an essential part of the database to support interactive optimization. Each document in this collection stores the detail of a single user. The document must contain these fields: identifier (present by default in all

documents), user name, authentication details, and a list of MOPs solved by the user.
The authentication details will enable the libraries to limit the user's access through the database. The document can contain additional fields, for example, to represent whether the user is an analyst or a DM. This information can help the library present different user interfaces to analysts (who may want to access complex analysis tools) and DMs (who may prefer a simpler user interface). The libraries can use another field to connect different users using their identifiers. This can be helpful to connect an analyst to a DM they are assisting or connect a group of DMs who are solving the same MOP together and giving them shared access to the stored data. Libraries that do not implement the concept of a user can use a "default" user template for anyone using the MOEAs for successful schema validation. We show an example of this collection in Listing 1. The listing includes two documents that contain the information of two users. Each document contains compulsory (i.e., schema-validated) fields and optional fields (that are not validated).

```
[{"_id": "ID_ALICE",
  "name": "Alice",
  "auth": {*****},
  "problem_ids":["ID_DTLZ1", "ID_DTLZ3"],
  "role":"analyst",
  "groupMembers":["ID_BOB"]},

  {"_id": "ID_BOB",
  "name": "Bob",
  "auth": {*****},
  "problem_ids":["ID_DTLZ1"],
  "role":"DM",
  "groupMembers":["ID_ALICE"]}]
```

Listing 1: An example of the Users collection, containing two documents

- **Problems**: Each document in the Problems collection contains information of the formulation of an MOP and must include the following schema-validated fields: identifier (used in the Users collection to connect problems to users), MOP name, list of connected collections (more details in the following items in the list), and the so-called objects for decision variables, objectives, and constraints. The decision variables object must contain the names, type (for example, real-valued or integer-valued), and bounds of the decision variables.
  The objective object must contain names of the objectives, as well as information regarding whether it is to be minimized or maximized. The constraint object may contain the names of the constraints (if any). The documents in this collection can also include additional fields. For example, suppose the problem formulation contains analytical functions (as opposed to black-box simulations). In that case, the formulation can be stored in the database as JavaScript functions (requires Javascript) or MathJSON objects[2] (can be parsed in any programming language). Alternatively, if surrogate-assisted MOEAs are used, the data for training the surrogate models can be put into the database as a new collection. The library can save the collection's name in the related document in the Problems collection.
  We show an example of this collection in Listing 2. Note that the identifier "_id" is generally a randomly generated string. Thus, for example, multiple problems with the same name (with a different number of decision variables or objectives) can exist in the collection. In the listing, we use "ID_DTLZ1" as the identifier for readability.

```
[{"_id": "ID_DTLZ1",
  "name": "DTLZ1",
  "collections": {
    "decisionVectors": "decisionVectors_DTLZ1",
    "objectiveVectors": "objectiveVectors_DTLZ1"},
  "decisionVariables": {"names": ["x1", ... , "x10"],
                        "types": ["real", ... , "real"],
                        "bounds": [[0, 1], ... , [0, 1]]},
  "objectives":{"names": ["f1", "f2", "f3"],
    "maximized": [false, false, false]}}
```

Listing 2: An example of the Problems collection, showing one document.

- **MOEAs**: A document in the MOEAs collection contains details of an MOEA implemented within the library. This information includes the name of the MOEA and a brief description of the capabilities of the MOEA (for example, the ability to handle constraints or whether the MOEA is surrogate-assisted or not). The library can use this information and the MOP document in the Problems collection to suggest MOEAs to the user. We show an example of this collection in Listing 3. This collection can contain multiple documents for the same MOEA if a user has run the MOEA multiple times (for example, with different hyperparameter settings). In such cases, each document will have a different identifier.

```
[{"_id": "ID_RVEA",
  "name": "RVEA",
  "hyperparameters": {"totalGenerations": 1000,
                      "populationSize":100, ...},
  "additionalDetails": {"surrogateAssisted": False,
                        "handlesConsts": True, ...}}]
```

Listing 3: An example of the MOEAs collection, containing one document

- **Collections for decision vectors**: These collections are not initially part of the database. Instead, the library creates a new collection each time a user solves a new MOP with an MOEA for the first time. As the name of the collection (which

[2] https://cortexjs.io/math-json/

the library can create randomly) is not known in advance, the library must store the generated name in the relevant document in the Problems collection. Each document in this collection stores information about a single population member and contains five compulsory fields: decision vectors (to store the decision variable values) [3], a metadata object, a field to hold the identifier of the user running the MOEA, and two fields to hold the identifiers of an objective vector and a constraint vector document (more details in the next point in the list). The metadata object must store the state of the MOEA when the population member is evaluated. This information includes the generation number and the exact values of the MOEA hyperparameters. We show an example of this collection in Listing 4. The collection shown in this listing is named "decisionVectors_DTLZ1" in the database, which connects it to the collection in Listing 2.

```
[{"_id": "DV123",
  "decisionVectors": [0.31, 0.1, ..., 0.05],
  "metadata" : {"moeaID": "ID_RVEA",
                "generation": 25,},
  "userID": ["ID_ALICE"],
  "objectiveVector": "OV127",
  "constraintVector": ""}]
```

Listing 4: An example of a collection storing decision vectors, containing one document.

- **Collections for objective vectors**: Similar to the decision vector collections, the library creates collections for objective vectors when needed and stores the name in the associated MOP document. Each document in this collection contains two compulsory fields: a vector for storing the component of the objective vector and a list of identifiers from documents in the related decision vector collection. Note that a single objective vector document may connect to multiple decision vector documents. Separating the objective vectors into an independent collection instead of adding them directly to the decision vector collection saves space by avoiding repetition. The document can contain additional information, such as whether an objective vector is nondominated.
  The library can also apply clustering algorithms on the objective vectors to identify clusters and add this information to the documents in this collection. The library may also create a separate similar collection, if a surrogate-assisted MOEA is used to store the predicted objective vectors. The documents in such collections should contain metadata regarding the surrogate modeling algorithm used for the prediction in addition to the information mentioned earlier. Moreover, the documents can also store the uncertainty of the prediction if the surrogate modeling algorithm provides such information. We show an example of this collection in Listing 5. The collection shown in this listing is named "objectiveVectors_DTLZ1" in the database, which connects it to the collection in Listing 2.

```
[{"_id": "OV127",
  "objectiveVectors": [5.1, 25, 3],
  "decisionVectors": ["DV123"],
  "clusterID": 5,
  "nondominated": True}]
```

Listing 5: An example of a collection storing objective vectors, containing one document.

- **Collections for constraint vectors**: The structure for constraint vectors collections is identical to that of objective vectors. However, these are only created if the MOP has constraints.
- **Additional collections**: The library can add further collections to store any information deemed useful by the library's maintainer. For example, the library may create a collection to store performance indicator values (such as the hypervolume indicator [29]). A document in such a collection may include the following fields: identifiers connecting to documents in the Problems and MOEAs collection, fields for different indicator values, and a field for the generation number. Using this collection, researchers can study the performance behavior of various MOEAs on different MOPs.

Note that many fields in the database need to be updated regularly. For example, the "nondominated" field in Listing 5 can change from true to false if additional solutions are discovered that dominate the objective vector in that document. Other examples include the values of some performance indicators and clustering information. Keeping this data up to date should not slow down the MOEA generating the solutions. We propose that the database and the MOEA be run on two different processes in parallel. Multiple processes can connect to MongoDB databases, which SIVA DB is based on, simultaneously without blocking any other connected processes. We can have different processes running to keep SIVA DB up-to-date, such as a process to keep checking the nondomination of objective vectors. Similar ideas have been proposed, for example, in [1].

4.2 Benefits of using SIVA DB

We have implemented SIVA DB using MongoDB in the Python programming language as the open-source package [Hidden] in a library-agnostic manner. This implementation provides the maintainers of MOEA libraries (at least those implemented in the same programming language) with an easy way to swiftly incorporate the database into their libraries to gain several benefits. The range of these benefits depends on how deeply the maintainer chooses to incorporate the SIVA DB into their library. Using SIVA DB can support the following:

- **Interactive multiobjective optimization:** SIVA DB makes it easier for library maintainers to create graphical user interfaces for DMs to apply interactive methods by providing a standard method of communication between the backend (the MOEAs) and the front end (the user interface). DMs

[3]Note that if the decision variables of the MOP are represented not as vectors but as a different data structure, we can use an appropriate MongoDB datatype. As a fallback, we can serialize the decision variable values and store the raw data in SIVA DB.

can benefit significantly from the features made possible via SIVA DB. The ability to save solutions (along with the associated preference) can enable the DM to see previously found solutions, making comparing and decision-making easier. These solutions can either be from a single MOEA or a group of MOEAs used by the DM.

Separating the user interface from the backend also allows the library maintainers to run multiple MOEAs at once, increasing the likelihood of finding acceptable solutions for the DM while keeping the interface simple and intuitive. The library can also use SIVA DB to run the backend on a more powerful computer or a remote server while serving the user interface to the DM on a desktop computer [15]. Running time-consuming optimization on powerful remote computers saves the DM's time while letting them provide preferences and analyze the results quickly from their desktop computers.

Finally, keeping an archive of solutions can enable the creation of more efficient interactive methods, for example, by finding the best solutions in the database to use to "warm start" an MOEA based on the preferences of a DM. We test this idea in Section 5.

- **Visualization:** SIVA DB stores not only the solutions found by the MOEA but also a vast amount of metadata associated with each solution. This can include clustering information, associated with the DM's preferences (in case of interactive optimization), and uncertainty information (in case of surrogate-assisted optimization). An analyst can use this information to create insightful visualizations for the DM, aiding decision-making.
- **Analysis:** As mentioned earlier, SIVA DB can help a DM and an analyst analyze solutions by enabling features such as saving solutions that are of DM's interest, storing metadata related to the solutions, and assisting in interactive optimization. SIVA DB can also help researchers analyze the performance of MOEAs. SIVA DB is designed to be easy to share online. Researchers who create new MOEAs or test already published MOEAs can save not only the performance of the final population of the MOEA but the performance of all generations of the population. They can also save additional information, such as performance indicator measurements, into the database. As we have designed SIVA DB to be mergeable, we can merge databases shared by different researchers into an expanding corpus of MOEA performance. Such a corpus can enable meta-analyses and studies of MOEAs that are currently infeasible to perform.
- **Advanced cases which require additional research:** SIVA DB is designed to be future-proof and can easily be updated to support currently unsupported cases. One such case of interest is group decision-making, where a group of DMs (whose interests may not be aligned) must collectively find acceptable solutions to an MOP. SIVA DB can form the backbone around which new MOEAs can be designed to tackle group decision-making challenges. Another case of interest is the automatic selection of MOEAs to solve MOPs. Using the aforementioned corpus of MOEA performance knowledge, we can design predictive algorithms to automatically select the best MOEAs based on the properties of the MOP. Another advanced case is to use SIVA DB is a channel of communication between various MOEA libraries. This can make it easier for a library to use MOEAs implemented in other libraries, making a large number of MOEAs accessible to researchers and analysts.

5 INITIAL EXPERIMENTS

In this section, we evaluate two rudimentary use cases of SIVA DB using experiments with benchmarking MOPs from the DTLZ [10] family of test problems. In the first set of experiments, we use SIVA DB as an archiving tool with an MOEA and show how we can improve the performance of any MOEA with negligible additional computational cost. In the second set of experiments, we demonstrate how SIVA DB can make interactive optimization faster and more efficient, thereby saving the DM's time and resources. We achieve these gains in efficiency by using the database, which contains the entire history of the evolution process, to find the best set of decision vectors (with known values for the objectives) to form a new population whenever the DM changes their preference. This population is composed of individuals that conform to the new preferences the best and thus making the convergence faster. Note that these experiments demonstrate the benefits of SIVA DB that rely solely on the SIVA DB implementation. Other benefits enabled by SIVA DB require the implementation of additional software (for example, a user interface that can use SIVA DB to provide visualization and analysis benefits) and are thus beyond the scope of this study. The codes for all experiments conducted in this study are available at https://github.com/industrial-optimization-group.

5.1 Using SIVA DB as an archive

In this experiment, we will use SIVA DB as an archive for RVEA [5], a popular MOEA that generalizes well to many objectives. We use SIVA DB only as a modular addition to RVEA without making any changes to the algorithm. We use the hypervolume indicator [29] to test the performance of RVEA on DTLZ{1-4} problems with and without SIVA DB. To test the performance without SIVA DB, we use the final population attained by RVEA to calculate the hypervolume. To test the performance with SIVA DB, all individuals (saved into SIVA DB every generation) in the archive that are non-dominated are used to calculate the hypervolume. We run the algorithm on 3, 6, and 10 objective versions of all four DTLZ problems considered. We set the number of decision variables to $10 + k - 1$, where k is the number of objectives. In all cases, we use the nadir point of the MOP (an objective vector with all components equal to 0.5 in DTLZ1 and 1 in DTLZ{2-4}) as the reference point to calculate hypervolume. Except for the population size, we use the default hyperparameter values for RVEA as suggested in [5]. We set the population size to 100 in all MOPs, which is lower than the recommended values (105 for three objectives and 275 for ten objectives). The recommended values led to much longer computation times and only a minor increase in the resultant hypervolume.

We ran RVEA on each MOP five times and recorded the hypervolume values at the end of one thousand generations with and without SIVA DB. We present the median and standard deviation of the hypervolume values in Table 1. The better hypervolume

values are highlighted in bold. RVEA with SIVA DB achieved better hypervolume values for every single MOP. Such performance is not unexpected, as the archive in SIVA DB contains the final population of the RVEA as well. Thus, at worst, RVEA with SIVA DB will be at least as good as RVEA without SIVA DB. The better performance with SIVA DB is due to the non-dominated individuals that were evaluated before the final population but were then removed in favor of individuals that performed better in the selection step of RVEA.

Problem	Number of objectives	Hypervolume without SIVA DB	Hypervolume with SIVA DB
DTLZ1	3	0.0980 (0.0005)	**0.0990 (0.0007)**
	6	0.0149 (0.0040)	**0.0152 (0.0030)**
	10	0.00096 (10^-6)	**0.00097 (10^-6)**
DTLZ2	3	0.413 (10^-4)	**0.426 (0.0021)**
	6	0.661 (10^-4)	**0.724 (0.0027)**
	10	0.838 (10^-1)	**0.870 (0.0018)**
DTLZ3	3	0.391 (0.01)	**0.395 (0.01)**
	6	0.611 (0.03)	**0.619 (0.03)**
	10	0.813 (0.009)	**0.825 (0.007)**
DTLZ4	3	0.412 (10^-4)	**0.428 (0.001)**
	6	0.66 (10^-4)	**0.72 (0.003)**
	10	0.839 (10^-4)	**0.873 (0.001)**

Table 1: Medians and standard deviations (in parentheses) of hypervolume values achieved by RVEA with and without SIVA DB.

In many cases, such as DTLZ3 with three objectives, the hypervolume values attained with and without SIVA DB are very similar. Thus, in such cases, the final population of RVEA and the archive of all non-dominated solutions discovered by RVEA have a similar performance. In the tests we conducted, the final population had converged close to the Pareto front of the MOP and was well distributed. Any additional individuals introduced by the archive thus only increased the hypervolume value by a small amount. However, the final population only contained close to a hundred individuals (the population size in RVEA can be smaller than the number set by the hyperparameter). However, the archive had stored thousands of non-dominated objective vectors for each MOP, attaining additional diversity. This diversity can allow a DM to choose a much more fine-tuned solution in real-life MOPs. In our implementation, SIVA DB runs parallel to the MOEAs that use it. Thus, the additional computational cost of storing individuals and finding non-dominated individuals does not slow down the MOEA. Therefore, using SIVA DB as a simple archive, even without making any changes to the associated MOEA, can lead to immediate benefits at negligible additional costs.

5.2 Using SIVA DB to increase efficiency in interactive optimization

In this experiment, we will use an artificial DM to generate random preferences for MOPs and conduct interactive optimization. As with the previous experiment, DTLZ{1-4} problems are used with three, six, and ten objectives. The artificial DM generates three sets of preferences for each MOP as aspiration levels (values for the objectives the DM aspires to attain), leading to three iterations in the interactive optimization process. The preference is generated close to the line connecting the ideal and nadir points in the first iteration. The artificial DM generates the second preference by making a large random change to the first preference. This behavior simulates a real DM exploring to check what objective values are attainable. The artificial DM generates the third preference by making a small random change to the second preference. This behavior simulates a real DM fine-tuning their preference to find the best solution. Readers can find the exact details of the artificial DM in the linked repository.

We find the solution that best represents the artificial DM's aspiration levels in each iteration by first converting the MOP into a single objective optimization problem. We use the augmented STOM scalarization function, which utilizes aspiration levels to form the single objective problem. We then solve this problem with CMA-ES [17], a state-of-the-art algorithm for continuous black-box optimization, as implemented in [16], using the default hyperparameter values. CMA-ES requires an initial solution and generates the first population in its vicinity. In the first iteration, i.e., before any individuals are evaluated, we choose the initial decision vector to be the average of the decision variables' lower and upper bounds. We refer to this strategy of choosing the initial decision vector as a "cold start".

To test the utility of SIVA DB, we run two versions of CMA-ES in the second and third iterations. The first version does not use SIVA DB, and cold starts in the second and third iterations. The second version uses SIVA DB and looks through the database to find the best solution to use as an initial solution for the second and third iterations, thus conducting a "warm start". To find this solution, we find the objective vector that minimizes the new single objective optimization problem formed using the augmented STOM function and the new preference provided by the artificial DM. We then use the decision vector connected to the objective vector in SIVA DB as the initial solution. If multiple decision vectors are connected to the best objective vector, we choose a random decision vector from that set. Note that we use the same aspiration levels in each iteration when comparing the cold and warm starts for a fair comparison. However, we repeat the tests five times for each MOP, and the artificial DM randomly generates a different set of three aspiration levels each time.

Problem		DTLZ1			DTLZ2			DTLZ3			DTLZ4		
Number of objectives		3	6	10	3	6	10	3	6	10	3	6	10
Iteration 1	Cold Start	4896 (473)	12433 (1890)	13549 (6689)	2135 (130)	4357 (447)	7232 (1017)	4918 (676)	13657 (1462)	15109 (6885)	760 (15)	949 (48)	1177 (75)
Iteration 2	Cold Start	5215 (186)	10921 (4110)	12325 (2877)	2311 (157)	5101 (977)	9205 (2814)	6293 (667)	6673 (5449)	20605 (5094)	**694 (104)**	1177 (149)	**1129 (162)**
	Warm Start	**3279 (398)**	**5509 (795)**	**10057 (8210)**	**1871 (232)**	**4105 (3067)**	**4633 (902)**	**3081 (847)**	**3481 (1194)**	**8233 (1922)**	804 (89)	**1141 (155)**	1237 (107)
Iteration 3	Cold Start	4742 (872)	13537 (4143)	14245 (4226)	2399 (392)	5977 (1199)	6001 (2282)	5853 (1225)	7489 (2841)	16729 (5580)	804 (127)	1105 (195)	1237 (50)
	Warm Start	**3642 (691)**	**5259 (904)**	**7189 (833)**	**1981 (126)**	**1199 (558)**	**4033 (1263)**	**3180 (348)**	**4801 (1264)**	**8413 (2596)**	**705 (147)**	**1093 (218)**	**1165 (108)**

Table 2: Medians and standard deviations (in parentheses) of the number of objective function evaluations conducted by CMA-ES with and without a warm start enabled by SIVA DB to solve MOPs with preferences from an artificial decision-maker.

In Table 2, we report the median and standard deviations in the total number of objective function evaluations conducted by CMA-ES in each iteration for each MOP, using the cold and warm start strategies. We highlight the strategy that required a lower median number of objective function evaluations in each iteration for each MOP by presenting the number in bold. As can be seen, the warm start strategy (which can only be used in the second and third iterations) required a lower median number of objective function evaluations in almost every MOP. In most MOPs, the number of objective function evaluations conducted by CMA-ES using the cold start strategy does not vary much in consecutive iterations. For example, in DTLZ1 with three objectives, CMA-ES using the cold start strategy took 4896, 5215, and 4742 (median) objective function evaluations. The warm start strategy took significantly fewer (median) objective function evaluations for the same problem. In some cases, such as the DTLZ3 problem with ten objectives, the cold start strategy took double the number of objective function evaluations as the warm start strategy enabled by SIVA DB.

The DTLZ4 problem is the only problem where the warm start strategy seemingly fails. However, the numbers of objective function evaluations in all DTLZ4 problems (and across cold and warm starts) are very low compared to DTLZ{1-3} problems. In our testing, this was because CMA-ES reached its termination criteria before converging to the true optima of the single objective problem. This behavior is not unexpected. The DTLZ4 MOP is designed so that it is difficult for MOEAs to find diverse solutions across its Pareto front [10]. In our case, as the artificial DM generates aspiration levels randomly, the resulting single objective problem created using the augmented STOM function is challenging to optimize. Neither a cold nor a warm start could help CMA-ES achieve convergence on the DTLZ4 problems.

In DTLZ{1-3} problems, SIVA DB helped us conduct interactive optimization using much fewer objective function evaluations using the warm start strategy. In real-life MOPs with real DMs, this would mean that the DM has to spend less time between providing their preferences and being able to analyze the resulting solutions. This increase in efficiency is especially beneficial in MOPs with expensive objectives. The computational cost of archiving the individuals each generation and finding the best decision vector from the set of decision vectors in SIVA DB to enable the warm start is insignificant.

6 CONCLUSIONS AND FUTURE PROSPECTS

In this paper, we have proposed incorporating databases into popular libraries of MOEAs to support interactive multiobjective optimization, visualization, and analysis. We have identified the requirements such a database should meet and the needs specific to multiobjective optimization that it should fulfill. We implemented SIVA DB in an open-source package in Python and documented the conventions it follows. Lastly, we showed through multiple experiments that SIVA DB can help increase the performance and efficiency of MOEAs (especially in the case of interactive optimization) while requiring minimal changes to the existing MOEAs and low computational costs.

The ideas presented in this paper have only glimpsed into the possibilities enabled by SIVA DB. As mentioned earlier, further research into using databases with MOEA libraries can help us tackle exciting challenges, including group decision-making and the automatic selection of the best MOEAs to solve MOPs. It enables open science by assisting researchers in sharing data regarding MOEAs and MOPs and merging and utilizing data shared by others. It can also help us unify the currently fragmented landscape of MOEA libraries and make them cross-compatible. This will lay the foundations of a one-stop software containing a massive compendium of MOEAs presented to researchers, analysts, and DMs alike in a user-friendly package.

ACKNOWLEDGMENTS

This research was partly funded by the Academy of Finland (grant 322221). The research is related to the thematic research area Decision Analytics utilizing Causal Models and Multiobjective Optimization (DEMO), jyu.fi/demo, at the University of Jyvaskyla.

REFERENCES

[1] B. Afsar, D. Podkopaev, and K. Miettinen. 2020. Data-driven Interactive Multiobjective Optimization: Challenges and a Generic Multi-agent Architecture. *Procedia Computer Science* 176 (2020), 281–290.

[2] A. Benitez-Hidalgo, A. J. Nebro, J. Garcia-Nieto, I. Oregi, and J. Del Ser. 2019. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation* 51 (2019), article 100598.

[3] F. Biscani, D. Izzo, and C. H. Yam. 2010. A global optimisation toolbox for massively parallel engineering optimisation. *arXiv:1004.3824* (2010).

[4] J. Blank and K. Deb. 2020. Pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[5] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff. 2016. A reference vector guided evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation* 20, 5 (2016), 773–791.

[6] T. Chugh, K. Sindhya, J. Hakanen, and K. Miettinen. 2019. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing* 23, 9 (2019), 3137–3166.

[7] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen. 2007. *Evolutionary algorithms for solving multi-objective problems.* Springer New York.

[8] Hadka. D. [n. d.]. Platypus: Multiobjective Optimization in Python. https://platypus.readthedocs.io. https://platypus.readthedocs.io Accessed February 9th, 2023.

[9] K. Deb. 2001. *Multi-objective optimization using evolutionary algorithms.* Wiley UK, Chichester.

[10] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. 2002. Scalable Multi-Objective Optimization Test Problems. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC 2002).* IEEE, 825–830.

[11] J. J. Durillo and A. J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760–771.

[12] J. G. Falcón-Cardona and C. A. C. Coello. 2020. Indicator-based multi-objective evolutionary algorithms: A comprehensive survey. *Comput. Surveys* 53, 2 (2020), 1–35.

[13] F.-A. Fortin, F.-M De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[14] D. Hadka. [n. d.]. MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization. http://moeaframework.org/. http://moeaframework.org/ Accessed February 9th, 2023.

[15] J. Hakanen, S. Radoš, G. Misitano, B. S. Saini, K. Miettinen, and K. Matković. 2022. Interactivized: Visual Interaction for Better Decisions With Interactive Multiobjective Optimization. *IEEE Access* 10 (2022), 33661–33678.

[16] N. Hansen, Y. Akimoto, and P. Baudis. 2019. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634.

[17] N. Hansen and A. Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

[18] M. J. Hernandez. 2013. *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design.* Addison-Wesley Professional.

[19] H. Ishibuchi, N. Tsukamoto, and Y. Nojima. 2008. Evolutionary many-objective optimization: A short review. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence).* 2419–2426.

[20] D. Izzo and F. Biscani. [n. d.]. PyGMO: Python Parallel Global Multiobjective Optimizer. https://esa.github.io/pygmo. https://esa.github.io/pygmo Accessed February 9th, 2023.

[21] Y. Jin, H. Wang, and C. Sun. 2021. *Data-Driven Evolutionary Optimization.* Springer.

[22] K. Kaur and R. Rani. 2013. Modeling and querying data in NoSQL databases. In *2013 IEEE International Conference on Big Data.* 1–7.

[23] K. Li, R. Wang, T. Zhang, and H. Ishibuchi. 2018. Evolutionary Many-Objective Optimization: A Comparative Study of the State-of-the-Art. *IEEE Access* 6 (2018), 26194–26214.

[24] K. Miettinen. 1999. *Nonlinear Multiobjective Optimization.* Kluwer Academic Publishers, Boston.

[25] G. Misitano, B. S. Saini, B. Afsar, B. Shavazipour, and K. Miettinen. 2021. DESDEO: The modular and open source framework for interactive multiobjective optimization. *IEEE Access* 9 (2021), 148277–148295.

[26] Y. Tian, R. Cheng, X. Zhang, and Y. Jin. 2017. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. *IEEE Computational Intelligence Magazine* 12, 4 (2017), 73–87.

[27] A. Trivedi, D. Srinivasan, K. Sanyal, and A. Ghosh. 2016. A survey of multiobjective evolutionary algorithms based on decomposition. *IEEE Transactions on Evolutionary Computation* 21, 3 (2016), 440–462.

[28] B. Xin, L. Chen, J. Chen, H. Ishibuchi, K. Hirota, and B. Liu. 2018. Interactive Multiobjective Optimization: A Review of the State-of-the-Art. *IEEE Access* 6 (2018), 41256–41279.

[29] E. Zitzler and L. Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (1999), 257–271.