LisBON: A framework for parallelisation and hybridisation of optimisation algorithms

C. Dürr, T. Fühner, P. N. Suganthan

Abstract—This paper introduces LisBON, a novel framework for distributed, hybrid optimisation algorithms. LisBON aims at simplifying the development of memetic algorithms—a combination of heuristic, population-based search approaches with local optimisers. Moreover, LisBON’s design allows for an integration of virtually any optimisation algorithm. It could hence be used to implement a large variety of different hybrid approaches, multiple-restart methods in local search routines, and multiple populations and meta-evolution in evolutionary algorithms. With LisBON, it is not only possible to distribute optimisers onto different computing nodes, but also the concurrent evaluation of merit functions can be defined in a straightforward manner.

In this paper, we present the design of LisBON and its key components. Furthermore, as an example, the steps required to develop a memetic algorithm are explained. It is shown that the obtained hybrid method is able to outperform the underlying genetic algorithm in terms of convergence speed on an established benchmark function (Griewangk).

I. INTRODUCTION

In real-world optimisation problems, function evaluations (of the fitness or target function) are often expensive in terms of computation time. Consequently, there is a constant need for new or improved optimisation algorithms that reduce the total number of function evaluations needed for obtaining optima with a required precision. As an additional requirement, it should be possible to conduct computations in parallel, ideally using available computing facilities at their full potential.

If the optimisation problem is convex in some form (e.g., pseudo- or quasi-convex), and the gradients are known, it can readily be solved with available gradient-based methods from mathematical optimisation, such as the SQP algorithm [1]. In real-world problems, however, the target function is often assumed to be a blackbox function, so that gradients can only be calculated numerically. Additionally, in general the convexity assumption does not hold. For this reason, it should be possible to conduct computations in parallel, ideally using available computing facilities at their full potential.

When hybridising several search methods to form a single MA, a good balance needs to be found between global and local search [7]. When defining this balance, one needs to take into account the properties of the constituent algorithms. We will illustrate this fact: On one hand, a GA depends on diversity of the population for advancement, and it is important not to degrade this diversity by triggering local search too frequently. On the other hand, a GA does not produce an entirely new set of individuals every generation. Thus, only a sparse application of local search is actually...
efficient, since otherwise previously found local (sub-)optima are used as start points for a new local search step (which is especially counterproductive for gradient-based approaches).

In [8], the authors used the mechanisms of trigger, selection and optimisation radius to fine-tune the balance between global and local search. A trigger function determines at which GA generation step a local search is to be performed. If local search is scheduled, a selection function chooses individual solution candidates as its start points. Finally, an optimisation radius restricts the area for local search.

While experiments with a GA/SQP hybrid yielded considerable improvement compared to a pure GA [8], later tests indicated that the use of simpler search methods might be more appropriate for box-constrained problems. In early tests, the combination of a GA with the Simplex algorithm as developed by Nelder and Mead [9] proved very successful on some test functions (cf. also Sect. IV).

Thus, in order to be able to explore more hybridisation options, a flexible framework was sought after. In this framework, algorithms should be easily exchanged, new algorithms seamlessly integrated, and the interaction patterns between the algorithms adjusted flexibly. The framework should also be flexible enough to allow for the implementation of a self-adapting MA, such as in [10].

As mentioned above, when the computing resources are available, Memetic Algorithms need to be parallelisable in order to be competitive. While a stand-alone GA can be parallelised in a straightforward manner for faster calculation, combinations of two or more different algorithms might complicate the required set-up significantly, especially if one or more of the integrated algorithms were not originally implemented for parallel computing.

LisBON was designed to address all of these challenges. It allows for a straightforward integration of any optimisation algorithm available as a Python, C, or Fortran module, by only adding a few lines of code. The framework also allows for a finely tuneable distribution of tasks among computers within a heterogeneous network. Moreover, if the code is not – or not sufficiently – parallelisable, a sufficient number of copies of this program can be started in order to fully utilise the potential of the available number of processors. LisBON makes use of Python standard libraries, and is thus platform-independent.

LisBON was designed to address all of these challenges. It allows for a straightforward integration of any optimisation algorithm available as a Python, C, or Fortran module, by only adding a few lines of code. The framework also allows for a finely tuneable distribution of tasks among computers within a heterogeneous network. Moreover, if the code is not – or not sufficiently – parallelisable, a sufficient number of copies of this program can be started in order to fully utilise the potential of the available number of processors. LisBON makes use of Python standard libraries, and is thus platform-independent.

LisBON was designed to address all of these challenges. It allows for a straightforward integration of any optimisation algorithm available as a Python, C, or Fortran module, by only adding a few lines of code. The framework also allows for a finely tuneable distribution of tasks among computers within a heterogeneous network. Moreover, if the code is not – or not sufficiently – parallelisable, a sufficient number of copies of this program can be started in order to fully utilise the potential of the available number of processors. LisBON makes use of Python standard libraries, and is thus platform-independent.

LisBON was designed to address all of these challenges. It allows for a straightforward integration of any optimisation algorithm available as a Python, C, or Fortran module, by only adding a few lines of code. The framework also allows for a finely tuneable distribution of tasks among computers within a heterogeneous network. Moreover, if the code is not – or not sufficiently – parallelisable, a sufficient number of copies of this program can be started in order to fully utilise the potential of the available number of processors. LisBON makes use of Python standard libraries, and is thus platform-independent.

This paper is organised as follows: Sect. II gives an overview of the underlying principles of this framework. Sect. III introduces the framework and its key concepts, namely the Broker-Worker task distribution and the concept of rules. Sect. IV demonstrates an exemplary Memetic Algorithm that is a hybridisation of a GA with a Simplex algorithm. Finally, Sect. V draws conclusions and gives an outlook on future work.

II. DESIGN ASPECTS

In this section, the key design aspects of LisBON are presented. These form the foundation for the framework components presented in Sect. III. In the presented version, LisBON is intended for use in high-reliability networks, where node failures can be ignored and network errors are rare. It has been successfully tested on 10 nodes with up to 15 processors. The broker-worker system (cf. Sect. III-A) obviously limits the scalability to smaller scale networks. Future development will enhance the framework for usage in true distributed systems or large-scale grid computing.

A. Distributed framework

LisBON is a high-level framework for distributed programming. In principle, it can be compared to other frameworks for parallel programming, such as MPI [11] and PVM [12]. These two frameworks, implemented in several programming languages, offer low-level support for parallelisation: Users must take care of the specific tasks submitted to different processes, and must tailor their program to the framework’s requirements.

In contrast to this type of framework, DisPyTE [13] can be classified as a mid-level framework: it supports execution of arbitrary Python scripts on nodes, taking full control of task distribution. Users need only specify a task stack that needs to be executed.

In contrast, LisBON is a high-level framework that is specialised for parallel hybrid (or arbitrary combinations of) optimisation algorithms. After adaptation of packages for the use within the framework (requiring but a few lines of code) and specification of the network, users can specify the flow on an algorithm level.

As this framework in its current version can only use TCP/IP connections in order to communicate between the broker and the workers (cf. Sect. III), its usage is only efficient (as compared to a sequential program) when the time spent on communication (order of magnitude: ms) is considerably lower than the time spent on calculations. This is generally the case in real-world problems, where fitness evaluations often take considerably longer than a second. In addition, it also means that the network infrastructure should be reliable and fast.

B. Memetic Algorithm Frameworks

In the area of Memetic Algorithms, there have been very few attempts so far to create a framework that allows the user to freely create and adjust new MAs. In [14], the Java-based framework MAFRA is presented that allows for the construction of MAs on a GA basis. In contrast to that, LisBON exhibits a greater flexibility. It does not require specific classes of search algorithms, but solely depends on functions to start, stop, and advance to the next iteration step in the underlying optimization routines.
In [15], the ParadisEO framework is discussed. Unlike MAFRA, and similar to LisBON, ParadisEO is designed as a parallel computation environment. As ParadisEO depends on an implementation of the search algorithms within its object hierarchy, an adaption of new search routines may become intricate. Furthermore, the balance of global/local search operations, which is a critical issue in Memetic Algorithms [7], [8], is not fully taken into account. LisBON, in contrast, incorporates rule-based trigger and selection mechanisms which allow for a flexible coverage of this problem.

C. Algorithms within the framework

LisBON is intended as an optimisation algorithm framework, and thus makes a few assumptions that are generally met by such algorithms. Firstly, it assumes that an algorithm is separable into discrete steps. For example, in GAs, the generation of a new population can be considered as one step. In the simplest case, an algorithm consists of just a single step, corresponding to a start-stop functionality: For example, the evaluation of the fitness function could be wrapped into such a one-step-algorithm. A local optimiser that is combined with a GA in order to improve selected individuals can be regarded in both ways. On the one hand, it could exhibit only a start-stop functionality, with an internally fixed termination criterion. On the other hand, it could be split up into steps, with the termination criterion evaluated externally, for example by the LisBON broker (cf. Sect. III-B).

The second assumption that is made is that each algorithm operates on a population of individuals. All population-based algorithms obviously fulfill this assumption. Single-point algorithms (such as gradient-based methods) can be defined as operating on a population of a single individual that represents the starting point of the search.

The third assumption is that each algorithm operates on a standard optimisation problem, that is to minimise or maximise one or more objectives with respect to given parameters. The parameters and fitness values can be defined in a very flexible way, since LisBON does not depend on any specific properties they should exhibit. Rules, however, may interpret or manipulate parameters or fitness values; they need to be adapted by the user eventually.

In order to provide a simple example, we now introduce an MA consisting of a GA and a Simplex algorithm. We will assume that both algorithms use the same fitness function. In this work, the Simplex algorithm uses a fixed, internal termination criterion. To LisBON, the Simplex is thus represented as a one-step-algorithm, since the entire local optimisation run is performed as an uninterruptible step. This Simplex algorithm will be given an individual (i.e., candidate solution) as input, and will produce a modified individual (i.e., a possibly improved candidate solution) as output. The MA will then be formulated as follows: The central part will be the GA. Every 20th generation, a random 7.5% of the GA population are passed on to a Simplex worker, and the modified individual replaces its unmodified version within the GA population. This MA is shortly presented in Fig. 1.

For simplicity, we will start by assuming that this algorithm operates in a sequential way.

D. Populations

As will be explained in Sect. III-C, population containers are needed for the correct handling of populations and individuals within the framework. What will be called individuals throughout this paper may be designated as “candidate solutions” in other contexts.

More precisely, MAs that are implemented in LisBON could be described on the macro level by operations on the populations. Basic operations on these populations would, then, include transfer of populations from broker to worker and vice versa, the modification of the population within a worker, and the splitting and re-coalescing of a population. For example, a parallel “mass” fitness evaluation could be described as shown in Alg. 1: The population is split up into sub-populations whose sizes correspond to the input size required by the fitness function (i.e., a single individual). These are then dispatched for fitness evaluations to fitness workers. As soon as they are returned, the individual is re-inserted into the population in its original place. When the fitness of all individuals is evaluated, the population is returned to the GA worker.
Algorithm 1

**POPULATION DISPATCH**

1. GA worker sends $P_1$ to broker for mass fitness evaluation
2. Broker splits $P_1$ up in populations of size 1, $P_1$, $P_2$, ..., $P_n$
3. Broker dispatches $P_i$ to fitness workers for evaluation

**Require:** a population $P_i$ is returned
4. re-insert its individual into $P_1$ at the correct place

**Require:** all $P_i$ have been returned
5. send population back to GA worker

**III. LisBON**

LisBON is a recursive acronym and is short for “LisBON is a Brokered Optimisation Network”. The framework consists of the broker and copies of workers, that each may run on different computers and communicate via TCP/IP. Each worker represents an algorithm (GA, Local Optimiser (LO), fitness evaluation). The interaction between the workers is exclusively managed by the broker, i.e., there is no peer-to-peer style interaction.

LisBON was modeled as a brokered distributed framework with a unique broker, as the flow of the MA needs to be unequivocally defined. For a smaller number of nodes, a centralised model is most efficient; depending on the number of nodes and the node and network architecture, a node dedicated only to broker tasks might be needed. In practice, the broker need not be unique but only synchronised, yet such issues will only be addressed in future versions.

The broker itself is steered by events which in turn are triggered by workers that have completed a step. Based on the occurrence of such an event, the broker will either execute a rule or signal to the worker to continue to execute the next step. Such rules can, for instance, take the shape of “after every fifth generation of the GA, modify the top 20% of the population with a Simplex algorithm”, or “stop the GA worker after the 100th step”.

**A. Workers**

Operational entities such as optimisation algorithms or fitness evaluation routines are encapsulated in “workers”. Copies of worker instances can be executed on several computation nodes, in order to allow for parallel execution of mutually independent tasks. Fig. 2 schematically shows, for instance, how fitness evaluations are calculated in parallel within the framework: The GA worker requests a number of fitness evaluations from the broker by sending a population with uninitialised fitness values. The broker then distributes the fitness evaluations as tasks to the fitness workers, each of which in turn modifies a single individual by setting its fitness value. When all fitness values have been calculated, the broker returns the so modified population to the GA worker.

Local optimisation tasks are distributed in the same way: Each generation, the GA requests instructions on whether to continue its work immediately or wait for an updated population. If an LO is to take place, a set of individuals (the top 10% of the population in our previous example) are distributed to the LO workers in parallel. As soon as all local optimisation tasks are finished, the modified population is returned to the broker.

All workers need to be registered with the broker. Within the broker, they are represented by WorkerProxy objects. Likewise, the broker is mirrored at worker side by a BrokerProxy object, as shown in Fig. 3. These objects provide the worker interfaces at broker side, and pass the tasks on via a communication protocol. For the current version of LisBON, XMLRPC\(^1\) [16] has been used. As a less communication intensive and non-synchronous communication technique, a customised message passing may be used instead in future versions.

**B. Broker**

The broker controls the flow of information between the workers. For instance, it distributes fitness evaluations to fitness workers (cf. Fig. 2). But it is also here that most of the MA’s design is determined.

To continue the example of Fig. 1: After generating a new generation, the GA worker calls the broker via the broker proxy object. The broker then checks the current state of the GA against its set of rules (cf. Sect. III-D) in order to determine what action to take. In our example, if the generation index is not divisible by 20, the broker will let the GA worker resume. Otherwise, it randomly selects 7.5% of the GA population, and dispatches them to the Simplex workers. After each completion of a Simplex worker, the

\(^1\)XMLRPC is short for “XML Remote Procedure Call”. It is used to transparently call a function on a remote machine and return the function result, as if the function had been executed locally.
individual is re-inserted into the broker representation of the GA population. As soon as all selected individuals have been thus modified, they are returned to the GA worker. The GA is then signalled to continue its evolution with this new population.

C. Population Containers

Since LisBON aims at being a flexible framework which allows for an integration of a great variety of optimisation algorithms and implementations, it does not restrict the internal data representation of utilised optimisers. In order to still be able to operate on the same solution candidates, parameters and objective values have to be unified without abandoning the original representation. That is, for solutions that are dispatched to an evaluation or optimisation worker, any algorithm-specific information must be maintained by the LisBON broker. For that, LisBON makes use of population containers which impart the transfer of solutions from one algorithm to another. These containers include representations of the solution candidates and can be used to manage specific information required by utilised optimisation or evaluation algorithms.

Furthermore, a new index is added to an individual object whenever it is transferred to another population container. This index is removed when it is returned to the previous container, so that it can replace its unmodified copy. A worker must operate on the solution representation in the population container. Any modification of parameters or fitness values must be performed in-place.

D. Rules

In Sect. I, we have introduced trigger, selection and optimisation radius mechanisms for controlling the flow of the Memetic Algorithm. This is appropriate for defining GA-centric MAs. However, for the purpose of this framework, a more general approach was to be developed. This is realised by the rules mechanism used in LisBON.

Rules consist of a head and a body. The head defines a boolean condition that determines whether the rule is executed; this condition depends on the id, type and current state of a worker. The body contains the code to be executed when the head evaluates to true. The rules that correspond to the MA model in [8] have been implemented in this first version of LisBON. These rules only transfer populations (or parts thereof) from one worker to another.

These rules that govern the flow of the MA need to be registered with the broker, who checks and executes them. Two of the most common rules are already displayed in Fig. 2: The “loop rule” initiates the dispatch of a population that is to be modified by different workers. Once the modification is complete, the “return borrowed rule” ensures that the population is re-transfered to the original worker.

E. Implementation

The framework has been programmed using the Python programming language and Python standard libraries. Python is an interpreted language, and is thus platform-independent. However, some convenience functions (such as automated starting of workers via ssh) are more readily available under Unix/Linux.

In order to use C/C++ libraries or programs in Python, SWIG [17] has proved to work with the LisBON framework. Fortran modules can be included using f2py [18].

Currently, the internal communication is done via XML-RPC [16], which is implemented as a standard Python library. XMLRPC is a standardised, reliable remote procedure call protocol layered on top of HTTP.

IV. APPLICATION EXAMPLE

The following introductory example demonstrates how LisBON can be employed to implement a distributed MA. As a global search method, a GA is used, whereas local optimisations are performed with a Simplex algorithm. For the sake of a simplified illustration, a well-known GA benchmark function (Griewangk) serves as figure of merit.

In many real-world examples, however, fitness functions may be much more expensive in terms of computation time. We will start by presenting the sequential algorithm in Alg. 2 (“mass fitness evaluation” refers to the evaluation of the fitness for the complete population in one step). For reasons of simplicity, we will assume a simple MA setup: every twentieth generation of the GA is to be locally optimised (starting at generation 40), and 7.5% of the population are randomly selected for LO with a Simplex algorithm. The decision and selection functions related to triggering of the LO and selecting which individuals are to be optimised can take more complex forms (for a tentative venture into this domain, cf. [8]), as long as a balance of global and local search is maintained that is appropriate for the complexity of the fitness function [7].

In LisBON, the re-formulation of the sequential MA into a parallel flow is relatively straight-forward. The division into three separate algorithms is obvious: GA, Simplex and fitness evaluation. First, the existing algorithms need to be
1: Initialise GA and LO parameters
2: Randomly create GA population $P$
3: while termination criterion not met do
4: Generate next generation $P$
5: Evaluate fitness of individuals in $P$
6: shuffle $P = \{I_1, \ldots, I_n\}$
7: if generation index is a multiple of 20 and $\geq 40$ then
8: Select $\hat{P} := \text{select randomly}(P, 0.075 \cdot |P|)$
9: for $I$ in $\hat{P}$ do
10: $I := \text{locally optimise}(I)$
11: end for
12: Re-insert individuals from $\hat{P}$ into $P$
13: end if
14: end while

Algorithm 2

SEQUENTIAL GA/LO HYBRID

re-formulated for parallelisation. This is shown in Fig. 4 for EA-like algorithms. For “closed” algorithms that have only start-stop functionality (such as the Simplex in our example), the “request/receive instructions” part can be omitted.

Second, the flow of the MA needs to be translated to fit the framework. In LisBON, this is done by adding rules to the broker, with appropriate decision and selection functions. For the distribution and return of fitness evaluations, the rules are automatically created. For the “borrowing” of populations among the algorithms (e.g., borrowing of partial GA population to the Simplex algorithm), appropriate rules need to be formulated for the broker.

In the presented example, a “loop rule” has been specified, which triggers a local optimisation step after each 20 generations of the GA. The same rule contains a selection function that randomly chooses 7.5% of the GA population’s solutions. In addition, a “return borrowed rule” is defined. It is responsible for re-inserting solution candidates modified by the Simplex algorithm into the original GA population, appropriate rules need to be formulated for the broker.

In the presented example, a “loop rule” has been specified, which triggers a local optimisation step after each 20 generations of the GA. The same rule contains a selection function that randomly chooses 7.5% of the GA population’s solutions. In addition, a “return borrowed rule” is defined. It is responsible for re-inserting solution candidates modified by the Simplex algorithm into the original GA population, appropriate rules need to be formulated for the broker.

In the presented example, a “loop rule” has been specified, which triggers a local optimisation step after each 20 generations of the GA. The same rule contains a selection function that randomly chooses 7.5% of the GA population’s solutions. In addition, a “return borrowed rule” is defined. It is responsible for re-inserting solution candidates modified by the Simplex algorithm into the original GA population, appropriate rules need to be formulated for the broker.

In the presented example, a “loop rule” has been specified, which triggers a local optimisation step after each 20 generations of the GA. The same rule contains a selection function that randomly chooses 7.5% of the GA population’s solutions. In addition, a “return borrowed rule” is defined. It is responsible for re-inserting solution candidates modified by the Simplex algorithm into the original GA population, appropriate rules need to be formulated for the broker.

We now use this algorithm to optimise the well-known Griewangk function [19], in twenty dimensions (1). This function has been re-formulated as a maximisation problem, for easier usage with the employed GA. The range of the variables is $-600 \leq x_i \leq 600$ for $i = 1, \ldots, 20$. The global optimum is located at $x^* = (0, \ldots, 0)$ with a maximum function value of $f(x^*) = 0$.

$$f(x) := -1 - \frac{1}{4000} \sum_{i=1}^{20} x_i^2 + \prod_{i=1}^{20} \cos \left( \frac{x_i}{\sqrt{i}} \right)$$ (1)

We used a population size of 200 for the GA and executed 10 test runs each for the GA and the GA/Simplex hybrid algorithm. We used 1000 generations for the GA and 750 generations for the GA/Simplex. The GA features restricted tournament selection [20] and Gray coding [21]. The results are shown in Fig. 5. It is evident that for this test function, the MA achieves considerably better results, even approximating the optimum up to a precision of $10^{-6}$ on average after 175,000 function evaluations.
Fig. 5. Experimental results of GA (dashed) and GA/Simplex (solid). Thick lines represent the average over all runs of the maximum function value; thin lines represent the max/min over all runs of the same value, respectively. The y-axis represents the function value; its absolute is equal to the approximation quality $|f(x) - f(x^*)|$. 

V. CONCLUSIONS AND OUTLOOK

In this paper, we have presented LisBON, a novel framework that can be used for creating new hybrid algorithms. As this framework represents both a piece of software and a manner of looking at MAs, related key concepts were introduced first. We then continued by describing LisBON itself in more detail by explaining its structure and components. Finally, we presented a simple application example by hybridising a GA with a Simplex algorithm. We showed that on a well-known highly multimodal test function, the new hybrid clearly outperformed the plain GA.

The presented framework lays the grounds for more thorough practical research work into the field of hybrid optimisation algorithms. On one hand, it permits the hybridisation of any existing optimisation algorithms into new combinations of two or more algorithms. On the other hand, the hybridisation itself is flexibly adjustable, allowing for hybrid algorithms of almost arbitrary shapes.

The application example presented in this paper was appropriate for a demonstration of the framework. However, it shows only a simple, static hybridisation of two algorithms. Much better results could be achieved by using a more dynamic hybridisation and by including more optimisation algorithms, with the choice of algorithms and the shape of the hybridisation tailored to the problem at hand.

LisBON is still under development. Currently, the functions described in this paper are implemented. Some key aspects of parallel programming, however, are not yet realised and are planned for the future versions.

The framework does not yet produce a maximum processor load in all MA settings. This is due to the fact that currently global and local searchers cannot manipulate the same population at the same time. Since the number of function evaluations that LOs need varies heavily, not all nodes in the network are used towards the end of an LO phase. A future version of LisBON will allow for parallel execution of all parts; this means, however, that more than one algorithm will be competing for the limited resource of fitness workers. This has serious implications for the inner logic of the framework, an issue which will need to be addressed.

A future area of research is the development of a self-adaptive MA, that can draw from a pool of global and local optimisers and adapt the interaction between those on a dynamic basis.

Communication within the framework is subject to further improvement. The number of messages and the amount of data passed can and will be reduced in future versions. More tolerance for network and node failures needs to be included. However, it should also be mentioned that currently the framework is not intended for use in wide-area high-failure networks, but is rather targeted at high-speed, closed, reliable networks.

The Broker-Worker scheme allows only for limited scalability due to a “bottleneck” of communication at the Broker – as any centralised scheme. A peer-to-peer system can resolve this communication problem, but increases the difficulty in efficient task handling and distribution.

In a future version, a Graphical User Interface is envisaged, that allows for more user-friendly control and feedback of the framework.

VI. ACKNOWLEDGEMENTS

The authors affiliated with the NTU would like to thank the Agency for Science, Technology and Research in Singapore for its financial support under the grant number 052 101 0020.

REFERENCES