Metaheuristics "In the Large"

Jerry Swan*, Steven Adriaensen, Alexander E. I. Brownlee, Kevin Hammond, Colin G. Johnson, Ahmed Kheiri, Faustyna Krawiec, J. J. Merelo, Leandro L. Minku, Ender Özcan, Gisele L. Pappa, Pablo García-Sánchez, Kenneth Sörensen, Stefan Voß, Markus Wagner, David R. White

Abstract

Following decades of sustained improvement, metaheuristics are one of the great success stories of optimization research. However, in order for research in metaheuristics to avoid fragmentation and a lack of reproducibility, there is a pressing need for stronger scientific and computational infrastructure to support the development, analysis and comparison of new approaches. To this end, we present the vision and progress of the "Metaheuristics 'In the Large' " project. The conceptual uderpinnings of the project are: truly extensible algorithm templates that support reuse without modification, white box problem descriptions that provide generic support for the injection of domain specific knowledge, and remotely accessible frameworks, components and problems that will enhance reproducibility and accelerate the field's progress. We argue that, via principled choice of infrastructure support, the field can pursue a higher level of scientific enquiry. We describe our vision and report on progress, showing how the adoption of common protocols for all metaheuristics can help liberate the potential of the field, easing the exploration of the design space of metaheuristics.

Keywords: Evolutionary Computation, Operational Research, Heuristic design, Heuristic methods, Architecture, Frameworks, Interoperability

1. Introduction

Optimization problems have myriad real world applications [42] and have motivated a wealth of research since before the advent of the digital computer [25]. Recent decades have seen enormous progress in the discipline of metaheuristic optimization[1]. In contrast to *exact* approaches that guarantee optimality, a metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality

*Corresponding author: jerry.swan@york.ac.uk. University of York, York YO10 5DD, UK.

[1]In this article (and in the spirit of Sörensen and Glover, [101]), we reserve the term "metaheuristic' for the generic, cross domain framework and the term "heuristic" for a customization of such a framework to one or more specific domains.

solutions. At each iteration, it manipulates either a complete (or partial) single solution or else a collection of such solutions. The subordinate heuristics may be high or low level procedures, or a simple local search, or just a construction method. The family of metaheuristics includes, but is not limited to: adaptive memory procedures, tabu search, swarm intelligence, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, scatter search, neural networks, simulated annealing, and their hybrids [101, 85]. One of the major advantages of metaheuristics is that they are abstract search methods [96]: the underlying search logic can be applied to any problem which can be decomposed into a few elementary aspects, namely solution representation, solution quality evaluation and some notion of locality. The latter denotes the ability to generate neighboring solutions via a heuristically-informed function of one or more incumbent solutions.

Very broadly speaking, one might distinguish between *classical* OR and metaheuristic approaches with respect to the former's emphasis on *analytic* methods and the latter's emphasis on *empirical* ones [108]. An analytic approach uses problem domain information (and typically also *a priori* human ingenuity) to derive effective algorithms for search components — edge-assembly (EAX) crossover for the Travelling Salesman Problem being one such example [71]. Indeed, the pre-eminent successes of OR often arise directly from a direct match between the solver (e.g. linear programming) and the analytic characteristics of the problem. In contrast, the use of analytic problem characteristics to choose solvers is not part of mainstream metaheuristics. The empirical approach performs configuration tuning (by hand, using statistical design or some Machine Learning technique) to create a metaheuristic biased either offline by a target distribution of problem instances and / or online by the search trajectory.

Despite the significant progress in metaheuristic optimization research, it is increasingly acknowledged within the scientific community that the field suffers from a duplication of effort and siloing of research between papers, groups, and software frameworks. This lack of re-use is evident at both *conceptual* and *implementation* levels:

- Conceptual: an over-reliance on reasoning by metaphor [100] hides commonalities between algorithms, leading to the repeated discovery of the same ideas and heuristics, and widespread duplication of research effort.
- Implementation: despite past efforts in developing software frameworks, there is a tendency to re-implement metaheuristics from scratch, hindering reproducibility and replicability [107, 76, 106].

This duplication of effort limits scientific progress; instead of building a cohesive body of knowledge consisting of robust scientific conclusions, accumulated wisdom in the field is more akin to "folklore": observations over individual algorithms and optimization problems, without a structured underlying narrative. In the words of Pamparà, there is a “throw away” research culture [76]: it is difficult to locate and compare against prior-art, and there is a lack of understanding as to which heuristics work best and in what context. Despite the availability of many software libraries, it is difficult to reuse existing implementations — in particular, it is extremely difficult to combine heuristics from different libraries or incorporate domain-specific knowledge in a general manner, which has hindered the creation of easily testable and deployable metaheuristic pipelines.

While many authors have identified and critiqued a lack of rigor and weak empirical method in the field (e.g. [46, 47, 21]), we are more concerned by the lack of generality of enquiry, and consequently the generality of conclusions. We believe that, instead of examining individual datapoints concerning a particular algorithm implementation and a problem set of an author's choosing, we should be investigating deeper scientific questions, such as:

Q1 Why do our methods work? In particular, how can we assign credit to individual components and eliminate those that do not matter?

Q2 How can we *analytically* arrive at a solution method given a problem description? For example, consider problem reduction: is it *practically useful* to map an optimisation problem of a given type into another? Is there a ubiquitous “SAT-like” problem to which many optimisation problems can usefully be reduced to and solved in practice?

Q3 Is it possible to more fully automate the exploration of the space of metaheuristics, allowing researchers to focus on improving that automation?

To these ends, this paper describes the vision and progress of ‘Metaheuristics in the Large’ (MitL). MitL is a research community initiative (first introduced in Swan et al [107]) that seeks to address the lack of re-use at both conceptual and implementation level. MitL is both a synthesis and extension of existing ideas dispersed throughout the literature, and simultaneously a project producing new software tools and exemplars to show how these problems can be overcome. We draw on many contributions previously made in this direction: hyper-heuristics, constraint programming and the fundamental principles of substitutability of software components. We rely heavily on functional programming constructs to express metaheuristic components in a truly reusable way. In constructing this synthesis, we have exposed gaps in the literature that we are now closing with new contributions: in particular the MitL initiative has introduced a) the “Automated Open Closed Principle” [106], which shows how to express algorithm frameworks as ‘closed’ design spaces which can nonetheless be configured in an open-ended manner via combinatorial assembly, and b) the removal of the domain barrier from hyper-heuristics [108], essential in raising the level of genericity with respect to problem domains. These address Q3, making easier automation of the space of metaheuristics, and hence progress towards answers to questions Q1 and Q2.

This paper provides a survey, progress report and roadmap of our attempts to reduce the fragmentation of metaheuristics research, improve reproducibility, and accelerate progress through infrastructure improvement. We have made several concrete steps forward and can see the road ahead, but there are many problems left to be solved.

2. Contemporary Research Practice in Metaheuristics

One may characterize *researchers* as being broadly concerned with the scientific practice of obtaining concise explanations of empirical observations [86]. In constrast, for *practitioners* (e.g. in industry), the goal is to maximally exploit the information obtained via research, with minimal expert knowledge. We now present in more detail some of the challenges facing metaheuristic research, drawing on previous discussions in the literature. These observations motivate the MitL approach described in Section 3.

2.1. Replication and Reuse

Scientific progress in any discipline requires ready determination of the nature and merits of previous contributions, and the ability to build on the work of others to make further progress. Algorithm descriptions in many papers on metaheuristics are far from precise enough to allow independent re-implementation, and public access to the associated source code is rarely mandated by editors or programme committees. As a consequence, replication studies are very uncommon: a recent such paper, the only replication study in metaheuristics of which the authors are aware, obtained results which were an order of magnitude worse than those originally claimed [102][2].

As noted above, metaheuristics, and in particular Evolutionary Computation research, has developed something of a 'throw away' culture [76, 21], in which a large percentage of researchers neither build upon the research *implementations* of their peers nor create such re-usable software artifacts themselves. This inability to consolidate is in contrast to other research areas that have successfully embraced re-use: for example the SBML standard [3] in systems biology, which allows the researcher to easily create test and deployment pipelines [55]; or the Taverna framework[4] used for workflow construction in a variety of other scientific disciplines. One might wonder why metaheuristics, which enjoy a small and ubiquitous set of abstract components such as acceptance or perturbation, has seen relatively little progress in large-scale re-use. It is possible that the very simplicity of metaheuristics at the component level is in part responsible for this culture of Babel-like proliferation. Metaheuristic researchers or practitioners often choose the creation of *ad hoc* solutions to

[2]It should be emphasized that this problem is not restricted to metaheuristics: a recent study [21] showed that in the computer science papers considered, 34.65% of them were not repeatable and the authors could not conclusively determine repeatability in another 20.87% of cases.

[3]`http://sbml.org`

[4]`https://taverna.incubator.apache.org`

using pre-existing resources, perhaps because implementing baseline versions of (say) Simulated Annealing or Genetic Algorithms from scratch is relatively simple — provided they are not intended for reuse by others.

The last few decades have seen the development of many popular metaheuristic libraries, implemented in a variety of programming languages, some of which feature components that are (in principle) reusable at the framework level. Taking Evolutionary Algorithms as an example, Parejo et al. [82] provide an overview of commonly used libraries along with their performances on benchmarks, including HeuristicLab [5] [114], ECJ [6] [62], FOM [7] [81], Opt4J [8] [60], jMetal [9] [29] and JAMES [10] [26]. The majority of these libraries support component interoperability within their frameworks. However, a component implemented in a specific framework cannot readily be reused within, or hybridized with, another framework. Recognizing this problem, an early attempt [67] sought to achieve interoperability through the use of a common description language based in XML, albeit restricted in focus to evolutionary algorithms. PISA [13] was another early attempt to achieve interoperability *across* frameworks[11]. In PISA, the problem domain component is separated from the metaheuristic component, and implementations of those components are reusable and interoperable, communicating via a file-based textual description.

In practice the extensibility of these frameworks is limited (though uniquely, to our knowledge, a progression towards mechanisms that enable wider reuse can be seen in CIlib [84, 78, 20, 76]), and, crucially, implementation often requires the modification of internal source code, presenting a barrier to distribution, reuse, and understanding for other practitioners. We are left with a fragmented set of implementations that are incapable of representing an extensible design space for metaheuristics, without requiring modification to the frameworks themselves.

2.2. Transparency

Metaphorically-inspired approaches have recently suffered strong criticism for their lack of rigor. Where the use of metaphor obscures specific solution-domain mechanisms [100] the novelty of the metaphorical contribution becomes difficult to determine. At worst, this can lead to the re-invention or renaming of mechanisms that are already well-understood. For example, it has been argued that the popular 'Harmony search' metaheuristic can be formulated as a simple variant of the foundational 'Evolution Strategies' approach [116], and it has recently been claimed [18] that the 'Intelligent Water Drops' algorithm is similarly not novel. Such 'explanation by metaphor' unnecessarily

[5] `https://dev.heuristiclab.com/trac.fcgi/wiki/`
[6] `https://cs.gmu.edu/~eclab/projects/ecj/`
[7] `http://www.isa.us.es/fom/`
[8] `http://opt4j.sourceforge.net/`
[9] `http://jmetal.sourceforge.net/`
[10] `http://www.jamesframework.org/`
[11] `http://www.tik.ee.ethz.ch/pisa/`

obfuscates the field and makes it appear impenetrable to outsiders.

This problem is at least partly cultural: the 'reward' of publications and citations in metaheuristics is often more readily achieved by producing a method that 'beats the competition' than one that makes the additional effort to be transparent about the contribution of its mechanisms [2]. While improving on the state of the art should always be a key driver for a research community, the relentless pursuit of (apparent) novelty and the 'up-the-wall game' [100] is counter-productive. For as long as researchers continue to labor in relative isolation, the risks of overfitting and misidentifying novelty remain present. In contrast, we propose in the following sections a more 'bottom up' approach. With such an approach, new solution methods can be grounded in the principled decomposition of existing ones [58], thereby allowing ready identification of potential novelty.

2.3. Knowledge Discovery

Hooker [41] and Sörensen [100] argue that there needs to be more scientific analysis of how metaheuristics solve problems. If a metaheuristic claims to work in some way — say, for example, it is claimed that a particular operator works by moving the search out of local minima — then experiments should be performed that test this, or (even better) a theoretical justification provided. This is a particular problem for complex metaheuristics, where a number of innovations are often introduced in tandem. Compounding this issue, the existence of 'No Free Lunch' theorems for optimization [117] implies that metaheuristics often require domain-specific heuristics for success [108], and current practices mean that any expert knowledge on which mechanisms work well on a given problem must typically be reverse engineered from publications on a per-case basis. To move to a more generalized level of enquiry, it is necessary to combine, exchange, and reason about metaheuristics and their component parts (such as acceptance, selection or perturbation) on a far larger scale than has been possible to date. Significantly, we believe that this requires a shift in community culture from 'individual competition' to 'collective knowledge discovery' and the development of a large pool of shared experimental data from which to draw general conclusions.

One approach to deriving such general conclusions from a large pool of data is the application of data mining and machine learning (ML) techniques successfully used by other communities (e.g. meta-learning [79]). Individual publications have applied ML methods to selecting or constructing heuristics, and provide some evidence for their efficacy: Xu et al. [119] proposed a portfolio solver that won several SAT competitions by automatically selecting between various state-of-the-art SAT solvers based on a learned model of their relative performance conditioned on problem properties; Thabtah and Cowling [111] show associative classification can indicate which heuristic to use in each iteration of a personnel scheduling problem; Miranda et al. [69] used fitness landscape information to decide whether to build or select a new particle swarm optimization algorithm; Nallaperuma et al. [72, 73] generated predictive models of the best parameters from ant colony optimization methods based on features of previously evolved instances; Malan and Engelbrecht [63] used landscape characteristics to predict the success of a collection of PSO algorithms on unseen continuous optimization problems; Consoli et al. [22] used online learning and features extracted from the fitness landscape of the problem to choose the most appropriate genetic operator and Asta et al. [7, 8] integrated knowledge discovery directly into the search algorithm.

More generally, Smith-Miles et al. [97] proposed a methodology where instances of a problem are represented by a set of features in an instance space, and machine learning algorithms used to classify the regions of the space where algorithms are expected to perform well or poorly, given many insights on algorithms strengths and weaknesses. Such feature-based approaches provide a baseline for generating and mining knowledge of relevance to metaheuristic research and practice.

The re-use of such knowledge will first require a knowledge-base and the associated effort to constantly update it. Initial efforts towards a schema for such a database was presented by Scheibenpflug et al [95]: their Optimization Knowledge Database (OKD), contained data about algorithms, the problems they were used to solve and their parameters. The authors emphasize that populating the database is time-consuming and requires the effort of the whole community. Other works in the literature have created specific instances of such datasets [69, 97]. Such a community effort to effectively create and populate a knowledge base is paramount for the success of metaheuristics mining. Given a sufficiently rich representation for components, such analysis could be carried out in a semi-automated way.

2.4. *Automated Design*

Contemporary scientific and engineering disciplines rely heavily on standardization and automated tools. The design of these tools and their underlying algorithms tends to be an *ad hoc* process, often regarded as an art rather than a science [44]. As a consequence, the design of an algorithm is time-consuming and costly. Furthermore, the process itself is rarely documented, making it untraceable, i.e. it is often unclear what motivated certain design decisions (e.g. expert knowledge, experimentation, intuition) and which alternatives were considered. Not only do we lose potentially interesting information and insights which can be used to design algorithms in the future, it also makes the process susceptible to accidental human bias.

The automated design of algorithms has significant potential to address these issues. Unsurprisingly, attempts to (partially) automate algorithm design, in one form or the other, are ubiquitous and can be traced back to the origins of computational intelligence (e.g. program synthesis [64], genetic programming [56], swarm algorithms [51], algorithm selection [91], algorithm configuration [12]). However, the application of these techniques has thus far been largely a privilege of experts, restricted to isolated case studies, and is far from a standard practice in algorithmics.

The metaheuristics community is no exception. While metaheuristics are most commonly designed manually, the idea of automating this process is hardly new, and has been actively pursued for almost two decades in the hyper-heuristics [15] and the algorithm configuration [104] communities. An important aspect of our vision (see Section 5) is to facilitate the integration and further development of these design automation techniques.

2.5. Scalability

Historically, computing systems have tended to get faster at an exponential rate. Software performance automatically scaled along, without requiring any additional efforts from the developer. The situation is no longer so simple: contemporary systems, rather than getting faster and faster, are able to do more and more work in parallel [105]. To take advantage of increasing parallel processing capabilities, computations must be subdivided into a set of interdependent tasks to be executed efficiently in parallel across multiple cores and / or across networked machines. In computer science in general, much human effort has been invested in algorithm-specific parallelization strategies.

Scalability is also an issue when solving ever larger problem instances: despite the increase in computing power, it is hard to solve large instances of many practical optimization problems. This will of course always be the case — contrary to other computational domains, the field of combinatorial optimization will never have "enough" computing power. Fortunately, many popular metaheuristics are 'embarrassingly parallel': for example, determining the fitness of each population member in evolutionary approaches can be readily parallelized; strictly from the performance point of view and depending on latency and throughput, this simplistic approach might not be the most efficient; however, the fact that it can be done at all shows that there are parallel approaches which are functionally equivalent to sequential ones and that have a straightforward implementation.

Currently many metaheuristic methods rely on parallelism at a specific level of abstraction, typically by parallelizing either fitness evaluation or part of a population of solutions using an island model. Both these approaches are limiting in the assumptions they make about the complexity of metaheuristics: to achieve sophistication beyond previous applications may require much more complex and involved search operators, for example, and we may wish to parallelize a metaheuristic not just at these fixed and algorithm-specific levels, but to the greatest extent possible, i.e. at the level of individual components; in this sense, it might also be convenient to simply use underlying concurrent or parallel models, such as communicating sequential processes [37].

3. The 'Metaheuristics in the Large' Approach

The challenges of the previous section motivate the creation of infrastructure support for community-wide sharing of problems, metaheuristic frameworks and heuristic components. In general, there are many good tools and libraries already in existence. We do not propose to reinvent or replace them; we are not proposing "just another library" but rather a different way of structuring and implementing metaheuristics research.

Our proposed approach has three conceptual underpinnings:

- **Extensible and re-usable framework templates**
  To support open-ended innovation and provide true reusability, these templates (or any other suitable problem description language) must be configurable via a palette of components that is extensible *without requiring the modification of existing code*. To support such extensibility and the automated configuration of these templates, we require a stronger notion of interoperability than existing software: there must be infrastructure support for *state-threading*, i.e. passing framework or component-specific state (such as the temperature in simulated annealing) via a dedicated mechanism. This strong notion of extensibility is described in Section 3.1. Our approach facilitates reuse through this extensibility, and transparency and automated design by explicitly encapsulating component behavior (rather than relying on metaphorical description) and allowing machine-inspection of behaviors.

- **White box problem descriptions**
  Having embraced the necessity of state threading, it follows that we can thread more than merely empirically-obtained data relating to the search trajectory. In particular, threaded state can include *analytic* information, such as declarative / whitebox problem descriptions. This analytic information can be used to guide algorithm selection or construction in a more informed manner than has traditionally been embraced by the hyper-heuristics community [108]. We discuss this further in Section 3.2.

- **Remotely accessible frameworks, components and problems**
  By building upon the two concepts above, it is possible to configure pre-existing, remotely-hosted, algorithm frameworks with some (potentially newly-devised) collection of heuristic components. The practical obstacle to further progress is then the relatively procedural one of community agreement on definitions for component interfaces and communication protocols. For inspiration, we look to work on 'Service Oriented Architecture', which we discuss in Section 3.3. This enables widespread reuse, replicability, and shared knowledge discovery.

3.1. Re-usable Framework Templates

The main obstacle to the open-ended extension and automated composition of existing implementations of metaheuristic components is that they suffer from an intrinsic lack of modularity. In this section, we illustrate why this is an issue for research 'in the large' and describe the proposed solution. In part, this is due to the lack of adoption of best practice from software engineering [37]. In the case of metaheuristics, there is an added complexity due to state dependencies between different algorithms components, which we now describe.

Framework configuration can be defined in general terms by expressing frameworks as higher-order functions that take components as parameters. For example, a possible function signature[12] for acceptance for some generic candidate solution `Sol` is:

$$accept : \text{incumbent} : Sol \times \text{incoming} : Sol \rightarrow \text{Boolean}$$

Listing 1 gives a simple local search framework that allows for three design decisions, viz. the choice of perturbation, acceptance and termination conditions. In order to support alternative designs, local search is a *higher-order function*: it takes as arguments separate *functions* for *perturb*, *accept* and *finished* and returns a candidate solution of type Sol. As described above, `accept` takes as argument a pair of candidate solutions (i.e. the incumbent and incoming solutions) and returns the prefered one, denoted in the following listing as: `accept : (Sol,Sol)=>Sol`. `perturb` and `finished` are defined the correspondingly obvious manner.

```
def localSearch(
      incumbent: Sol,
      perturb: Sol => Sol,
      accept:  (Sol,Sol)  => Sol,
      finished: Sol  => Boolean
    ): Sol {

    while( not finished( incumbent ) )
        incumbent = accept( incumbent, perturb( incumbent ) )

    return incumbent;
}
```

Listing 1: Local Search framework parameterized by design choices

Each specific triple of components (*perturb*, *accept*, *isFinished*) used to configure the framework corresponds to a specific local search algorithm. This allows us to concisely specify a combinatorial design space of alternative component configurations, and also makes design space commonalities explicit. In order for a framework to permit the substitution of different choices for each component, components must ultimately conform to some well-defined interface, e.g. the higher-order function arguments to the framework have some signature that is fixed *a priori*. In our example, for candidate solution type *Sol* (e.g. a list of cities in the Traveling Salesperson Problem) perturbation is assumed to have type $Sol \rightarrow Sol$.

However, such fixed signatures are problematic if we wish such frameworks to be 'closed to modification', i.e. be able to accommodate unanticipated

[12]The *signature* of a function is the formal description of its parameter and return types.

component dependencies without requiring changes to framework code. The need for such modification is clearly incompatible with the MitL goal of frameworks that can be both shared across the research community and be configured (perhaps automatically) with new components. As a concrete example: suppose we now wish to incorporate a further heuristic that requires information about the search trajectory, e.g. a tabu list of solutions [35] that promotes search diversification. We are therefore required to change the implementation of local search to keep track of the trajectory. Listing 2, gives a revised version in which the history list of previous incumbent solutions is denoted by $[Sol]$.

```
def localSearch(
    current: Sol,
    history: [Sol],
    perturb: (Sol,[Sol]) => Sol,
    accept: (Sol,Sol,[Sol]) => (Sol, [Sol]),
    finished: (Sol,[Sol]) => Boolean
    ): (Sol,[Sol]) {

    while( not finished( current, history ) )
        (current,history) = accept(current,perturb(current,history),
            history);

    return (current,history);
}
```

Listing 2: Explicit incorporation of solution history

The modified implementation now supports solution-based tabu mechanisms, but the issue of course persists if we wish to incorporate components which require new state dependencies, for example Metropolis-Hastings acceptance, which requires some measure of 'temperature' [52] to be statefully maintained. In the general case, we clearly cannot anticipate in advance what information will be required by some component yet to be devised. These are examples of *environmental state*, which provides the context for decisions made by the search process. For extensibility, it is therefore necessary for support for environmental state to be open-ended, i.e. for frameworks to be configurable with components that access aspects of environmental state that are not known at the time of framework implementation. Principled handling of environment state is key to metaheuristic modularization, and is therefore essential for both component interoperability and scalability in automated construction of metaheuristics. The technical specifics of MitL support for this approach are described in an associated publication [106] and summarized in Appendix A. Software exemplars of the proposed infrastructure support are publicly available, as described in Appendix B.

3.2. *White box Problem Representations*

It is well-known that the exploitation of problem information is key in rendering optimization problems tractable [117]. As such, a principal challenge lies in devising frameworks and solvers that support injection of problem information to drive the search process, without incurring loss of genericity (i.e. they can be applied to many different problems).

By including *white box problem descriptions* as part of the environmental state, it is possible to define frameworks in terms of rich domain information, allowing the aforementioned challenge to be tackled using an open-ended combination of human ingenuity and automation.

What Kind of Information Can Be Exploited?

In principle, *any* machine-readable knowledge could be exploited to bias the search, to synthesize feasible operators, etc. As discussed in the introduction, this knowledge can be split into two categories:

Analytic knowledge about intrinsic features of the problem.

Empirical knowledge gained through experience, i.e. experimentation.

Many examples of successfully exploiting a combination of analytic and empirical knowledge can be found in the literature:

> Reactive tabu search [10] exploits knowledge about the presence of specific substructures in candidate solutions to diversify the search, and uses trajectory information to adapt the tabu tenure parameter dynamically.
>
> Variable neighborhood descent [40] exploits knowledge about the relative sizes of multiple domain-specific neighbor relations to (local) search them more efficiently, and uses empirical information (a candidate solution being locally optimal / improving) to switch between neighbor relations.
>
> Matheuristics (e.g. [1, 75] typically combine analytical approaches such as ILP to solve sub-problems, with higher-level searches using empirical feedback on solution quality to build the results into solutions for a larger-scale problem.
>
> Solution spaces can be decomposed using techniques with their origins in mathematical programming, to increase efficiency at the metaheuristic level [90].
>
> Constraint relaxation (e.g. [32]) typically uses analytical knowledge of the acceptable constraint bounds to allow a metaheuristic to search across infeasible regions of the space using empirical feedback on solution quality to determine when the relaxation should be reduced. Similarly, different heuristics can be targeted at different constraints, driven by analytical knowledge of the constraints themselves [36].
>
> Portfolio solvers (e.g. SATzilla [119]), select between multiple solvers based on analytic features of the problem instance to be solved. The mapping from features to solvers is generated empirically, using machine learning.

Historical Development Towards White Box Approaches

While the importance of exploiting problem structure is widely recognized, arguably there is a historical aversion to do so at the hyper-heuristic level, leaving this task up to the domain-specific instantiations or low-level heuristics [28]. Maintaining generality is often cited as the motivation for this information hiding practice. For example, Chakhlevitch and Cowling [19] argue for the importance of limiting problem domain information in achieving cross-domain generality in selection hyper-heuristics. They argue that a framework can be applied to any problem that shares the "lowest common denominator" characteristics. While sufficient for generality, information hiding is not necessary. It is easy to see that a framework can exploit arbitrary information without loss of generality, as long as it is also capable of solving the problem without it. For instance, a general optimizer could use gradient information when available (e.g. when training neural networks) and default to a derivative-free approach otherwise.

The progression of hyper-heuristic research demonstrates an increased acknowledgment that use of white box problem descriptions is both possible and desirable. Following the pattern set by initial work [23], most of the selection hyper-heuristics studies maintain a black-box interface between the hyper-heuristic and problem domain known as the *domain barrier*. The original rationale for the domain barrier, which disallows a hyper-heuristic from retrieving any problem-specific information, was thought to be necessary for cross-domain generality. However, it has been recognized that the domain barrier might be more a problem than a feature: Ross [93] argued that an explicit domain barrier that enforces a strict separation between the hyper-heuristic and the problem-specific aspects makes hyper-heuristics undesirable for use in large real-world applications. Furthermore, Parkes et al [83] and Pappa et al [79] suggested an increased exchange of information between the problem domain and the higher search level which could then be analyzed via data science techniques and machine learning. More advanced learning for heuristic selection has progressively been introduced [16, 89, 103, 50, 4, 11, 3].

Recent work throws further doubt on the necessity of the domain barrier. Swan et al [108] state that work in constraint-satisfaction provides abundant evidence that problems can be described in a domain-independent manner without loss of solver generality. The lack of necessity for the domain barrier was further evidenced by Kheiri [49], who designed a hyper-heuristic utilizing extended domain information that nonetheless manages low-level heuristics in a domain-independent manner. Martin et al [66] designed a hyper-heuristic controlling the parameter settings of randomised heuristics based on an agent-based cooperative search framework. An ontology is used to translate problem-specific elements into problem-independent abstract objects.

White Box Descriptions and Automation

In order to communicate the required information to the solver and its heuristics, it is necessary to move beyond the bespoke approaches above: it must be possible to communicate arbitrary domain knowledge to an algorithm framework template. In a black box setting, life is simple: solvers can choose from a closed set of interfaces and can solve any problem that implements it. In contrast, the white box setting is completely open-ended: solvers can require whatever information they deem fit. Clearly, this presents an array of novel challenges. Most notably:

1. Deducing (analytically) which solvers can be applied to which problems.

2. Overcoming limited applicability due to interface mismatches.

This is clearly a rich topic for further research. However, we believe that some foundational aspects can be identified:

1. The use of a declarative, machine-readable language to express the information problems provide, solvers require, and their relations.

2. Automated algorithm selection and problem (re)formulation, as facilitated by white box descriptions [108].

The practical choice for such a language is rightfully an open-ended research question, but the MitL proposal [108] is that prior art in Constraint Programming provides a suitably generic baseline. Existing standards within Constraint Programming, such as XCSP3 [14] support whitebox descriptions of constraints that capture a very wide range of common problems. As research advances, the set of supported constraints can further expand, without limiting applicability. Ultimately, interface boundaries will fade, causing a paradigm shift, where human researchers specify components using problem information and computers assemble them into a single framework, automatically selecting the component believed to work best, based on all available analytic and empirical information.

3.3. Service Oriented Architecture

A large-scale solution to lack of re-use lies in “Service Oriented Science”, which applies the increasingly-widely adopted practice of service-oriented architectures [38] to scientific computing. The concept is defined as “the pursuit of scientific research using distributed and interoperable services, the accessibility of these interfaces being the key to success” [31]. By such means, researchers can discover and access services without developing specific programmatic clients for each data source, or program. Such an approach clearly has the potential to increase scientific productivity via public and distributed services, and also to increase data analysis automation. There are many examples that attempt to boost this paradigm, such as the Open Science Grid [5] and GLOBUS [30]. These projects include scientific communities and globally distributed infrastructures that support scientific and integrated applications of different domains.

As we have argued above, it is highly advantageous for metaheuristic researchers and practitioners to converge on a standard machine-readable language for problem description, experimental configuration and results. Service

Oriented Architectures (SOAs) offer several ways to build a research workflow from these elements. SOA is a computational paradigm in which agents interact using loosely coupled, coarse-grained, and autonomous components called *services* [94]. A *service* is a distributed entity, such as a node, program or function, used to obtain a result, increasing the integration of systems that are heterogeneous in respect of operating systems, protocols or languages.

The SOA perspective promotes the creation of services that are discoverable and dynamically-bound, self-contained / modular, loosely-coupled, location-transparent and composable [112]. As such, SOA is clearly therefore a good fit for the process of "devolved community research" which we advocate here. Lately, SOA has seen a trend towards "microservice architectures": distributed, cloud-based and cloud-native, these architectures follow the principle of separation of concern to create applications that are easily scalable and deployable, with a stable response and maximum availability. Several frameworks, such as the one proposed by Khalloof et al. [48], exploit the capabilities of microservices to create scalable systems that can be used at different levels (from the desktop to the web) for optimization. However, at the time of writing there are no generally accepted standards for microservice discovery, and although they offer some advantages in term of composability and scalability, they lack the service representation feature that would make it amenable to use within a large-scale metaheuristics framework.

SOA for Metaheuristics

Previous work on SOA for metaheuristics has mainly been concerned with the application of a specific metaheuristic, such as Genetic Algorithms, to optimize a service selection or composition based on the QoS (Quality of Service) of their execution [92].

Different SOA technologies, such as web services, have been proposed for solving optimization problems via grid computing [24, 98, 99], where services are defined using WSDL (Web Services Description Language) interfaces and other transmission mechanisms (such as Remote Procedure Call [57] or Globus Toolkit [45]). ROS (Remote Optimization Service) [33] was one of the first attempts to allow remote execution of metaheuristics, with inputs and outputs described via XML specification. Other metaheuristic frameworks such as HeuristicLab include plug-ins to allow parallelism and interoperability using web services [115]. GridUFO is a service oriented framework [70], but it only allows the modification of the objective function and the addition of whole algorithms, without combining existing services.

García-Sánchez et al [34] have previously proposed a SOA for Evolutionary Algorithms. Several suggestions on different concerns about the design and development of the elements of an EA using SOA were presented, such as the operator behavior, dynamism or solution representation. A specific SOA technology, OSGi, was used as an example of implementation. More recently, MOSES [80] was proposed as the design of a global architecture based on service contracts, allowing the automation of the experimentation process in a metaheuristic optimization context. This architecture is based on different tools, such as specific automatic experimental description languages and statistical services, forming a contract-based chain of software components for experimental execution. These global architectures have already been proposed for other fields, such as distributed simulation [110].

Most of these approaches are more or less direct mappings from the original implementation to a SOA framework; however, one key way in which the proposed approach facilitates knowledge discovery is the ability to add arbitrary instrumentation to components via the generic environment representation. In particular, this allows for data mining on metaheuristic traces. In addition, by employing our generic notion of state (which denotes one or more solutions, together with any environmental information required to represent the current algorithmic state of the search), the same framework can instantiate metaheuristics operating at different scales. For example, a composite recombination operator can choose from different types of recombination strategies, putting meta and hyper-heuristics under the same framework (e.g. in the manner of [118]). Having these different types of algorithms under a common framework greatly facilitates their extension and comparison.

More generally, we envision the emergence of a distributed, community driven suite of tools, providing an expanded repository of interoperable frameworks and components, bringing together researchers and practitioners across domains, unifying the field and closing the gap between scientific research and empirical practice.

4.

5. Use-Cases for MitL

As we have intimated, a purely technical solution to the issues of metaheuristics research is insufficient: community-level engagement is also required. While we have proposed a means by which extensible algorithm templates can integrate with other frameworks, the ultimate arbiter for success is the enthusiasm of the wider research community to embrace such initiatives. In this section, we describe some of the prospective benefits of doing so.

A recent paper by Kendall et al. [47] has, importantly, emphasized the need for good laboratory practice in optimization research. The set of practices that they advocate include making datasets available in a standardized format, reporting the results from the individual components of a hybrid approach, describing in a reproducible way the evaluation function and the metaheuristic used, clearly presenting computational times, the use of appropriate statistical tests, etc. We disagree with none of this. However, a driving philosophy for MitL is that it is necessary to go beyond the mere *advocacy* of good practice, to making it easy — indeed, almost inevitable — that good practice can happen. The MitL proposal is that it is possible to embed foundational support for good practice directly into the software that is used by metaheuristics researchers, consequently making the fruits of that research available to other practitioners as via their default workflow. Below, we describe some specific use cases in metaheuristic research which are facilitated by the proposed approach.

5.1. Comparison between Metaheuristics

Many papers in the metaheuristics literature compare the performance of a new metaheuristic against a small sample of other metaheuristics. Sometimes, it is made clear that the comparators chosen have been specifically selected because they represent the state-of-the-art for that particular problem area: however, in many cases this is not made clear. Furthermore, many papers simply compare the new metaheuristic against other metaheuristics of the same broad type, for example, comparing a new variant on Particle Swarm optimization (PSO) against other PSO variants.

Providing some evidence for the effectiveness of a new method is clearly important. However, as the number of metaheuristics continues to expand, comparing against a few other metaheuristics seems weak; even where an assertion is made that the comparators represent the state-of-the-art, this is usually presented as an assertion to be taken on trust, and the method used to choose the comparators is unstated. In particular, there is no guarantee that the chosen problem instances actually exhibit different landscape characteristics. We therefore propose that creators of new metaheuristics should test their metaheuristics against *all* other appropriate metaheuristics.

The overall aim is therefore to ensure that comparisons are both thorough and fair. This is clearly only realistic at a large scale if the process of comparisons is automated. Some initial progress in this direction was made several years ago: Taillard et al. [109] suggest that iterative metaheuristics should be compared not only for a single computational effort (e.g. giving the best solution found after a fixed number of iterations), but also continuously at each iteration[13]. Some recent work has also started to address the issue of fair comparison of algorithms by providing statistical testing frameworks which ensure that the preconditions for the various tests applied are actually met [74]. This is particularly important for metaheuristics, since common assumptions (e.g. of normality) are not in general true. There is also a need to ground reported results in terms of "effect magnitude" [74]: for example, an improvement of 0.1% on the state-of-the-art may have more practical relevance for the Traveling Salesman Problem than for Bin-packing. In addition to statistical considerations, the specifics of the termination condition are obviously also a vital aspect of fair comparisons. We claim that the transparency afforded by the proposed approach is vital in ensuring that comparisons are commensurate.[14]

[13] `http://mistic.heig-vd.ch/taillard/qualopt/`

[14] For a community effort that promotes best practices in benchmarking, we refer the curious reader to [9]. It discusses eight topics: clearly stated goals, well-specified problems, suitable algorithms, adequate performance measures, thoughtful analysis, effective and efficient designs, comprehensible presentations, and guaranteed reproducibility.

5.2. Testing Against Problem Instances

Establishing the effectiveness of a metaheuristic requires two components: other metaheuristics against which to compare, and a range of problem instances upon which to compare them. Again, many papers in the metaheuristics literature are considerably less than comprehensive in the range of problems which are investigated. While certain benchmark test suites are available, the application of a specific new method to these benchmarks is often done in a seemingly *ad hoc* way, with a number of examples from the benchmark chosen without any justification [41].

As described in previous MitL work [108], progress in this area depends on the ability to express a wide variety of problem types in a common format. This further facilitates the creation of systems that could apply a new metaheuristic systematically over a wide range of problems. Researchers would not be limited by the amount of time it would take to set up experiments with a large number of problem instances; an automated script can work through a repository of problems, automatically applying the new metaheuristic to each appropriate example.

Of course, benchmarking metaheuristics cannot be reduced to simply counting the number of problems for which a particular metaheuristic is “better” than another one [9]. Rather, the aim is to gather a rich set of data about the performance of each metaheuristic on a wide variety of problem types and instances. Such knowledge database, in combination with white box descriptions of problems (and introspectable methods), could then be mined to gain deeper insights into which methods work best when (and why). In particular, it allows more general metaheuristics to be constructed automatically from more specialized ones (e.g., using algorithm selection portfolios as in [119]).

5.3. Hybridization

The hybridization of solution methods has been a successful approach for combining the complementary strengths of different optimization paradigms and to reduce their individual weaknesses with the aim of obtaining more effective algorithms (see e.g., [113] for a review from the perspective of the CP-AI-OR community). Hybrid approaches can be classified according to many dimensions [88], e.g. whether the components involved in the hybridization come from different search paradigms (usually constructive methods or exact methods such as Constraint Programming or Integer Linear Programming) or whether they are homogeneous (e.g., local search or evolutionary methods). Indeed, presenting a specific hybridization of two or more metaheuristics is a common source of novelty in metaheuristic research. Unfortunately, the way in which these hybridizations are evaluated is often a very simplistic comparison in terms of accuracy or error measures, without any attempt to attribute specific behaviors in a run of the metaheuristic to particular components, or indeed to perform any elimination of “accidental complexity” [2].

Often, the way in which metaheuristics are combined is the strongest contribution of a method. For example, ’Fair Share ILS’ [2] performs well because of the synergistic interaction between its acceptance and perturbation heuristics. More generally, it is of relatively little use for perturbation to be operating strongly as a "search intensifier" if the acceptance criterion only permits large increases in solution quality. Such decisions are best informed via large-scale studies (as supported by combinatorial assembly over a range of problems and algorithm configurations) and component instrumentation (as supported via the proposed environmental state threading).

5.4. Matching Metaheuristics to Problem Types

Although there has long been interest in relating problem characteristics to solution strategies [91], we claim that this is an area that has been particularly hindered by the lack of re-use. One problem with these studies, valuable as they are, is that they represent a single sample point in time. As new metaheuristics are created, the value of that cross-cutting analysis becomes weaker, as researchers present new, *ad hoc* evidence for the value of a particular metaheuristic on a particular problem. Given the importance of such cross-cutting studies, we propose that it is key for the community to support a constantly-updated repository of metaheuristics and experiments, and subsequently so that the most effective metaheuristic for a particular problem area can be identified and kept up-to-date. Importantly, this would need more than just the creation of such a repository. Analysis tools would also be needed, which would mine the ever-expanding repository to find features that best predict which kind of metaheuristic is well-suited to a novel dataset. This would ideally involve the automated application of metaheuristics to problems, with the repository constantly being updated as new problems and (meta)heuristics are added.

6. Conclusion

Metaheuristics in the Large (MitL) is a community project that addresses some of the cultural and technical issues we believe are impediments to progress in metaheuristic research:

- Through the MitL component-based architecture and explicit state threading outlined in Section 3, heuristics can be described from a behavioral standpoint, moving away from an over-reliance on metaphor and the accidental re-invention of established heuristics.
- MitL eliminates the need to modify existing framework source code when implementing new heuristics. Whilst this is of great benefit to a practitioner, it is *essential* for the open-ended combinatorial assembly of metaheuristics. Design automation can raise the abstraction level of research from manual labor such as parameter tuning and selecting and combining heuristics, towards answering more general scientific questions.

- Within the space of a few years, Deep Learning approaches have changed the perspective on what is possible in Machine Learning. By following MitL's approach to defining heuristics, it should be easier for practitioners to recursively define very large-scale metaheuristic architectures (e.g. exploiting parallelism) without undue concern for low-level implementation details, enabling exploitation of the large-scale parallelism of modern compute platforms.

The approaches described in Section 3 combine to provide a basis for extensible Software as a Service implementations of metaheuristics provided via stateless web-services, supporting shared framework templates which allow combinatorial assembly and comparison of metaheuristics. The language and platform agnosticism of this approach in turn addresses issues of reproducibility and scalability.

Metaheuristics are one of the great contributions to practical computer science of the last few decades. However, without interoperable frameworks for analyzing, comparing and hybridizing them, advances in the *science* of metaheuristics are few and far between. Once such frameworks are in place, we will be able to put metaheuristics on a much more experimentally rigorous footing, to advance the science of metaheuristics, and to build a communal resource that is of benefit to both practitioners and researchers in this important area of computational intelligence.

In this article, we have described the key functionality that supporting infrastructure requires. What is now needed is community consensus on the relatively procedural aspects of interoperability protocols. Editors and reviewers can then insist on a thorough and systematic application of new metaheuristics to a wide range of problems, with the attendant rich analysis possibilities that are opened up.

Acknowledgments

Many people have generously given their time to the various activities of the MitL initiative. Particular gratitude is due to Adam Barwell, John A. Clark, Patrick De Causmaecker, Kevin Hammond, Emma Hart, Zoltan A. Kocsis, Ben Kovitz, Krzysztof Krawiec, John McCall, Nelishia Pillay, Kevin Sim, Jim Smith, Thomas Stützle, Eric Taillard and Stefan Wagner.

J. Swan acknowledges the support of UK EPSRC grant EP/J017515/1 and the EU H2020 SAFIRE Factories project. P. García-Sánchez and J. J. Merelo acknowledges the support of TIN2017-85727-C4-2-P by the Spanish Ministry of Economy and Competitiveness. M. Wagner acknowledges the support of the Australian Research Council grants DE160100850 and DP200102364.

References

[1] Addis, B., Carello, G., and Ceselli, A. (2013). Combining very large scale and ilp based neighborhoods for a two-level location problem. *European Journal of Operational*

Research, 231(3):535 – 546.

[2] Adriaensen, S., Brys, T., and Nowé, A. (2014). Fair-share ILS: A Simple State-of-the-art Iterated Local Search Hyperheuristic. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1303–1310, New York, NY, USA. ACM.

[3] Agarwal, A., Colak, S., and Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169(3):801 – 815.

[4] Ahmed, L., Mumford, C., and Kheiri, A. (2019). Solving urban transit route design problem using selection hyper-heuristics. *European Journal of Operational Research*, 274(2):545–559.

[5] Altunay, M., Avery, P., Blackburn, K., Bockelman, B., Ernst, M., Fraser, D., Quick, R., Gardner, R., Goasguen, S., Levshina, T., Livny, M., McGee, J., Olson, D., Pordes, R., Potekhin, M., Rana, A., Roy, A., Sehgal, C., Sfiligoi, I., Würthwein, F., and Open Sci Grid Executive Board (2011). A Science Driven Production Cyberinfrastructure-the Open Science Grid. *Journal of GRID Computing*, 9(2, Sp. Iss. SI):201–218.

[6] Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, USA.

[7] Asta, S. and Özcan, E. (2015). A tensor-based selection hyper-heuristic for cross-domain heuristic search. *Information Sciences*, 299:412 – 432.

[8] Asta, S., Özcan, E., and Curtois, T. (2016). A tensor based hyper-heuristic for nurse rostering. *Knowledge-Based Systems*, 98:185 – 199.

[9] Bartz-Beielstein, T., Doerr, C., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., Lopez-Ibanez, M., Malan, K. M., Moore, J. H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., and Weise, T. (2020). Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*.

[10] Battiti, R. and Tecchiolli, G. (1994). The reactive tabu search. *ORSA journal on computing*, 6(2):126–140.

[11] Bengio, Y., Lodi, A., and Prouvost, A. (2020). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*.

[12] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[13] Bleuler, S., Laumanns, M., Thiele, L., and Zitzler, E. (2003). PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In Fonseca, C. M. et al., editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508, Berlin. Springer.

[14] Boussemart, F., Lecoutre, C., and Piette, C. (2016). XCSP3: an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398.

[15] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

[16] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

[17] Cahon, S., Melab, N., and Talbi, E.-G. (2004). ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3).

[18] Camacho-Villalón, C. L., Dorigo, M., and Stützle, T. (2019). The intelligent water drops algorithm: why it cannot be considered a novel algorithm - A brief discussion on the use of metaphors in optimization. *Swarm Intelligence*, 13(3-4):173–192.

[19] Chakhlevitch, K. and Cowling, P. (2008). *Hyperheuristics: Recent Developments*, pages 3–29. Springer Berlin Heidelberg, Berlin, Heidelberg.

[20] Cloete, T., Engelbrecht, A. P., and Pamparà, G. (2008). Cilib: A collaborative framework for computational intelligence algorithms - part II. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*, pages 1764–1773. IEEE.

[21] Collberg, C., Proebsting, T., and Warren, A. M. (2015). Repeatability and benefaction in computer systems research. Technical Report TR 14, University of Arizona.

[22] Consoli, P., Minku, L. L., and Yao, X. (2014). Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics. In *10th International Conference on Simulated Evolution And Learning*, volume 8886 of *LNCS*. Springer.

[23] Cowling, P., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In Burke, E. and Erben, W., editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg.

[24] Cox, S. J., Fairman, M. J., Xue, G., Wason, J. L., and Keane, A. J. (2001). The grid: Computational and data resource sharing in engineering optimisation and design search. In *30th International Workshops on Parallel Processing (ICPP 2001 Workshops), 3-7 September 2001, Valencia, Spain*, pages 207–212. IEEE Computer Society.

[25] Dantzig, G. B. (1990). A history of scientific computing. chapter Origins of the Simplex Method, pages 141–151. ACM, New York, NY, USA.

[26] De Beukelaer, H., Davenport, G. F., De Meyer, G., and Fack, V. (2017). James: An object-oriented java framework for discrete optimization using local search metaheuristics. *Software: Practice and Experience*, 47(6):921–938. spe.2459.

[27] Di Gaspero, L. and Schaerf, A. (2003). Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience*, 33(8):733–765.

[28] Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428.

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

[30] Foster, I. (2005a). Globus Toolkit version 4: Software for service-oriented systems. In Jin, H and Reed, D and Jiang, W, editor, *Network and Parallel Computing Proceedings* Özcan, .955 volume, of

[10] Foster, I. (2005b). service-oriented

[43] Hughes, J. (1989). Why functional programming matters. *The Computer Journal*, 32(2):98–107.

[44] Hunt, A. and Thomas, D. (2001). The art in computer programming. *The Pragmatic Programmers, LLC*.

[45] Imade, H., Morishita, R., Ono, I., Ono, N., and Okamoto, M. (2004). A grid-oriented genetic algorithm framework for bioinformatics. *New Gen. Comput.*, 22(2):177–186.

[46] Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59:215–250.

[47] Kendall, G., Bai, R., Błazewicz, J., De Causmaecker, P., Gendreau, M., John, R., Li, J., McCollum, B., Pesch, E., Qu, R., et al. (2016). Good laboratory practice for optimization research. *Journal of the Operational Research Society*, 67(4):676–689.

[48] Khalloof, H., Jakob, W., Liu, J., Braun, E., Shahoud, S., Duepmeier, C., and Hagenmeyer, V. (2018). A generic distributed microservices and container based framework for metaheuristic optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1363–1370. ACM.

[49] Kheiri, A. (2020). Heuristic sequence selection for inventory routing problem. *Transportation Science*, 54(2):302–312.

[50] Kheiri, A. and Özcan, E. (2016). An iterated multi-stage selection hyper-heuristic. *European Journal of Operational Research*, 250(1):77–90.

[51] Khichane, M., Albert, P., and Solnon, C. (2008). Integration of ACO in a constraint programming language. *Ant Colony Optimization and Swarm Intelligence*, pages 84–95.

[52] Kirkpatrick, S., Jr., D. G., and Vecchi, M. P. (1983). Optimization by simmulated annealing. *Science*, 220(4598):671–680.

[53] Kocsis, Z. A., Brownlee, A. E. I., Swan, J., and Senington, R. (2015). Haiku - a Scala combinator toolkit for semi-automated composition of metaheuristics. In Barros, M. and Labiche, Y., editors, *Search-Based Software Engineering*, volume 9275 of *Lecture Notes in Computer Science*, pages 125–140. Springer International Publishing.

[54] Kocsis, Z. A. and Swan, J. (2017). Dependency injection for programming by optimization. *CoRR*, abs/1707.04016.

[55] König, M. (2020). Executable simulation model of the liver. *bioRxiv*.

[56] Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

[57] Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B., and Lee, B.-S. (2007). Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23(4):658 – 670.

[58] López-Ibáñez, M., Mascia, F., Marmion, M.-E., and Stützle, T. (2014). A template for designing single-solution hybrid metaheuristics. In *GECCO Comp '14*, pages 1423–1426, New York, USA.

[59] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58.

[60] Lukasiewycz, M., Glaß, M., Reimann, F., and Teich, J. (2011). Opt4j - a modular framework for meta-heuristic optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland.

[61] Luke, S. (2010). The ECJ owner's manual. `http://www.cs.gmu.edu/~eclab/projects/ecj`.

[62] Luke, S. (2017). ECJ then and now. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 1223–1230, New York, NY, USA. ACM.

[63] Malan, K. M. and Engelbrecht, A. P. (2014). *Fitness Landscape Analysis for Metaheuristic Performance Prediction*, pages 103–132. Springer Berlin Heidelberg, Berlin, Heidelberg.

[64] Manna, Z. and Waldinger, R. (1980). A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(1):90–121.

[65] Marmion, M.-E., Mascia, F., López-Ibánez, M., and Stützle, T. (2013). Automatic design of hybrid stochastic local search algorithms. In *International workshop on hybrid metaheuristics*, pages 144–158. Springer.

[66] Martin, S., Ouelhadj, D., Beullens, P., Ozcan, E., Juan, A. A., and Burke, E. K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, 254(1):169 – 178.

[67] Merelo-Guervós, J. J., Castillo-Valdivieso, P. Á., Romero-López, G., and García-Arenas, M. (2003). Specifying evolutionary algorithms in xml. In *International Work-Conference on Artificial Neural Networks*, pages 502–509. Springer.

[68] Merelo Guervós, J. J. and Valdez, J. M. G. (2018). Mapping evolutionary algorithms to a reactive, stateless architecture: using a modern concurrent language. In Aguirre, H. E. and Takadama, K., editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 1870–1877. ACM.

[69] Miranda, P. B., Prudêncio, R. B., and Pappa, G. L. (2017). H3ad: A hybrid hyper-heuristic for algorithm design. *Information Sciences*, 414(Supplement C):340 – 354.

[70] Munawar, A., Wahib, M., Munetomo, M., and Akama, K. (2010). The design, usage, and performance of gridufo: A grid based unified framework for optimization. *Future Generation Computer Systems*, 26(4):633 – 644.

[71] Nagata, Y. and Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In *Proceedings of the 7th International Conference on Genetic Algorithms, MI, USA*.

[72] Nallaperuma, S., Wagner, M., and Neumann, F. (2014). *Parameter Prediction Based on Features of Evolved Instances for Ant Colony Optimization and the Traveling Salesperson Problem*, pages 100–109. Springer International Publishing, Cham.

[73] Nallaperuma, S., Wagner, M., and Neumann, F. (2015). Analyzing the effects of instance features and algorithm parameters for max–min ant system and the traveling salesperson problem. *Frontiers in Robotics and AI*, 2:18.

[74] Neumann, G., Swan, J., Harman, M., and Clark, J. A. (2014). The executable experimental template pattern for the systematic comparison of metaheuristics: Extended abstract. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1427–1430, New York, NY, USA. ACM.

[75] Nikzad, E., Bashiri, M., and Abbasi, B. (2021). A matheuristic algorithm for stochastic home health care planning. *European Journal of Operational Research*, 288(3):753 – 774.

[76] Pamparà, G. and Engelbrecht, A. (2015). Towards a generic computational intelligence library: Preventing insanity. In *2015 IEEE Symposium Series on Computational Intelligence: IEEE Workshop on Computational Intelligence Tools (2015 IEEE WCIT)*, Cape Town, South Africa.

[77] Pamparà, G. and Engelbrecht, A. P. (2019). Evolutionary and swarm-intelligence algorithms through monadic composition. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 1382–1390, New York, NY, USA. Association for Computing Machinery.

[78] Pamparà, G., Engelbrecht, A. P., and Cloete, T. (2008). Cilib: A collaborative framework for computational intelligence algorithms - part I. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*, pages 1750–1757. IEEE.

[79] Pappa, G. L., Ochoa, G., Hyde, M. R., Freitas, A. A., Woodward, J., and Swan, J. (2014). Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35.

[80] Parejo, J. A. (2016). MOSES: A metaheuristic optimization software ecosystem. *AI Commun.*, 29(1):223–225.

[81] Parejo, J. A., Racero, J., Guerrero, F., Kwok, T., and Smith, K. A. (2003). *FOM: A Framework for Metaheuristic Optimization*, pages 886–895. Springer Berlin Heidelberg, Berlin, Heidelberg.

[82] Parejo, J. A., Ruiz-Cortés, A., Lozano, S., and Fernandez, P. (2012). Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561.

[83] Parkes, A. J., Özcan, E., and Karapetyan, D. (2015). A software interface for supporting the application of data science to optimisation. In *International Conference on Learning and Intelligent Optimization*, pages 306–311. Springer.

[84] Peer, E. S., Engelbrecht, A. P., Pamparà, G., and Masiye, B. S. (2005). Ciclops: computational intelligence collaborative laboratory of pantological software. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 130–137.

[85] Pellerin, R., Perrier, N., and Berthaut, F. (2020). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 280(2):395 – 416.

[86] Popper, K. (1963). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge classics. Routledge.

[87] Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.

[88] Puchinger, J. and Raidl, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, pages 113–124. Springer.

[89] Qu, R., Burke, E. K., and McCollum, B. (2009). Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404.

[90] Raidl, G. R. (2015). Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66 – 76.

[91]

[100] Sörensen, K. (2013). Metaheuristics—the metaphor exposed. *International Transactions on Operational Research*, 22(1):3–18.

[101] Sörensen, K. and Glover, F. W. (2013). *Metaheuristics*, pages 960–970. Springer US, Boston, MA.

[102] Sörensen, Kenneth and Arnold, Florian and Palhazi Cuervo, Daniel (2019). A critical analysis of the "improved Clarke and Wright savings algorithm". *International Transactions in Operational Research*, 26(1):54–63.

[103] Soria-Alcaraz, J. A., Ochoa, G., Sotelo-Figeroa, M. A., and Burke, E. K. (2017). A methodology for determining an effective subset of heuristics in selection hyper-heuristics. *European Journal of Operational Research*, 260(3):972–983.

[104] Stützle, T. and López-Ibáñez, M. (2019). Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pages 541–579. Springer.

[105] Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs Journal*, 30(3).

[106] Swan, J., Adriaensen, S., Barwell, A. D., Hammond, K., and White, D. R. (2019). Extending the 'open-closed principle' to automated algorithm configuration. *Evolutionary Computation*, 27(1):173–193.

[107] Swan, J., Adriaensen, S., Bishr, M., Burke, E. K., Clark, J. A., Causmaecker, P. D., Durillo, J., Hammond, K., Hart, E., Johnson, C. G., Kocsis, Z. A., Kovitz, B., Krawiec, K., Martin, S., Merelo, J. J., Minku, L. L., Özcan, E., Pappa, G. L., Pesch, E., Garcia-Sànchez, P., Schaerf, A., Sim, K., Smith, J., Stützle, T., Voß, S., Wagner, S., and Yao, X. (2015). A research agenda for metaheuristic standardization. In *Proceedings of the Eleventh Metaheuristics International Conference (MIC), Agadir, Morocco*.

[108] Swan, J., De Causmaecker, P., Martin, S., and Özcan, E. (2018). *A Re-characterization of Hyper-Heuristics*, pages 75–89. Springer International Publishing, Cham.

[109] Taillard, É. D. (2005). Tutorial : Few guidelines for analyzing methods. In *Metaheuristic Interantional Conference (MIC'05) proceedings*.

[110] Taylor, S. J. (2019). Distributed simulation: state-of-the-art and potential for operational research. *European Journal of Operational Research*, 273(1):1 – 19.

[111] Thabtah, F. and Cowling, P. (2008). Mining the data from a hyperheuristic approach using associative classification. *Expert Systems with Applications*, 34(2):1093–1101.

[112] Valipour, M. H., Amirzafari, B., Maleki, K. N., and Daneshpour, N. (2009). A brief survey of software architecture concepts and service oriented architecture. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 34–38.

[113] Van Hentenryck, P. and Milano, M., editors (2011). *Hybrid Optimization: The Ten Years of CPAIOR*, volume 45 of *Springer Optimization and Its Applications*. Springer, Berlin, Germany.

[114] Wagner, S. and Affenzeller, M. (2005). *Adaptive and Natural Computing Algorithms: Proceedings of the International Conference in Coimbra, Portugal, 2005*, chapter HeuristicLab: A Generic and Extensible Optimization Environment, pages 538–541. Springer Vienna, Vienna.

[115] Wagner, S., Kronberger, G., Beham, A., Kommenda, M., Scheibenpflug, A., Pitzer, E., Vonolfen, S., Kofler, M., Winkler, S., Dorfer, V., and Affenzeller, M. (2014). *Advanced Methods and Applications in Computational Intelligence*, volume 6, chapter Architecture and Design of the HeuristicLab Optimization Environment, pages 197–261. Springer.

[116] Weyland, D. (2010). A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology. *Int. J. Appl. Metaheuristic Comput.*, 1(2):50–60.

[117] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82.

[118] Woodward, J., Swan, J., and Martin, S. (2014). The 'Composite' Design Pattern in Metaheuristics. In *GECCO '14 Companion*, New York, USA.

[119] Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.

Appendix A. The 'Automated Open-Closed Principle'

Existing metaheuristic implementations handle *environmental state* in different ways. Many make *ad hoc* use of non-local variables to share information between components. This runs counter to automated assembly requirements: in order to present them to a configuration tool, such dependencies must be specified manually, i.e. the configuration space cannot be derived from the implementation automatically. That being said, some implementations treat environmental state in a more principled manner (e.g. [27, 17]). Here, metaheuristics are typically described as compositions of generically typed, stateful components having a well-defined interface controlling access to the encapsulated state. A framework then explicitly passes the shared state between subordinate components. While it has been demonstrated in prior-art that such implementations can be successfully coupled to configuration tools to perform bottom-up automated assembly [65, 104] *in the small*, they do not support the open-ended extension we propose is required *in the large* (as explained in Section 3.1).

In software engineering, a framework which can be configured from an open-ended palette of components while remaining unchanged is said to conform to the "Open-Closed principle" ('a framework should be *closed* to modification, but *open* to extension by new components'). We have extended this principle [106] to incorporate the behavior required to support automated design, yielding the "Automated Open Closed Principle" (AOCP); that paper discusses the issue in greater technical detail and describes experiments with a suitably equipped algorithm configurator. The adoption of the AOCP provides open-ended reuse of components, and thus a systematic approach to the automated exploration of the metaheuristic design space. A key aspect is that components must be "pure functional", as a first approximation[15] this can be interpreted as meaning:

- They do not rely on hidden state.
- For the same argument, they always return the same result.

Note that aforementioned metaheuristic implementations using global variables and / or stateful components clearly violate the AOCP. An alternative to *state encapsulation* is the use of *state threading* [53, 68], as can be seen in Listing 2 where `perturb`, `accept`, and `finished` each take an additional parameter that is used to *thread* the environmental state through the search algorithm. More generally, using state threading, the desired signature for `accept` is of the form:

$$accept_{Sol,Env} : Sol \times Sol \times Env \to Sol \times Env$$

However, manually threading the state through the algorithm (as in Listing 2) of Section 3 is error prone, since the framework implementer must ensure that the correct value of the state is passed to the correct stage of the algorithm. It is instead desirable to use a mechanism that *implicitly* performs state threading in a *well-defined* and *consistent* manner.

In functional programming, the problem of state propagation is addressed via a well-known design pattern: the *State monad*. For the purposes of this article, we can simply consider a monad to be a principled means of sequencing computations whilst abstracting over possible side-effects (in this case, state manipulation).

Functional languages such as Haskell and Scala provide syntactic sugar for monads. In particular, they allow monad operations to be chained together using syntax

[15]The interested reader is referred to the wealth of literature on 'referential transparency'

```
type Perturb[Env,Sol]    = Sol => State[Env,Sol]
type Accept[Env,Sol]     = (Sol,Sol) => State[Env,Sol]
type IsFinished[Env,Sol] = Sol => State[Env,Boolean]

class LocalSearch[Env,Sol] {

  def apply[Sol](incumbent : Sol,
                 perturb : Perturb[Env,Sol],
                 accept : Accept[Env,Sol],
                 finished : IsFinished[Env,Sol]) : State[Env,Sol] = {
    def until(s : Sol) : State[Env,Sol] = {
      for {
        perturbed <- perturb(s)
        accepted <- accept(s, perturbed)
        c  <- finished(accepted)
        result <- if (c) {
                    State.pure[Env,Sol](accepted)
                  } else {
                    until(accepted)
                  }
      } yield result
    }

    for {
      result <- until(incumbent)
    } yield result
  }
}
```
Listing 3: Local Search in Scala with State monad

that looks like a traditional *for* loop. This is illustrated in Listing 3, a re-formulation of our local search example in Listing 1 that uses the State monad. The state, in this case an integer representing the number of iterations, is implicitly threaded through each stage of the computation. This can be extended to yield a re-formulation of Listing 2, by defining that `Env` allows access to `[Sol]`, the search trajectory. Although the algorithm in Listing 3 looks similar to its imperative counterpart, the internal state is fully encapsulated within the definition of `LocalSearch`.

By virtue of open-ended support for state dependencies, the proposed approach therefore supports bottom-up automated assembly. Such an approach is less subject to human bias than the *a priori* prescription of a particular metaheuristic and therefore has relevance to foundational knowledge discovery efforts. In other areas of design (e.g. manufacturing), standardization has allowed a shift from the design of integrated systems to the design of individual components within the system. In metaheuristics, this reflects the natural trend for incorporating specialized problem- or solution- domain knowledge, i.e. a researcher can specialize in a particular kind of component such as acceptance criteria and determine their cross-domain ubiquity.

It might be thought that a monadic workflow requires metaheuristic researchers to become expert functional programmers, so it should be emphasized that this workflow is a consequence of our proposed formulation, rather than a mandatory aspect. In particular, the intention is that core metaheuristic templates can be written monadically 'once and for all', allowing non-expert users of these frameworks to obtain the benefits.

Another minor but pleasing property is that the explicit denotation of state makes the parameter space of a component explicit, facilitating configuration via automated tools such as *Irace* [59].

This pure functional perspective also provides a number of other advantages [43], of particular relevance to large-scale and automated design of metaheuristics: they make it easy to reason about behavioural equivalence and coupling between components, hence improving transparency. Determinism and lack of side-effects yields reproducibility of behavior. Furthermore, a functional treatment of metaheuristics greatly facilitates architectures which can take advantage of abundant computing resources, e.g. thread-safe parallelism [39] or 'Service Oriented Architecture' (SOA) implementation via stateless web-services, as subsequently described in Section 3.3.

To our knowledge, the first proposed use of monads for state threading in metaheuristics was as part of the MitL initiative [107], whilst the first concrete implementation subsequently appeared in CILib [76] and has since been further developed [77]. Although there have been many frameworks and publications that describe 'modular decompositions' of metaheuristics, to the best of our knowledge only MitL and CILib employ this principled approach to open-ended state dependencies. The additional contribution of the MitL initiative in this respect is the use of the monadic approach to explicitly support automated assembly [106]: of particular value in this respect is the fact that a strict type-system can be used to discriminate between stateless and stateful operations and to provide information about *which* aspects of component behaviour contribute to solution quality, this being vital for the elimination of accidental complexity.

Appendix B. MitL Software Libraries

Realizing the MitL vision of community-level research based on shared scientific infrastructure requires the development of three central building blocks:

1. Support for modular, extensible metaheuristic frameworks.
2. Machine-readable descriptions of problems, heuristic components and results.
3. A two-tier architecture defining both *Programmatic* and a *Service Oriented* interfaces, the latter being in direct correspondence with the former.

Taken together, these building blocks provide necessary support for the construction of a community knowledge base, in which fixed 'reference versions' of metaheuristic templates can be configured with problems and components in an open-ended manner. Although the main purpose of this this paper is to describe the motivation and vision for MitL, the project has nonetheless made concrete implementation progress. The MitL repository (`https://github.com/MitLware`) contains various software libraries providing infrastructure support, together with a number of examples of how the proposed approach can be applied in practice. The infrastructure support libraries are:

- `MitLware-java`
  This library contains Java interfaces for the 'Metaheuristics in the Large' components (Perturb, Evaluate etc), as motivated by the discussion in Section 3.1.
- `mitl-support`
  This library contains general utilities, metaheuristic-specific and otherwise. The former includes random selection and sampling.

- `mitl-problem`
  Example problem domains, defined in terms of the `MitLware-java` interfaces. The problem domains include: various bitvector problems, such as Checkerboard, Royal Road, Trap and HIFF; blocksworld; tower of Hanoi; the Iterated Prisoner's Dilemma; Magic Square; the n-puzzle; the De Jong suite of real-vector problems; SAT; the TSP; the Travelling-Thief Problem; Windfarm placement.
- `mitl-solution`
  Representations for ubiquitous candidate solution types (e.g. permutations, bit vectors and polynomials), as used in `mitl-problem`.

Although the following examples happen to be implemented in Java/Scala, adoption at the Service Oriented Architecture level means that components written in other languages can nonetheless interoperate via standard serialization protocols (such as JSON or XML). For example, either or both of client or server in `mitl-soa-example` could be written in any language, as long as it is capable of serializing candidate solutions in JSON. The example applications include:

- `mitl-whitebox-hyper-heuristics`
  As an elementary example of the approach described in "A Re-characterization of Hyper-Heuristics", this demonstrates a whitebox analog of the hyflex hyper-heuristic framework which takes as input any problem domain (examples used are SAT, bin-packing, TSP, VRP) generically described via the XCSP constraint programming format. It then uses heuristic pattern matching to determine if the problem constraints are isomorphic to the TSP: if so, then the problem is rewritten on the fly to TSPLib format and a dedicated TSP solver (Concorde's 'LINKERN' [6]) is used, if not then the generic Choco Solver [87] is invoked instead.
- `mitl-soa-example`
  This provides a concrete demonstration of the 'two tier architecture' described above: the MitL component interfaces defined in `MitLware-java` are 'lifted' to the service level via RPC (Remote-Procedure Call) support. There is thus a 1-1 correspondence between local and remote component interfaces. A metaheuristic framework can therefore be transparently configured with components that happen to be hosted remotely. A simple client-server example is provided, with remote invocation of a perturbation heuristic via json-RPC. The server-side implementation of perturb is actually achieved via a constraint solver, thereby giving another example of how one may freely mix between analytic 'OR-style' and empirical 'metaheuristic-style' approaches.
- `mitl-ecj-jmetal-interoperability-example`
  The ECJ [61] and JMetal frameworks [29] are both popular and widely used. However, it is not an easy task to achieve interoperability between them. This application shows how both can be represented as a MitL `Perturb` operator, allowing either to be interoperably invoked.
- `mitl-aocp`
  This provides an example application of the our proposed *Automated Open-Closed Principle* to automated algorithm configuration of the Traveling Salesperson Problem over a fixed algorithm framework. It uses ant-programming as a generative hyper-heuristic [54] to automatically configure a local search framework with components which have different state dependencies. Further technical specifics of enabling communal research via extensible algorithm templates are described in detail in Swan et al. [106].

- `mitl-hyperion`
  This provides extensible algorithm templates for several of the evolutionary algorithms described in ‘Essentials of Metaheuristics’ [61], as combinatorially instantiated in `mitl-aocp`, above.