Handbook of Research on Emergent Applications of Optimization Algorithms

Pandian Vasant
Universiti Teknologi Petronas, Malaysia

Sirma Zeynep Alparslan-Gok
Suleyman Demirel University, Turkey

Gerhard-Wilhelm Weber
Middle East Technical University, Turkey

A volume in the Advances in Business Information Systems and Analytics (ABISA) Book Series

IGI Global
DISSEMINATOR OF KNOWLEDGE

Published in the United States of America by
IGI Global
Business Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Vasant, Pandian, editor. | Alparsian-Gok, Sirma Zeynep, 1980- editor. | Weber, Gerhard-Wilhelm, editor.
Title: Handbook of research on emergent applications of optimization algorithms / Pandian Vasant, Sirma Zeynep Alparsian-Gok, and Gerhard-Wilhelm Weber, editors.
Description: Hershey, PA : Business Science Reference, [2018] | Includes bibliographical references.
Identifiers: LCCN 2017013837| ISBN 9781522529903 (hardcover) | ISBN 9781522529910 (ebook)
Subjects: LCSH: Industrial engineering--Mathematics. | Engineering mathematics. | Mathematical optimization. | Algorithms.
Classification: LCC T57 .H35 2018 | DDC 620.001/5196--dc23 LC record available at https://lccn.loc.gov/2017013837

This book is published in the IGI Global book series Advances in Business Information Systems and Analytics (ABISA) (ISSN: 2327-3275; eISSN: 2327-3283)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 1
Metaheuristic Design Patterns: New Perspectives for Larger-Scale Search Architectures

Krzysztof Krawiec
Poznan University of Technology, Poland

Christopher Simons
University of the West of England, UK

Jerry Swan
University of York, UK

John Woodward
University of Stirling, UK

ABSTRACT

Design patterns capture the essentials of recurring best practice in an abstract form. Their merits are well established in domains as diverse as architecture and software development. They offer significant benefits, not least a common conceptual vocabulary for designers, enabling greater communication of high-level concerns and increased software reuse. Inspired by the success of software design patterns, this chapter seeks to promote the merits of a pattern-based method to the development of metaheuristic search software components. To achieve this, a catalog of patterns is presented, organized into the families of structural, behavioral, methodological and component-based patterns. As an alternative to the increasing specialization associated with individual metaheuristic search components, the authors encourage computer scientists to embrace the 'cross cutting' benefits of a pattern-based perspective to optimization algorithms. Some ways in which the patterns might form the basis of further larger-scale metaheuristic component design automation are also discussed.

DOI: 10.4018/978-1-5225-2990-3.ch001

BACKGROUND

In recent years, modern optimization algorithms have attracted a growing number of scientists, decision makers and practitioners. Indeed, powerful intelligent computational techniques such as metaheuristics have emerged for solving a vast number of complex real-world problems, exploiting both optimization theory and practice. Increasingly, metaheuristic optimization techniques offer generic, flexible, robust, and versatile frameworks for solving complex problems of optimization and search in real-world application areas such as economics and engineering. Many metaheuristics - evolutionary algorithms, particle swarms, ant colonies, to name a few - are population-based, which makes them particularly robust and applicable to a diverse range of application domains. Nevertheless, as is the case with many other algorithms, tailoring the design of larger-scale metaheuristic search techniques, components and frameworks can be complex and decidedly non-trivial and may raise cross-cutting concerns that are critical for system performance.

Previously, concerned with a diversity of buildings' architecture, Alexander (1979) advocated a 'timeless way of building', drawn from thousands of years of traditional construction. Central to this approach is a 'quality without a name': good architectural quality is something that can be recognized, but difficult to describe in words, i.e. 'you know it when you see it'. Alexander suggested that this quality of a design could be captured in terms of patterns such that *"*each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution in such a way that you can use this solution a million times over, without ever doing it the same way twice". (Alexander et al., 1977, p. x).

Inspired by this notion of 'heuristics at a broad, architectural scale', Gamma, Helm, Johnson, & Vlissides (1995) applied design patterns to software design, an approach which revolutionized software development. Defined as "... descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*"*, they proposed that design patterns in software should comprise four essential elements: a pattern name, a design problem, a design solution and the consequences, i.e. the results and tradeoffs of applying the pattern. According to Gamma et al., "These patterns solve specific design problems and make object-oriented designs more flexible, elegant, and ultimately reusable" (Gamma et al., 1995, p. 1). Design patterns prompted widespread change in the accepted practice of software design, leading to component descriptions at a more abstract architectural level. Subsequently, pattern-oriented software architectures have been proposed, which *"*…present, discuss and contrast and relate the many known flavors and applications of the pattern concept: stand-alone patterns, pattern complements, pattern compounds, pattern stories, pattern sequences, and … pattern languages". (Buschmann, Henney, & Schmidt, 2007, p. xxxii).

Applying design patterns offers many benefits. For example, reuse of successful designs and architectures is easier, since *"*expressing proven techniques as design patterns makes them more accessible to developers of new systems*"* (Gamma et al., 1995, p. 2). To address the complexity and associated non-trivial issues of designing and applying metaheuristics, this chapter advocates a pattern-based perspective on optimization architectures, with a particular focus on the construction of larger scale frameworks.

Firstly, the motivation for metaheuristic search design patterns is discussed and the format used is described (Motivation section). A catalog of a dozen patterns is then presented, organized into structural, behavioral, methodological and component-based categories (in their own sections respectively). Looking towards possible future directions for metaheuristic design patterns (in the section Future Direction), we

speculate on the possibilities for collaborative infrastructures based on greater component-based design automation and knowledge discovery, followed by some closing remarks.

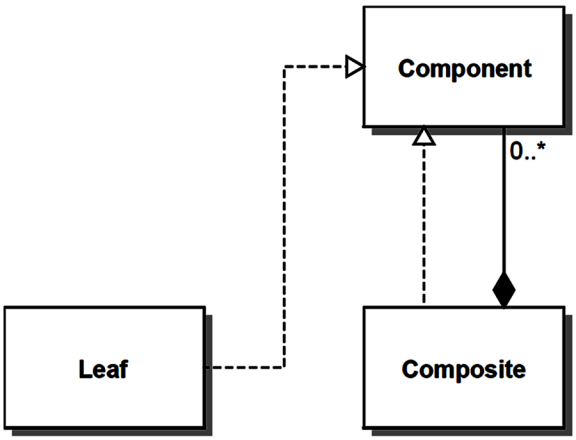
MOTIVATION

A traditional decomposition of metaheuristic optimization component architectures could be considered 'vertical', in the sense that much activity tends to cluster around individual bio-inspired frameworks (Genetic Algorithms (GAs), Particle Swarm Optimization (PSO), etc.), each of which tends to direct design into increasingly specialized and distinct 'silos'. Recent instances of the specialist application of such 'vertical' bio-inspired frameworks for optimization are plentiful in the literature. For instance, GAs have been applied to problem domains ranging from small enterprise default prediction (Gordini, 2014), to variable selection in multiple linear regression (Trejos, Villalobos, & Espinoza, 2016), to vehicle routing problems (Frutos, Tohme, & Miguel, 2016), and to rigid space interpretation (Singh, 2017). PSO has been utilized in domains ranging from the identification of melanoma skin lesions (Popa, 2014) to the optimization of manufacturing process parameters (Majumder & Majumder, 2015), to supply chain network design (Yuce & Mastrocinque, 2016), and to optimal power flow problems (Polpresert, Ongsakul, & Dieu, 2016). In addition, various applications of frameworks inspired by bee colony behavior have emerged (e.g. Yuce, Mastrocinque, Packianather, Lambiase, & Pham, 2015), and one recent example can be found in relation to antenna design problems for RFID applications (Goudos, Siakavaya, & Sahalos, 2016). Furthermore, recent examples of cuckoo search applications include engineering optimization (Ong & Kohshelan, 2016) and hydrothermal scheduling (Nguyen & Vo, 2016).

However, a significant consequence of the application of distinct bio-inspired frameworks for search and optimization is the fragmentation of knowledge and expertise within the metaheuristic design research community. Specialized knowledge and expertise can be confined within the bounds of a particular framework, and so it can be difficult to rigorously evaluate bio-inspired mechanisms from two different 'silos'. Fortunately, it is often possible to find perspectives on these mechanisms which are common across the 'silos'. We therefore believe that the wider interest lies in identifying such common aspects - not only to prevent duplication of research effort, but also to more widely communicate and advance metaheuristic design knowledge across the field. By adopting a pattern-based perspective, it is possible to describe common metaheuristic concepts as abstractions, and decompose such recurring aspects 'horizontally', thus describing such metaheuristic design abstractions in terms of a 'pattern language' (see Krasnogor, 2009). For example, the pattern format lends itself well to describing heuristics for component selection/generation: an essential part of the automated design of algorithms. In this respect, it is interesting to note that two of design patterns popularized by Gamma et al., viz 'Template Method' and 'Composite' are of general use in metaheuristics: applications are given in the Structural Patterns and Behavioral Patterns sections respectively.

In addition to the benefits described, we also believe that the architectural design of metaheuristic components would greatly benefit from an injection of techniques well-understood in other domains e.g. the incorporation of principled methods for statistical testing and design of experiments, and an enlarged vocabulary of proven reusable components/architectures. To this end, describing pattern-based abstractions of good 'horizontal' practices would not only make metaheuristic design more accessible and reusable, but also provide an educational aspect for novice researchers wishing to exploit proven approaches.

Figure 1. UML class diagram of the 'composite' diagram



Last, but not least, pattern languages make it easier to detect similarities between the methods developed independently in particular frameworks, and help understanding their mutual relationships. In particular, it should help reduce the proliferation of seemingly novel techniques that often ultimately turn out to be other algorithms in disguise of a bio-inspired metaphor - the phenomenon aptly summarized in Sörensen (2015).

As is the usual practice in the software design patterns, the structure of patterns is illustrated where appropriate in this chapter via UML diagrams (Fowler, 2004). As an example of the notation, the ubiquitous 'Composite' pattern is shown in Figure 1. The dotted line indicates a 'subtype' relationship: 'Leaf' and 'Composite' are specialized types of 'Component'. The line headed with a black diamond indicates that 'Composite' contains zero or more components.

Despite such notation having its roots in software development, it is important to emphasize that the pattern descriptions we present here are in the spirit of Alexander's original vision, i.e. conceptual abstractions of design solutions, rather than necessarily being descriptions of implementation detail at the source code level. The initial application of design patterns to metaheuristic design is due to Krasnogor (2009), who proposed a pattern-language to capture the various sensibilities involved in designing memetic algorithms, as well as introducing versions of the 'Template Method' and 'Surrogate' patterns (variants of which are described in this paper).

Pattern Description Format

Following the work of previous pattern communities (Alexander, 1979; Gamma et al., 1995) and pattern languages (Alexander et al., 1977; Krasnogor, 2009), we describe our patterns in a manner we believe appropriate for the metaheuristic design. The format encourages consistency of structure and comprises the following parts:

- **Name:** The pattern's name succinctly conveys the essence of the pattern. The naming of a pattern is important as it increases the design vocabulary, promoting shared understanding.

- **Problem:** The pattern's problem context illustrates the situation in which the pattern is best applied and can be described by its *forces acting,* i.e. the preconditions for the application of the pattern. It is typical for many, potentially conflicting, forces to be acting. Forces acting may also present as problem constraints to successful pattern application. The *intent* of the pattern conveys the goal behind the pattern, and how it addresses the problem context.
- **Solution:** A description of the abstract solution mechanism of the pattern, together with any appropriate structure. The solution description reveals the structural components of the solution mechanism and where appropriate, behavioral interactions between instances of the components.
- **Consequences:** The results and tradeoffs of applying the pattern. Typically, the forces acting influence the tradeoffs between various design alternatives, and the consequences of pattern application may be crucial for evaluating competing designs.
- **Examples:** Concrete representative instances of the pattern, possibly coming from different 'vertical' frameworks mentioned in the beginning of this section, to illustrate the pattern's universality.

Catalog Organization

To assist navigation, metaheuristic design patterns are classified into families of related patterns. Previous pattern catalog taxonomies include that of Gamma et al. (1995) (the so-called 'Gang of Four' or 'GoF' patterns) wherein software design patterns are structured as follows:

- **Creational:** Concerned with the process of creation and generation
- **Structural:** Concerned with the structure of components and their composition.
- **Behavioral:** Concerned with dynamic interaction/division of responsibility.

However, subsequent pattern literature notes that the above occupies the middle tier of a more general description (Buschmann et al., 1996):

- Architectural Patterns (e.g. at the 'framework' level)
- Design Patterns (e.g. 'GoF' patterns, at the level of domain-agnostic collections of concepts)
- Idioms (e.g. patterns phrased directly in the language of the target domain)

Because metaheuristic design patterns operate in a more specialized domain than software engineering, we can reasonably expect to cover many of the necessary design sensibilities by grouping patterns into families related to the purpose of the pattern i.e. reflecting what the pattern achieves. According to this notion, we divide the metaheuristic design patterns into structural, behavioral, methodological and component-based:

Structural patterns provide mechanisms for (semi-)automated assembly of algorithms. The examples of structural patterns given provided in this paper are:

- A Template for Hybrid Iterated Local Search
- Template Method Hyper-Heuristics
- Composite

Behavioral patterns are concerned with the dynamic interaction/division of responsibility and include:

- Blackboard
- Structural Stigmergy
- Tagging

Methodological patterns orchestrate the metaheuristic workflow in a specific manner that is (broadly) known *a priori* to the experimenter. Examples include:

- Executable Experimental Template
- Interactive Solution Presentation

Component-based patterns are concerned with generic notions of operators, evaluation or candidate solutions. Examples include:

- Solution Repair
- Surrogate
- Subjective Preference

The spectrum of useful patterns is of course completely open-ended. We therefore aim to sketch the landscape of possible designs and point out interesting avenues. For that reason, we believe that metaheuristic design patterns should encompass notions that are well-rooted in metaheuristic research and practice (e.g., Surrogate and Solution Repair) as well as selected patterns that are less 'obvious' and/or less frequent, like Blackboard (Graham, Swan, & Martin, 2015) and Representative (Swan, Kocsis, & Lisita, 2014). Each family of related patterns is described in the following sections.

STRUCTURAL PATTERNS

A Template for Hybrid Iterated Local Search

Problem

The *intent* of this pattern is to distil into a template the accumulated knowledge of which aspects of single-solution metaheuristics that work well.

Single-solution metaheuristics are popular and ubiquitously effective. They include classical local search algorithms, in which an algorithm starts from a single candidate solution and iteratively moves to some neighboring solution. When no improvements are possible in the immediate neighborhood, local search becomes trapped at a local optimum. To address this, mechanisms such as restarts or jumps across the search space may be applied e.g. in Iterated Local Search (ILS) and variable neighborhood search (VNS). Other mechanisms include varying perturbation strength according to some temperature schedule as in simulated annealing (SA). For real-valued search spaces, single-solution evolution strategies e.g. (1+1)-ES, (1,1)-ES are effective.

However, in terms of *forces acting*, a problem exists in that algorithms of a similar approach tend to be named differently according to their "vertical silo". For example, iterated greedy (IG) can be similar to large-neighborhood search (LNS). This can inhibit knowledge sharing among researchers and makes it difficult to discern the best algorithm for a given problem.

Solution

The Template for Hybrid Local Search pattern by López-Ibáñez, Mascia, Marmion, & Stützle (2014) addresses this problem by focusing on the common components of single-solution iterated local search. The common components of this pattern include perturbation, local search, an acceptance criterion, and a termination criterion. Each component may influence the state of the search. An outline of the template pattern, as suggested in López-Ibáñez et al., (2014), is given in Figure 2.

Components of the template pattern are:

- **Perturbation:** i.e. a stochastic move within the global neighborhood. Perturbation may actually involve multiple moves in the search landscape or even a change of neighborhood structure (e.g. VNS). Perturbation may have an associated strength parameter, used heuristically to control the tradeoff between intensification (exploiting the current local structure of the search space) and diversification (e.g. rejecting solutions with previously-encountered attributes, as in tabu search).
- **Acceptance Criterion:** i.e. a procedure to select between incumbent and incoming solutions. Some acceptance criteria (e.g. Metropolis-Hastings as used in Simulated Annealing) may actually accept worsening moves.
- **Local Search:** i.e. any procedure that starts from a candidate solution, and iteratively moves to a superior neighbor solution. As the local search can itself be an ILS, this facilitates hybrids of different types of single solution metaheuristics. Defining hybrid ILS algorithms in this way has been described previously in Vaessens (1998).
- **Termination Criteria:** i.e. a function based on state progress, returning 'true' when the main loop should stop. Termination may be based, for example, on a time limit, a given number of iterations, or a number of iterations without improvement in the candidate solution.

Figure 2. Hybrid Iterated Local Search (ILS) pattern template

ILS Algorithm

1: s_0 := Initialization()
2: s^* := ILS(s_0)
3: **return** s^*

Function ILS(s_0)
Require: Perturbation, LS, Acceptance, Termination
1: s^* := LS(s_0, $State$)
2: **repeat**
3: s' := Perturbation(s^*, $State$)
4: s' := LS(s', $State$)
5: s^* := Acceptance(s', s^*, $State$)
6: **until** Termination($State$)
7: **return** s^*

Consequences

This pattern enables Iterated Local Search to be described at an empirically-useful useful level of granularity, based on the common software components used. The pattern also offers flexibility in terms of the substitutability and configuration of its components, including how many levels of ILS are appropriate to the problem at hand. A further consequence is that the pattern has also enabled the generation of code from a grammatical description of the template (Marmion, Mascia, López-Ibáñez, & Stutzle, 2013). A fuller description of pattern languages as grammars is discussed in a later section.

Examples

Taking Iterated Greedy-Variable Neighborhood Search (IG-VNS) as an illustration of hybrid ILS, its name fails to precisely convey the components used in what combination. However, using the pattern template, a number of concrete possibilities exist. For example:

- A two-level ILS that uses Iterated Greedy at an outer level combined with a Variable Neighborhood Search for the inner level. Perturbation at the inner level ILS comprises destruct/reconstruct jumps. However, the strength of perturbation in the outer loop increases if a new best-so-far solution is not found.
- Iterated Greedy with increasingly strong destruct/reconstruct jumps. This resembles single level ILS, with destruct/reconstruct perturbation. However, perturbation strengthens when no new best-so-far solutions are found.
- Iterated Greedy with a subordinate Variable Neighborhood Search. This resembles a two level ILS, although the perturbation of the outer level is destruct-reconstruct with the inner level classic VNS.

Template-Method Hyper-Heuristics

Problem

The intent of this pattern is to enable the generation of customized versions of pre-existing algorithms. The forces acting are:

- When it is possible to explicitly describe a family of algorithms in terms of their similarities and differences.
- When the target algorithms are larger than algorithms typically evolved using a ‘bottom up’ approach by e.g. Genetic Programming. The template provides a top-down structure which acts as a context in which the outputs from (potentially multiple) ‘bottom up’ can be orchestrated.

One of the significant motivators for the adoption of this pattern in research practice is cultural: metaphorically-inspired metaheuristics have saturated the optimization literature (Sörensen, 2015). These methods are often described using language from the domain of inspiration (e.g. the behavior of cats or jazz musicians) rather than the more neutral and objective language of mathematics and computer science (Woodward & Bai, 2009a). Such *ad hoc* descriptions make it difficult to communicate and compare

metaheuristics. In addition, much research is incremental, involving only relatively small changes to existing algorithms. This manual process of 'tweaking' a metaheuristic is labor intensive, and contributes little understanding to how these algorithms perform. Rather than looking to metaphors for inspiration, we can instead use hyper-heuristics to generate algorithmic components which plug into a template.

Solution

The well-known 'Template Method' pattern (Gamma et al., 1995) defines an algorithmic 'skeleton' (i.e. a fixed framework) which accommodates a collection of components. The framework prescribes the behavior of these components, and within these constraints, various components are substituted. Examples include frameworks for evolutionary computation (EC), where the components include selection, recombination (crossover) and mutation operators.

This pattern combines the original 'Template Method' design pattern (Gamma et al., 1995) with the approach of generative hyper-heuristics (Burke et al., 2010). The variant components can be generated using methods such as GP (Koza, 1992). This search-based approach enables an enormous number of variations of components to be automatically generated, and evaluated against a test set, to obtain good performance on unseen problem instances.

In the particular context of generating metaheuristic components, this is of particular relevance as a consequence of the No Free Lunch theorems (Giraud-Carrier & Provost, 2005). Since no universally good optimizer exists, it is intrinsic that a metaheuristic is biased to a particular probability distribution of problem instances. By treating the automatic generation of metaheuristics as a supervised machine learning application, this pattern generates metaheuristics specialized to the distribution on which they are trained.

Consequences

Adopting a semi-automated approach to the design of algorithms, where the practitioner supplies the template, and the variant components are generated via search, has the following consequences:

- **Optimized to the Probability Distribution:** If the algorithm is generated in response to a set of problem instances drawn from a given probability distribution, the resulting algorithms can be expected to perform well on problem instances drawn from a similar probability distribution.
- **Decreased Human Effort:** In general, automation can reduce human cost.
- **Increased Training Time:** Fitness evaluation involves running the underlying algorithm (possibly multiple times if it is stochastic), which can be a highly expensive activity. If the space of hyper-parameters is large, then it may be imperative to use a principled approach such as the iterated racing of iRace (López-Ibáñez, Dubois-Lacoste, Marmainm & Stützle 2011) for sampling the parameter space.

Examples

The first two examples given are in the context of metaheuristic design, while the third is more general.

Two well-known GA mutation operators are one-point (which alters a single bit), and uniform mutation (which alters all bits with a fixed probability). As described above, a GA can be considered

as a framework parameterized by its mutation operator. In Woodward and Swan (2012), GP is used to automatically generate a mutation operator that outperforms both one-point and uniform mutation. Key to the success of this approach is a template which can readily express one-point and uniform mutation.

Another component that is key to a population-based metaheuristic is the selection operator. Two well-known evolutionary computation selection operators are fitness-proportional (which assigns a value in proportion to the absolute fitness of a solution), and rank- selection (which assigns a value according to the index of a solution in the sorted population). One generalization of these two selection operators is as a function *p(f, r),* where *f* is the absolute fitness and r is the rank index. This more general function can clearly reduce to either fitness-proportional or rank selection. The function *p* is generated by GP and yields a value for each member of the population. By this means, a new selection operator was obtained that outperforms both rank and fitness-proportionate selection (Woodward & Swan, 2011).

Beyond the application domain of metaheuristics, we can of course consider hyper-heuristics as operating in the space of *any* algorithm that has a heuristic component. The TEMPLAR framework (Swan & Burles, 2015) provides a generic implementation of the 'Template Method Hyper-Heuristic' pattern, giving a case study in which the heuristic operation of the 'pivot function' in the well-known Quicksort algorithm is optimized to reduce power consumption.

Composite

Problem

The intent of the Composite pattern (Gamma et al., 1995) is to treat a collection of objects exactly as if they were a single object of that type. Regarding *forces acting*, when attempting to describe complex systems (such as a metaheuristic with many interconnected components and/or multiple levels of search activity), it is useful to simplify their description (whether conceptually or in implementation terms) by providing a uniform terminology, such as afforded by the Composite pattern. The prototypical exemplar of a composite is a recursive data structure such as a tree, defined as being either a leaf node or else a node with a list of nodes as its children. Considered in object-oriented terms, both objects and collections (e.g. leaf and non-leaf nodes) are abstractly defined by the same interface.

The interface for metaheuristic components can often be expressed as a single function $perturb : S \rightarrow S$ for some solution representation S. Consider, without loss of generality, the class F of functions with the signature $A \rightarrow R$. The composite C of such functions is then a function:

$$C : \left\{F\right\} \rightarrow F \tag{1}$$

which takes a set of functions $\{F\}$ as input, and yields a function of type F. For, if we take S = {+, -,*,\} then *C* can represent rational functions. In the same manner, arbitrary combinations of metaheuristic components can be expressed in terms of composites by taking:

$$F = S \rightarrow S \tag{2}$$

Solution

The Composite design pattern (as introduced here in the Motivation Section) is ubiquitous in computer science. In metaheuristic design, it affords a unifying perspective that is of value because of:

- **Terminological Clarification:** The well-defined entity relationships allow for ease of communication.
- **Facilitated Automation:** A composite implicitly defines a 'grammar template' for the generation of new metaheuristic components (See the Future Directions Section for further discussion).

Consequences

We discuss below two consequences of this pattern - firstly regarding ensembles of classifiers, and secondly regarding hyper-heuristics.

In machine learning, classifiers are functions mapping a set of features to class labels (Mitchell, 1997). An ensemble classifier is a composite of base classifiers (Bishop, 2006). By taking a diverse set of base classifiers and aggregating their outputs, an ensemble is expected to be more accurate than any base classifier. The classifiers can be diversified by randomized training algorithms, re-sampling of training data (bagging), or iterative adjustment of 'hardness' of particular training examples (boosting). There are a number of popular methods for composing classifiers, including: majority vote; averaging; weighted-, algebraic- and learned- combiners.

The theoretical underpinnings behind the success of classifier ensembles can be conveniently phrased using the eponymous bias-variance tradeoff (Geman, Bienenstock, & Doursat 1992). Bias represents the learner's inherent propensity toward certain realizations (models), while variance reflects the variability of model's predictive accuracy. These quantities are inseparable: a highly biased predictor tends to have low variance and vice versa. However, by aggregating multiple low-bias, high-variance predictors (a.k.a. weak classifiers), the variance can be reduced at no extra cost to bias. These properties make many classifier ensembles provably better predictors than individual base classifiers.

First introduced in a paper by Cowling, Kendall, and Soubeiga (2000), a commonly-used definition of hyper-heuristics is "heuristics for selecting or generating heuristics" (Burke et al., 2010). As this definition indicates, a hyper-heuristic can be seen as a metaheuristic (Sörensen, 2015), more specifically one that performs part of its search over programs spaces. One way of making this statement concrete is to show that a hyper-heuristic can be considered simply as the application of the Composite pattern to metaheuristics. We proceed to demonstrate this via a formal and generalized description of hyper-heuristics. Alternative descriptions are possible (see e.g. Swan, Woodward, Ozcan, Kendall, & Burke, 2014), but the simplified one given here makes the nature of the composition explicit.

Let S denote (candidate) solution type, i.e. S is generic, but can be instantiated with bit-string, or permutation, or without loss of generality, a population or Pareto-front of solutions. We define an operator 0_S as: $O_s : S \rightarrow S$ i.e. O_s takes a solution as input and returns a perturbed solution. We can, without loss of generality, consider a metaheuristic to have the same signature. For some solution type *T*, we then define a hyper-heuristic as a function with the type signature (Woodward, Swan, & Martin, 2014):

$$H_T : T \times \left[O_T\right] \rightarrow T \times \left[O_T\right] \tag{3}$$

i.e. H_T operates on a pair (Cartesian product) containing a solution and a list of operators. This formulation generalizes both selective and generative hyper-heuristics: the former may update only the solution, the latter the list of operators. By instantiating *S* as *T* x $[0_T]$ in (3), we can see that *H* presents the same signature as O, but in addition manipulates a list of O --- precisely the definition of a composite.

In the functional setting described above (Eq. 3), the Composite pattern has a very natural correspondence with methods for generating programs: *C* can be seen as being equivalent to a GP tree which generates a function of type *F* from a function set *S*. This is particularly easily expressed in the case of type consistency (Poli, Langdon, & McPhee, 2008), i.e. when the argument and return type of *F* are the same, but may require greater design consideration for the function set if they are different. More generally, every metaheuristic component (see Component-based Patterns Section) might be considered as a Composite of elementary components. For example, there are benefits to treating a surrogate fitness function (Figure 3) as an ensemble of simpler surrogates (Lim, Jin, Ong, & Sendhoff, 2010). Alternative composition methods might therefore be explored by combining the Composite pattern with 'Template Method Hyper-heuristics'.

Examples

- Decision trees tend to over-fit to the training data, due to the very effective partitioning of the data space (input variables). Random forests (Breiman, 2001) solve this dilemma by generating a number of decision trees which are then combined. This has successfully been employed in the Kinect motion-sensing device (Flach, 2012).
- Kittler, Hatef, Duin, & Matas (1998) give a composite scheme for combining classifiers which can reproduce a range of popular classifiers as a special case.
- The Hyperion framework (Swan el at, 2011), (Brownlee et al 2015) uses the Composite pattern to express both metaheuristics and (selective) hyper-heuristics using the same source code, via a composite representation of the neighborhood function.
- Tibermacine, Sadou, That, & Dony (2016) propose a 'reuse-by-composition' approach for software architecture constraints, wherein such constraints describe invariant properties that can be verified within an architectural design. Tibermacine et al. (2016) further suggest that architectural constraints can be composed from constraint instantiations and connections to customize architectural component descriptions.

BEHAVIORAL PATTERNS

Blackboard

Problem

It is well-known that domain knowledge is a vital aspect of metaheuristic performance. However, many traditional metaheuristic frameworks do not readily support the injection of arbitrary domain information. The particular effectiveness of techniques such as Tabu search (Glover, 1989) and squeaky wheel optimization (Joslin & Clements, 1999) in a wide variety of domains is largely due to the fact that both have generic mechanisms for the inclusion of domain knowledge. These generic mechanisms (i.e. tabu list and analyzer respectively) are accessed at specifically-orchestrated points in the flow of control. The intent of the blackboard pattern is to provide a unified framework for the injection of arbitrary domain-knowledge (and in consequence, solution representation) and decouple it from control flow. The inspiration is that of a group of experts collaborating via a shared workspace (the 'blackboard') to solve a problem that none are necessarily competent to solve individually. In terms of *forces acting*, blackboard architectures are applicable to problems which:

- Are not easily solved by a strategy which is fixed *a priori*.
- Are data-driven/opportunistic (i.e. can benefit from information that arises during the search process)
- Will benefit from heterogeneity, in respect of both a) solution representation and b) sources of domain knowledge and the granularity at which they are acquired and/or acted upon.

For metaheuristic purposes, it is particularly interesting to note that blackboard architectures have traditionally been used for 'hard' problems where it is often difficult to even define a 'top down' objective function a priori. In this sense, they can be seen as a specific architectural realization of the 'Structural Stigmergy' pattern (q.v.).

Solution

The original instantiation of the blackboard pattern is the Hearsay II system for speech recognition (Erman, Hayes-Roth, Lesser, & Reddy, 1980) where information in a variety of different forms is manipulated and integrated to form a solution. It is typical that communication between experts (a.k.a. 'Agents' or 'Knowledge Sources') takes place only via the workspace. This loose coupling means that agents can operate concurrently in an asynchronous fashion. Conceptually, the workspace can be considered as a dictionary of heterogeneous records, indexed by their features. For metaheuristic purposes, these features correspond to some specific form of domain knowledge or solution representation (e.g. in the case of the Travelling Salesman Problem (TSP), the ability to variously designate solution representations as permutations or graphs). Agents can register their competence to work with particular features and are given the opportunity to contribute when relevant information is placed in the workspace by other agents. Booch (1994) gives a motivating example of injection of domain knowledge with a cryptanalysis system for substitution ciphers in which different types of agent have competences at the different hierarchical levels of letter, word and sentence, with each level being driven by different information sources.

Consequences

This architecture is of particular relevance to hyper-heuristics, as characterized (via the 'Algorithm Selection Problem' (Rice, 1976), by 'a mapping from features to algorithms'. Hyper-heuristics were originally motivated by an attempt to provide generic cross-domain search strategies. However, the popular conception of selective hyper-heuristics as exemplified by the HyFlex framework (Ochoa et al., 2012) has a minimal cross-domain feature set (viz. the objective value obtained by the application of an operator, and the associated execution time) and has helped to propagate the widespread belief that this restriction is necessary for generality. As explained in detail by Swan et al. (2017), the range of useful cross-domain information is completely open-ended. To give a simple but concrete example, the notion of 'inverse of an operator' (e.g. as used in the 'Reverse Elimination Method' of tabu search (Glover, 1989)) can be exploited hyper-heuristically in a cross-domain manner without any loss of generality. By providing support for arbitrary features and agents with associated competences, the blackboard pattern offers an alternative to the unfortunately prevailing notion of an 'artificially high domain-barrier'. In addition, by virtue of concurrent and asynchronous activity of the agents, there is implicit support for integrating computations of varying process granularities. One area where this might be of value is the incorporation of fitness landscape metrics, an expensive activity that is traditionally performed offline. For example, the effectiveness of an agent which acts as a perturbation operator might be dependent on knowledge of the 'fitness-distance correlation' (Jones & Forest,1995): this information can be obtained as a background process, with the perturbation agent only acting when it is suitably informed. This pattern thus facilitates the interleaving of online and offline activities, rendering the distinction less important than is traditional. A negative consequence arising from the flexibility of de-centralized control is that it can be quite difficult to impose a very specific desired behavior on a blackboard architecture: fine-tuning of the conditions for agent applicability may be necessary, although this does generally have the advantage of being able to exploit domain-specific knowledge.

Examples

In addition to those given above:

- Martin, Ouelhadj, Smet, Berghe, and Ozcan (2013) demonstrate the effectiveness of a blackboard-based approach to nurse rostering.
- An asynchronous agent-based hyper-heuristic framework (Ouelhadj & Petrovic, 2010) offers superior performance over sequential hyper-heuristics on the permutation flow shop problem.
- Metaheuristics in the MAGMA system (Milano & Roli, 2004) are implemented via agent interaction, working at different hierarchical levels and granularities, with a variety of agent communication schemes including a global blackboard and message-passing.
- A blackboard architecture that uses a variant of the formulation given above to unify all the aspects of the hyper-heuristic design space, i.e. selective/generative, constructive/perturbative, online/offline is given in Swan et al. (2014).

Structural Stigmergy

Problem

In combinatorial problems, candidate solutions are compounds of interacting elementary entities, which can be conveniently represented as graphs (more generally as multi- or hyper-graphs). For such problems, domain knowledge may be available that may help identify the entities that are in 'wrong' relationships and ways of correcting such 'flaws'. For instance, in the travelling salesman problem (TSP), a part of a route may be obviously suboptimal and easy to improve by, e.g., swapping two cities. This can be attained by purely local considerations, without referring to the objective function. The *intent* behind the structural stigmergy pattern proposed in Kovitz and Swan (2014a) is to manipulate such graphs locally by multiple uncoordinated actions issued by search operators, or more generally agents. In terms of *forces acting*, this pattern is applicable whenever (i) domain knowledge allows defining the above 'flaws', (ii) means exist for 'marking' such flaws and so informing the other metaheuristic components about their presence, and (iii) methods for addressing the flaws can be designed.

Solution

Structural stigmergy relies on domain knowledge about the desirable and undesirable configurations of entities that form solutions. To begin, a notion of local flaw in a solution graph must be defined (e.g., unconnected wire in an electronic circuit). For each such a flaw, one provides (one or more) patch, i.e., a simple way of fixing it. By being simple and local, application of a patch may cause new flaws in a graph. In this design pattern, such 'flaw propagation' drives the search dynamics. Because single fixes may have no material impact on the overall evaluation and progress may require a series of fixes to be applied to a given part of a graph, every act of patching rises the local level of salience, which in turn increases the likelihood of other patches to be applied in a vicinity. Also, because the quality of a solution as a whole typically depends on non-local interactions, it is essential to detect and exploit the non-trivial properties of groups of nodes. To that end, the mechanism of tagging (Kovitz & Swan, 2014b) is introduced. In its simplest instantiation, epitomized by the tabu search algorithm (Glover, 1989), tagging is essentially the marking of visited solutions with 'been here' labels and prohibiting a re-visitation within a certain period. In general, tagging may involve decorating not only candidate solutions but also their components, and using arbitrarily complex tags rather than single symbols.

Note that all above mechanisms are agnostic about the aim of the search process; by contrast, we may employ specialized tags known as goal tags to express the desired properties of components with respect to the search goal (an objective function in optimization problems or a goal predicate in search problems). Goal tags signal flaws whenever the properties of candidate solutions do not meet the 'expectations' expressed by problem specification, and so impose a top-down pressure on the actions applied to a graph, guiding the search (albeit only indirectly, via interactions between flaws, fixes, and tags) towards the solutions with desired properties. If no progress is observed for certain time in a given part of a graph, it can be split into elements and processed again, with a likely different outcome (due to the patches and tags applied in the meantime to the surrounding context).

Consequences

The Structural Stigmergy pattern allows convenient expression of the expected properties of candidate solutions and inclusion of arbitrary amount of domain knowledge in a local and straightforward way. A human with basic knowledge of electronics may not know how to design a heterodyne, but may know how to avoid basic flaws and how to fix them. In this sense, stigmergy facilitates implicit problem decomposition and addresses the problem of credit assignment. Flaws, patches, and tags allow the designer to directly inject a means to 'criticize' the candidates and offer local means for improvement (or at least change), on the scale that usually escapes the conventional metaheuristics (which normally 'compress' all details on evaluation in a single value of scalar objective function). This can be seen as a generalization of the analysis phase of Squeaky Wheel Optimization (Joslin & Clements, 1999), which typically uses the less flexible notion of an a priori priority ordering.

Arguably the most important risk incurred by adoption of this pattern is that stigmergy may bias the search process in a way that is hard to predict in advance. The particular flaws defined by a designer and the particular means selected for fixing them may improve local 'architectural' properties of candidate solutions, but are in general not guaranteed to guide to an optimal solution (or even a solution of acceptable quality). Also, the more advanced realizations of stigmergy can be challenging in implementation.

Examples

The exemplar of Structural Stigmergy is the Copycat system (Hofstadter & Mitchell, 1995) capable of discovering and building sophisticated analogies between character strings. In particular, note that this is a domain in which it is very difficult to formulate an explicit 'top-down' objective function. Given an example of string transformation, Copycat produces an analogous transformation for another, possibly very different string. For instance, given that "abc" is transformed to "abd" and knowing the ordering of letters in the alphabet, Copycat will propose to transform "iijjkk" to "iijjll", "iijjkl", "iijjdd", "ijl" - outcomes that are strikingly convergent with the answers typically given by humans. To attain this aim, Copycat relies on stigmergy and tagging (see Tagging Section and (Kovitz & Swan, 2014a) for more details). Copycat can be also seen as a special case of a blackboard system (cf. Blackboard Section).

Among the commonly used metaheuristics, it is Ant Colony Optimization (ACO) that arguably best epitomizes stigmergy (Dorigo & Stützle, 2004). Stigmergy is here a resultant of multiple searches ('ants') attempting to construct new candidate solutions based on the knowledge accumulated in 'pheromones' left by previous such attempts. Importantly, this process is predominantly local, despite the fact that no explicit additional information on qualities of solutions' components is available: the acts of following the pheromone trails and constructing solutions gradually propagate the global information on solutions' overall quality to the level of solution components.

METHODOLOGICAL PATTERNS

Executable Experimental Template

Problem

The *intent* of this pattern is to provide a sound methodological basis for research practice by ensuring fair comparison between metaheuristics (Neumann, Swan, Harman, & Clark, 2014). The motivation to incorporate this pattern into the research workflow arises from the following *forces acting*:

- For purposes of clarity and rigor, it is desirable to have a baseline methodology for statistical testing, the need for which has been apparent for some time (Hooker, 1994).
- Determining the appropriate statistical test for fair comparison of a given collection of datasets is non-trivial (Arcuri & Briand, 2014).
- Commonly-used statistical tests such as the t-test have preconditions (e.g. normality) which do not hold in general for distributions arising from the application of randomized algorithms (Arcuri & Briand, 2011).

Solution

The solution is to provide a 'template method' framework (Gamma et al., 1995) for orchestrating the gathering of data-points and the performance of statistical tests on them. Creating a framework which is re-usable in a wide range of experimental contexts immediately brings computational efficiency concerns to the forefront, e.g. data-point generation might be very expensive (e.g. it might be obtained from an entire run of a metaheuristic); the runtime of some statistical tests can increase very rapidly, even for small numbers of data-points.

The need for an explicit template then arises because:

- It is important to explicitly order tests by computational expense;
- With sufficiently many data-points, statistical significance can arise even when experimental treatments differ to only a very slight degree.

For both of these reasons, the template should ideally be designed to ask for further data-points (i.e. performing an 'executable experiment') only if more are a prerequisite for a given test.

Consequences

The consequences of adopting a statistical testing framework into the workflow include:

- No requirement for researchers to have detailed knowledge of empirical methods, since the necessary expertise is embedded within the template. This is of particular importance with respect to some of the more philosophically-subtle aspects of statistical testing (Neumann, Harman, & Poulding, 2015).
- Provides standardized and documented research practice. Consistency of statistical testing is clearly highly desirable, but many cases in recent study (Arcuri & Briand, 2011) were found to be methodologically flawed (e.g. with insufficient data-points or inappropriately-applied tests). Of course, obtaining complete consensus on the validity of a given testing framework across the empirical methods community may never be possible, but by providing a concrete basis for further discussion and improvement, this will still yield a much-needed improvement on the current state of affairs.
- Increased experimental reproducibility. An explicit template structure makes it easier to audit experimental results. For example, when the most appropriate statistical test is a function of the degree of normality, logging the resulting normality score ensures that all information relevant to statistical testing is made available for publication.

Examples

Neumann et al. (2014) describe Astraiea, an open source framework for principled statistical testing (available at https://goo.gl/VcD8lh). Notably, the framework supplements statistically significant p-values with an effect size measure, in order to give a grounded measure of the difference between the two metaheuristics (Arcuri & Briand, 2011). Weise et al. (2014) describe a framework for detailed investigation of TSP solver performance, and argue that analysis of the entire solution trajectory is necessary for meaningful comparison. An expert system for generalized hypothesis testing is presented by Hathaway (2013) using decision-tree mediated user-interaction to guide statistical testing.

Interactive Solution Presentation

Problem

Interactive metaheuristic search incorporates 'human-in-the-loop' into the search process, whereby the human (wholly or partially) replaces the evaluation function by providing feedback at intermediate points (Takagi, 2001). However, it is challenging to present solutions in such a way that the human can supply effective evaluation. In particular, a large number of manual evaluations can cause user fatigue, resulting in varying levels of attention and inconsistencies. Given these *forces acting*, the *intent* of the interactive solution presentation pattern is to keep user participation focused by reducing the number of interactions.

Solution

The Interactive Solution Presentation pattern (Shackelford & Simons, 2014) reduces the load on the human evaluator by, for example, selecting a limited number of solutions from a population, or not asking the human to evaluate at every iteration.

Methods for selecting a subset of solutions include banded presentation, cluster-representative presentation, surrogate presentation, and rank-based presentation. In banded presentation, the population is

partitioned, and one solution from each partition is presented to the user. The evaluation of one solution in a partition is then used to assign a fitness to all solutions in that partition. In cluster-representative presentation, clustering algorithms are used to identify similar solutions which are representative of those in a neighborhood. These representative solutions are then presented to the user. Fitness is assigned to solutions in the neighborhood as a function of their distance from the human-evaluated solution. In surrogate presentation, 'quick and dirty' quantitative measures are applied as surrogates of user evaluation to eliminate poor-quality solutions. Rank-based presentation is an example of a mechanism to effectively reduce population size, in which a subset of solutions is ranked by the user, and their fitness is assigned based on their ranking.

Examples of intermittent human evaluations include fixed presentation interval and fitness proportionate presentation interval. In fixed presentation interval, solutions are presented periodically to users after a fixed number of generations. Conversely, in fitness proportionate presentation interval, solutions are presented after a varying number of iterations have elapsed. During early iterations, the iteration interval is large enabling diversification of the search process. However, during later iterations, the iteration interval is small enabling intensification of the search process. Thus, the influence of the user is more prominent as the search progresses to high performance regions and the fitness of the population increases.

Consequences

When user evaluation is the only means of solution fitness measurement, *rank-based presentation* can be valuable. However, for interactive search where user evaluation is combined with computational fitness calculation, selected individuals (e.g. *banded, cluster representative* or *surrogate presentation*) are recommended. When a large number of generations are needed to explore the search space effectively, *fixed presentation interval* or *fitness proportionate presentation* interval are suitable.

Examples

Rank-based presentation has been applied to large, real-world multiple-project scheduling with complex quality criteria (Shackelford & Corne, 2001). Schedules are presented as Gantt charts or resource profiles, enabling the search to be guided by both the explicit formalized schedule quality criteria and also the (often implicit) human scheduler's non-formalized intuition and experience. Rank-based presentation has also been used in the adaptive surface inspection of rolled steel sheets (Caleb-Solly & Smith, 2007). Eight images are generated from a single image of steel sheet, and an engineer inspects the surfaces side-by-side, scoring them in the range [1,10].

A natural application area of *banded-presentation* is drug discovery. Interestingly, the goal is not to find optimal solutions, or even molecules that could be synthesized. The aim is to explore the drug design space to arrive at fruitful molecule that might be just outside the designers' experience. The intention is to foster search diversity by maintaining a large population, whilst simultaneously allowing interactive fitness on a small number of solutions.

Cluster-representative presentation has been used in numerous engineering applications. Genetic Algorithms have been used interactively to identify high-quality clusters of complex design spaces (Parmee, Cvetkovic, Watson, & Bonham, 2000).

Fitness proportionate presentation is effective in searching for early life-cycle software designs with limited human interaction (Simons & Parmee, 2012). Quantitative measures of the design coupling are combined with qualitative estimates of design elegance. Early in the search, the number of human evaluations is low while the coupling is minimized automatically. Later in the search, the number of human evaluation is increased, as the focus of the design process shifts from automated and quantitative to more manual and qualitative.

COMPONENT-BASED PATTERNS

Solution Repair

Problem

In constrained problems, the search space S of candidate solutions is partitioned into the feasible solutions $S_F \subset \mathrm{S}$ and infeasible solutions $S_I \subset \mathrm{S}$, with $S_F \cup S_I = S$, and $S_F \cap S_I = S$. When the feasible solutions are few and far in between or frequently neighbored by the infeasible ones, designing search operators that produce only them can be challenging and result in operators that are biased in a not necessarily desirable way. It may be more appropriate to allow search operators to produce both feasible and infeasible solutions, and then repair the latter ones. The intent behind solution repair is thus to map the solutions from S_I to S_F so that they can be evaluated. Accordingly, forces acting for this pattern include problems and domains for which (i) it is difficult to design search operators that guarantee producing feasible candidate solutions, and (ii) the sought solutions may be expected to be located in isolated parts of the search space, hard to achieve by visiting feasible solutions only.

Solution

There are two ways in which this design pattern can be structured and combined with other metaheuristic components. A repair operator $r : S \rightarrow S_F$ may be used as a 'wrapper' around a search operator $s : S \rightarrow S$ so that their composition *sr* always produces a feasible solution. Alternatively, the infeasible solutions may be allowed in a working population, in which case r is used only 'lazily', when an infeasible solution needs to be evaluated. Given an evaluation function *f*, evaluation is then redefined as follows:

```
function evaluation(x : S)
begin
  if x ∈ S_F then f(x)
  else f(r(x))
end
```

Consequences

In that latter case, an infeasible solution x receives the evaluation of its feasible 'proxy' $r(x)$, and search may traverse the infeasible regions (provided search operators can work there). This is the arguably most interesting consequence of repair, as it allows the algorithm to find the 'shortcuts' to the parts of S_F that would be hard (or even impossible) to reach via the feasible solutions only. In this sense, *r* embeds S_F in *S*.

On the other hand, introducing solution repair may bias the search process in a hard to predict way (cf. Structural Stigmergy pattern for a similar caveat). When and how often a repair operator is engaged is out of designer's control; therefore, the extent to which a search process is influenced by this feature may vary. Last but not least, when used in the second of the modes mentioned in the above section, an important and often hard to meet requirement is that the evaluation of the repaired solution *(f (r (x)))* should correlate well with the unknown quality of the non-repaired one *(x)*.

Examples

Solution repair is particularly common in continuous optimization, where the allowed ranges for variables are often explicitly given as problem constraints. In the simplest case, repair can be implemented as 'clamping' of values to the admissible domain; see Michalewicz (1995) for a review and more sophisticated methods.

Simplification of programs in genetic programming in order to prevent bloat (excessive growth of program size) can be seen as another form of solution repair (in which case the violated constrain is the program size). In the simplest form, this can boil down to removal of introns, i.e., code fragments that do not contribute to program performance (see, e.g., Haynes, 1996). More sophisticated simplification in genetic programming may lead to overhaul of an entire program. In an extreme case, it may boil down to writing down the behavior of the original program in a different form and re-synthesizing a new program from that description. In Moraglio, Krawiec, and Johnson (2012), the authors rewrite Boolean programs as disjunctive normal forms, and use those forms to rebuild programs from scratch. Further examples and a broader discussion on this design pattern can be found at Krawiec (2014).

Surrogate

Problem

The objective function in an optimization problem may be subject to several forces acting:

- It may be costly to execute, e.g. require a long-running simulation or confrontation with many environments/tests (as may be the case in genetic programming).
- It may be noisy, returning different values for each evaluation of the same solution
- It may fail to effectively guide the search process to high-quality solutions. This is particularly true when the fitness landscape is rugged, which increases the likelihood of search getting trapped in a local optimum, or/and it has plateaus, providing no distinction between solutions. These issues

are illustrated in Figure 3. The objective function (solid blue) has local optima (region a), and a plateau (region b), both of which may hinder the search.

Solution

The solution is to design (or generate) a *surrogate model* for the objective function. An example surrogate function for the example in Figure 3 is shown as a dashed line. It is the responsibility of the metaheuristics practitioner to craft a function which is an effective substitute and is at least one of the following: less expensive to evaluate, less noisy, or a better guide for the search process.

Surrogates may be developed through a trial and error process, or using domain-specific knowledge. A surrogate function maybe used to drive the entire run, or be interleaved with evaluations of the original fitness function. A UML diagram describing a surrogate function and its relation to other aspects of metaheuristic search is shown in Figure 4. Dynamic surrogates are typically models built online during the search using machine learning techniques. Dynamic surrogate functions can be updated or replaced over time, and adaptive components of this form can be seen as an instance of the 'Bridge' (a.k.a. 'Handle/Body') pattern (Gamma et al., 1995). As observed in the section describing structural patterns, a surrogate can also in turn be an ensemble of simpler surrogates.

Consequences

There are a number of consequences for the different purposes of a surrogate function. A less expensive surrogate function can be derived from the objective function in conjunction with a domain expert. The effectiveness of a surrogate is determined in the context of the representation of the problem and the operators (e.g. mutation or crossover). A negative consequence is that designing a surrogate function by hand requires the metaheuristic designer to think carefully about the interplay of the representation, operator and fitness function. Designing well-informed surrogates can therefore be challenging (as illustrated in the final example below).

Figure 3. A raw objective function (blue) and a surrogate (red)

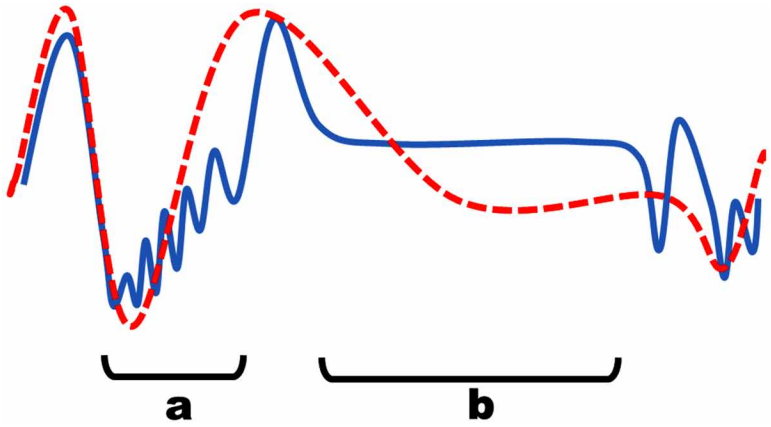
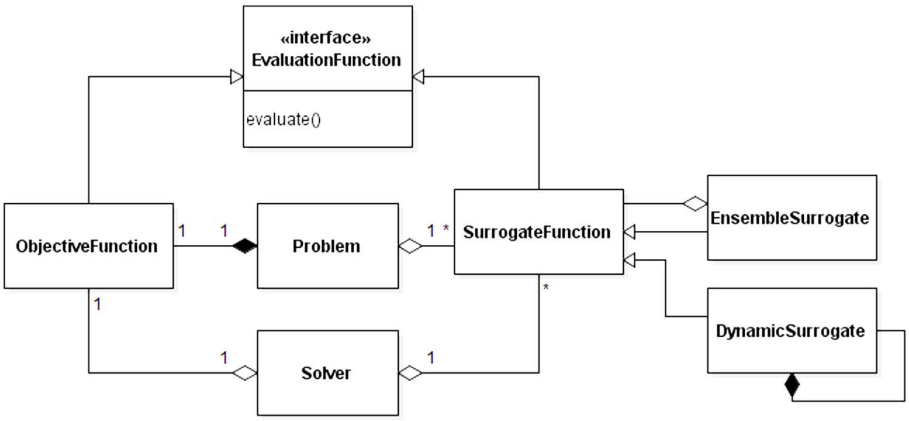


Figure 4. UML diagram of the 'Surrogate' pattern



Examples

Some problem domains may have no explicit objective function (see the 'Subjective Preference' pattern of the Methodological Patterns section), or the objective function is computationally expensive (Jin, Olhofer, & Sendhoff, 2002; Brownlee & Wright, 2015). The bin-packing problem (Burke, Hyde, Kendall, & Woodward, 2007; Poli, Woodward, & Burke, 2007), in which the aim is to pack a set of items into the smallest number of bins possible, is one area where another function can be substituted more effectively for the objective function. The naive objective function is simply the number of bins. However, this is a rather coarse measure, and does not take account of how the items are placed within the bins. Yet, such information may provide a valuable gradient.

A surrogate function can effectively add noise to the objective function making it less likely to stall in a local optima local optima (Holmstrom & Koistinen, 1992). Supplementing an objective function with noise effectively smooths out local optima. However, adding too much noise, or the wrong sort of noise, can be detrimental to the search process (Van Gorp, Schoukens, & Pintelon, 1998).

Other examples of surrogates include objective functions with noise (Bhattacharya 2008), plateaus and multi-modality (Ong, Zhou, & Lim, 2006; Liang, Yao, & Newton, 1999; Liang, Yao, & Newton, 2000; Shakya, McCall, & Brown, 2006), constraints (Jin, Oh, & Jeon, 2010), and deceptive gradient information (Brownlee, Regnier-Coudert, McCall, & Massie, 2010). The reader desiring further illustrative examples is directed to Jin (2005,2011).

Subjective Preference

Problem

In interactive metaheuristic search, a human provides information to manually steer the search trajectory. The human is essentially part of the generate-and-test cycle, and is effectively 'in-the-loop', providing qualitative evaluation to assist in solution selection (Takagi, 2001). Either qualitative human evaluation alone can be used to select solutions, or in conjunction with quantitative objective fitness functions. However, human evaluation is partly subjective, based on preferences which are not readily expressed.

In terms of forces acting, user preferences can range from vague qualitative assessments, to explicit statements of desiderata. To further complicate matters, preferences may also be a conflation of many specific concerns, with many incomparable preferences being evaluated subconsciously. Capturing human preferences is thus a challenging and ill-defined problem. To address this issue, the *intent* of the pattern is to exploit subjective preference information along to two directions: firstly, the type of preference information, and secondly, at what point preference is used in the search process.

Solution

The Subjective Preference design pattern (Aljawawdeh, Simons, & Odeh, 2015) addresses this problem of capturing both explicit and implicit human preferences. Explicit preferences are readily articulated and expressed programmatically (typically in the form of production rules). In contrast, implicit preferences may be a combination of many hidden, subconscious biases and are therefore more difficult to elicit directly. Nevertheless, the user is still able to express a preference between a pair of solutions. The difficulty of articulating a preference for one solution over another resonates with 'quality without of name' advocated by Alexander (1979) whereby 'you know it when you see it'.

Human feedback can be incorporated into a metaheuristic either before search (a priori), during search (interactive), or after search (a posteriori):

- A priori preferences are useful in addressing "multi-subjective" concerns.
- The interactive method provides preference information distinguishing the fitness of solutions being referred to as 'human-in-the-loop' metaheuristics
- A posteriori method is useful in the user selection of solutions from a final pool of solutions.

Consequences

Where user preference information tends to explicit, *a priori* and interactive approaches can be beneficial. On the other hand, where user preference information is implicit, interactive and a posteriori approaches may be more appropriate. It is also possible for user preference information to be exploited at different stages of metaheuristic search. Thus the major force driving significant trade-offs in the subjective preference pattern is to distinguish the nature of user preference information along a continuum from implicit to explicit. The more explicit the preference, the earlier in metaheuristic search the preference can be exploited for effective and efficient search.

Examples

It is interesting that few examples of this pattern being applied to implicit, *a priori* preference are apparent. On reflection, this is perhaps not surprising as implicit preference information is difficult to elicit before metaheuristic search. However, there are many example problem domains where implicit preference information has been used in interactive search, including conceptual engineering design (e.g. Parmee et al., 2000), software design (e.g. Simons & Parmee, 2012), and software refactoring (e.g. Simons, Smith, & White, 2014). Other plausible example problem domains include simulation test functions, groundwater conceptual design, fragrance optimization, aerodynamic airfoil shape design, fuzzy path planning for mobile robotics, and software product lines (e.g. feature maps).

On reflection, this is also perhaps not surprising as implicit preference is well suited to the 'softer', more aesthetic aspects of human evaluation. Examples of implicit preference information being used a posteriori are also available and include problem domains such as automotive assembly line balancing and nurse roster scheduling.

Examples of the application of this pattern with explicit preference information are also available, principally for *a priori* and interactive metaheuristic search. Example application problem domains include multi-mode resource investment project scheduling problems, and fashion design. However, there are also many examples of applying explicit preference information for various numerical optimization problems (e.g. Swiss Federal Institute of Technology, Zurich, 2016). We speculate that in these applications, explicit preference information may act as a search constraint. Further details and references for all examples above are available in Aljawawdeh et al., (2015).

FUTURE DIRECTIONS

The catalogue of design patterns presented in this paper is by no means complete. Presenting even those which are currently apparent to the authors would easily result in a textbook rather than a book chapter. The omitted patterns can be divided into two groups. The first embraces 'sophisticated' patterns that typically build upon the conventional components and intend to improve performance of metaheuristic search: examples include elitism; helper objective; island model; spatial embedding of candidate solutions; archive, as well as domain-specific patterns, like code reuse or evaluation on external examples in genetic programming. The second group includes the elementary 'idiomatic' patterns related to the common components like selection, acceptance, perturbation etc. A pattern-based perspective on these leads to interesting observations which we elaborate on in the remainder of this section.

A Catalogue of Qualitative Components

Despite an increasing diversity of metaheuristic methods (GAs, ACO, PSO, ILS etc.), a number of subordinate mechanisms ('components') are common to almost all. These components (notably selection, perturbation, acceptance, recombination) therefore could be said to correspond to patterns at a low-level 'idiomatic' scale (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996). It is therefore interesting to try and re-state the intuition which motivated these components in a more explicit fashion. One way of doing this is outlined in the following listings, which provide examples of 'qualitative' versions of annealing- and tabu- style acceptance mechanisms, the idea being that the motivation can be informatively described via the 'linguistic hedges' of fuzzy logic (Zadeh, 1996) with rule antecedents and consequents corresponding to 'forces acting' and 'consequences'. An example of how this might be done for popular acceptance criteria is given in the following listing:

```
function annealingAcceptance(timeRemaining, deltaF)
begin
  if crisp(deltaF) IS >= 0
    prAccept IS 1.0
  if timeRemaining IS HIGH and deltaF IS SMALL
    prAccept IS HIGH
```

```
  ...
  if timeRemaining IS LOW and deltaF IS SMALL
    prAccept IS MEDIUM
  ...
  if timeRemaining IS LOW and deltaF IS LARGE
    prAccept IS VERY_LOW
end

function tabuAcceptance(tabuList, incoming)
begin
  for all features f in tabuList
  begin
    if similarity(f, incoming) IS HIGH
        prAccept IS LOW
    ...
    if similarity(f, incoming) IS LOW
        prAccept IS HIGH
    ...
  end
end
```

There is of course extensive previous work using fuzzy logic as an (adaptive) control mechanism for metaheuristics (see e.g. Castillo, & Melin, 2015, and in particular Pelta, Sancho-Royo, & Verdegay, 2003). The interest from a patterns perspective is that of cross-domain knowledge discovery, rather than simply seeking to produce a metaheuristic which excels in a particular problem domain. The idea is to obtain parametric descriptions of effective components using large scale data-mining, described in more detail in the next section.

Pattern Induction

Although there are variations in the nature of the operators applied, the increasing diversity of metaheuristics is still characterized by a relatively small number of framework templates, with the well-known local search and EC templates predominating (e.g. see Bezerra, Lopez-Ibanez, & Stuzle, 2016). It is possible that research has suffered from an historical bias in this respect: for example, it is something of an article of faith that the 'select\recombine\mutate\merge' generational cycle that characterizes EC frameworks is universal.

In contrast, some recent work by Martin and Tauritz (2013) hyper-heuristically evolved an unconstrained operator pipeline (in which individual operators take a population as input and output a modified population). In case studies, it happened that two successive recombinations yielded the best cross-domain solution. It would therefore be of benefit to perform much wider investigations into generic

ways of expressing preconditions for the effectiveness of an operator. One way to achieve this would be to further extend the above approach of characterizing 'forces acting' in parametric terms, using metrics extracted from the solution trajectory. For example, it might be hoped that the applicability of a metaheuristic component can be described via 'universal' metrics of intensification, diversification and randomness (Blum, 2003).

Pattern Languages as Grammars

As described above, the ability to parameterize families of components via a relatively small collection of metrics opens up new possibilities for selecting\generating them. If we additionally wish to generate frameworks for invoking these components, then this is an area where pattern languages have great potential utility. As noted in Krasnogor (2009), a pattern language can be considered to define a grammar.

Some foundational work in this area is based on the ILS template given in the Structural Patterns section, where Marmion et al. (2013) describe an automated approach for the offline hybridization of local search algorithms, manually defining the associated grammar so as to limit the depth of recursive local search instantiations. The production rules employed in this approach are in the general form of grammatical evolution (O'Neill & Ryan, 2003), i.e. unguarded transitions chosen via a random integer index. Assuming the existence of classes corresponding directly to those in the class diagram, one alternative approach is to use a combination of reflection (Lucas, 2004) and Ant Programming (Roux & Fonlupt, 2000) e.g. as implemented in the ContainAnt system (Kocsis and Swan, 2017) for online construction.

Further automation may be facilitated by expressing the production rules of the grammar more strongly in terms of 'forces acting'. By using such structured heuristic preconditions as production rules, this leads to an informed realization of Krasnogor's 'Self-Generating Strategy Pattern' (2009), for example by using any of the qualitative methods previously described in this Section. In addition, the AI and machine learning communities have a long tradition of expressing and inducing rule-based systems, leading to many possible avenues of further research.

CLOSING REMARKS

The authors of this paper find the pattern-based perspective essential for the avoidance of duplication and further progress in the field. As a whole, the commonly adopted conceptual architecture of metaheuristic search components (i.e., cutting across all paradigms, like EC, tabu search, simulated annealing, etc.) is surprisingly limited. Apart from the widespread agreement upon a handful of essential terms (e.g. candidate solution, search operator, objective function, etc.), little has been done to build consistent, collaborative and consecutive layers of abstraction upon them. In other words, our vocabulary to describe anything between those basic components and entire complex algorithms is rather narrow. Metaheuristic Design Patterns are meant to fill that gap. The successful adoption of the pattern-based perspective in other disciplines is a clear indication that this avenue is worth pursuing in the field of larger-scale computing architectures, systems and search.

ACKNOWLEDGMENT

We would like to thank contributors to the Metaheuristic Design Patterns (MetaDeeP) workshops held in 2014 and 2015 at the Genetic and Evolutionary Computing Conference (GECCO). John Woodward was funded in part by EPSRC grant EP/J017515/1 (DAASE). Jerry Swan acknowledges the support of the EU H2020 SAFIRE Factories project (Ref. 723634). Krzysztof Krawiec acknowledges support from grant 2014/15/B/ST6/05205 funded by the National Science Centre, Poland.

REFERENCES

Alexander, C. (1979). *The timeless way of building*. New York, NY: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A pattern language: Towns, buildings, construction*. Oxford University Press.

Aljawawdeh, H. J., Simons, C. L., & Odeh, M. (2015). Metaheuristic design pattern: Preference. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 1257-1260. doi:10.1145/2739482.2768498

Arcuri, A., & Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, 1-10. doi:10.1145/1985793.1985795

Arcuri, A., & Briand, L. (2014). A hitchhikers guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing. *Verification and Reliability*, *24*(3), 219–250. doi:10.1002/stvr.1486

Bezerra, L. C., López-Ibánez, M., & Stützle, T. (2016). Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *20*(3), 403–417. doi:10.1109/TEVC.2015.2474158

Bhattacharya, M. (2008). Reduced computation for evolutionary optimization in noisy environment. *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*, 2117-2122. doi:10.1145/1388969.1389033

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Secauscus, NJ: Springer-Verlag New York Inc.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 268–308. doi:10.1145/937503.937505

Booch, G. (1994). *Object oriented analysis & design with applications* (2nd ed.). Redwood City, CA: Benjamin-Cummings Publishing.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. doi:10.1023/A:1010933404324

Brownlee, A. E., & Wright, J. A. (2015). Constrained, mixed-integer and multi-objective optimisation of building designs by NSGA-II with fitness approximation. *Applied Soft Computing*, *33*, 114–126. doi:10.1016/j.asoc.2015.04.010

Brownlee, A. E. I., Regnier-Coudert, O., McCall, J. A., & Massie, S. (2010). Using a markov network as a surrogate fitness function in a genetic algorithm. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, 4525-4532. doi:10.1109/CEC.2010.5586548

Brownlee, A. E. I., Swan, J., Özcan, E., & Parkes, A. J. (2014). Hyperion2: A Toolkit for {Meta-, Hyper-} Heuristic Research. *Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation*, 1133–1140.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In M. Gendreau, & J. Potvin (Eds.), Handbook of metaheuristics (pp. 449-468). Springer. doi:10.1007/978-1-4419-1665-5_15

Burke, E. K., Hyde, M., Kendall, G., & Woodward, J. R. (2007). The scalability of evolved on line bin packing heuristics. *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, 2530-2537. doi:10.1109/CEC.2007.4424789

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-oriented software architecture; on patterns and pattern languages*. Chichester, UK: John Wiley & Sons.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture: A system of patterns*. New York, NY: John Wiley & Sons.

Caleb-Solly, P., & Smith, J. E. (2007). Adaptive surface inspection via interactive evolution. *Image and Vision Computing*, *25*(7), 1058–1072. doi:10.1016/j.imavis.2006.04.023

Castillo, O., & Melin, P. (2015). Fuzzy logic augmentation of nature-inspired optimization metaheuristics. Berlin, Germany: Springer.

Cowling, P., Kendall, G., & Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT '00)*, 176-190.

Dorigo, M., & Stutzle, T. (2004). *Ant colony optimization*. Cambridge, MA: Bradford Books, MIT Press.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, *12*(2), 213–253. doi:10.1145/356810.356816

Flach, P. (2012). *Machine learning: The art and science of algorithms that make sense of data*. New York: Cambridge University Press. doi:10.1017/CBO9780511973000

Fogel, D. B. (1998). *Evolutionary computation: The fossil record*. Wiley-IEEE Press. doi:10.1109/9780470544600

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Boston, MA: Addison-Wesley Professional.

Frutos, M., Tohmé, F., & Miguel, F. (2016). A Genetic Algorithm's Approach to the Optimization of Capacitated Vehicle Routing Problems. In Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics. IGI Global. doi:10.4018/978-1-4666-9644-0.ch008

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reuseable object-oriented software*. Addison-Wesley.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58. doi:10.1162/neco.1992.4.1.1

Giraud-Carrier, C., & Provost, F. (2005). Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. *Proceedings of the ICML-2005 Workshop on Meta-Learning*, 12-19.

Glover, F. (1989). Tabu search-part I. *ORSA Journal on Computing, 1*(3), 190-206.

Gordini, N. (2014). Genetic algorithms for small enterprises default prediction: Empirical evidence from Italy. In *Handbook of Research on Novel Soft Computing Intelligent Algorithms* (pp. 258–293). IGI Global. doi:10.4018/978-1-4666-4450-2.ch009

Goudos, S. K., Siakavara, K., & Sahalos, J. N. (2016). Application of Artificial Bee Colony Algorithms to Antenna Design Problems for RFID Applications. In Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics. IGI Global. doi:10.4018/978-1-4666-9644-0.ch009

Graham, K., Swan, J., & Martin, S. (2015). The blackboard pattern for metaheuristics. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 1265-1267. doi:10.1145/2739482.2768500

Hathaway, W. M. (2013). *Automated hypothesis testing* (US Patent 8,370,107 ed.). Google Patents. Retrieved 24 January 2017 from http://www.google.com/patents/US8370107

Haynes, T. (1996). Duplication of coding segments in genetic programming. *Proceedings of the 13th National Conference on Artificial Intelligence*, 344-349.

Hofstadter, D. R., & Mitchell, M. (1995). *The copycat project: A model of mental fluidity and analogy-making. In Fluid concepts and creative analogies* (pp. 205–267). New York, NY: Basic Books Inc.

Holmstrom, L., & Koistinen, P. (1992). Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, *3*(1), 24–38. doi:10.1109/72.105415 PMID:18276403

Hooker, J. N. (1994). Needed: An empirical science of algorithms. *Operations Research*, *42*(2), 201–212. doi:10.1287/opre.42.2.201

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, *9*(1), 3–12. doi:10.1007/s00500-003-0328-5

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, *1*(2), 61–70. doi:10.1016/j.swevo.2011.05.001

Jin, Y., Oh, S., & Jeon, M. (2010). Incremental approximation of nonlinear constraint functions for evolutionary constrained optimization. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, 2966-2973. doi:10.1109/CEC.2010.5586355

Jin, Y., Olhofer, M., & Sendhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, *6*(5), 481–494. doi:10.1109/TEVC.2002.800884

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. *6th International Conference on Genetic Algorithms*, 184-192.

Joslin, D. E., & Clements, D. P. (1999). Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, *10*, 353–373.

Kittler, J., Hatef, M., Duin, R. P., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(3), 226–239. doi:10.1109/34.667881

Kocsis, Z. A., & Swan, J. "Dependency Injection for Programming by Optimization". In: CoRR abs/1707.04016 (2017). url: http://arxiv.org/abs/1707.04016

Kovitz, B., & Swan, J. (2014). Structural stigmergy: A speculative pattern language for metaheuristics. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1407-1410. doi:10.1145/2598394.2609845

Kovitz, B., & Swan, J. (2014). Tagging in metaheuristics. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1411-1414.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT press.

Krasnogor, N. (2009). *Memetic algorithms. Handbook of natural computation* (N. Computation, Ed.). Berlin: Springer.

Krawiec, K. (2014). Metaheuristic design pattern: Candidate solution repair. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1415-1418.

Liang, K., Yao, X., & Newton, C. (1999). Combining landscape approximation and local search in global optimization. *Proceedings of the IEEE World Conference on Computational Intelligence (WCCI '99)*, *2*, 1514-1520.

Liang, K., Yao, X., & Newton, C. (2000). Evolutionary search of approximated n-dimensional landscapes. *International Journal of Knowledge Based Intelligent Engineering Systems*, *4*(3), 172–183.

Lim, D., Jin, Y., Ong, Y., & Sendhoff, B. (2010). Generalizing surrogate-assisted evolutionary computation. *IEEE Transactions on Evolutionary Computation*, *14*(3), 329–355. doi:10.1109/TEVC.2009.2027359

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). *The iRace package: Iterated racing for automatic algorithm configuration* (Technical Report No. TR/IRIDIA/2011-004). Brussels, Belgium: IRIDIA, Universite Libre de Bruxelles.

López-Ibáñez, M., Mascia, F., Marmion, M., & Stützle, T. (2014). A template for designing single-solution hybrid metaheuristics. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*, 1423-1426.

Lucas, S. M. (2004). Exploiting reflection in object oriented genetic programming. In *Lecture Notes in Computer Science: Vol. 3003. Proceedings of the 7th European Conference on Genetic Programming* (pp. 369-378). Berlin: Springer. doi:10.1007/978-3-540-24650-3_35

Majumder, A., & Majumder, A. (2015). Application of standard deviation method integrated PSO approach in optimization of manufacturing process parameters. In *Handbook of Research on Artificial Intelligence Techniques and Algorithms* (pp. 536–563). IGI Global. doi:10.4018/978-1-4666-7258-1.ch017

Marmion, M., Mascia, F., López-Ibánez, M., & Stützle, T. (2013). Automatic design of hybrid stochastic local search algorithms. In *Lecture Notes in Computer Science: Vol. 7919. International Workshop on Hybrid Metaheuristics* (pp. 144-158). Berlin: Springer. doi:10.1007/978-3-642-38516-2_12

Martin, M. A., & Tauritz, D. R. (2013). Evolving black-box search algorithms employing genetic programming. *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, 1497-1504. doi:10.1145/2464576.2482728

Martin, S., Ouelhadj, D., Smet, P., Berghe, G. V., & Özcan, E. (2013). Cooperative search for fair nurse rosters. *Expert Systems with Applications*, *40*(16), 6674–6683. doi:10.1016/j.eswa.2013.06.019

Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation methods. *Evolutionary Programming*, *4*, 135–155.

Milano, M., & Roli, A. (2004). MAGMA: A multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics*, *34*(2), 925–941. doi:10.1109/TSMCB.2003.818432 PMID:15376840

Mitchell, T. M. (1997). *Machine learning* (1st ed.). New York, NY: McGraw-Hill, Inc.

Moraglio, A., Krawiec, K., & Johnson, C. G. (2012). Geometric semantic genetic programming. In *Lecture Notes in Computer Science: Vol. 7491. International Conference on Parallel Problem Solving from Nature* (pp. 21-31). Berlin: Springer. doi:10.1007/978-3-642-32937-1_3

Neumann, G., Harman, M., & Poulding, S. (2015). Transformed vargha-delaney effect size. In *Lecture Notes in Computer Science: Vol. 9275. International Symposium on Search Based Software Engineering* (pp. 318-324). Berlin: Springer. doi:10.1007/978-3-319-22183-0_29

Neumann, G., Swan, J., Harman, M., & Clark, J. A. (2014). The executable experimental template pattern for the systematic comparison of metaheuristics. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1427-1430. doi:10.1145/2598394.2609850

Nguyen, T. T., & Vo, D. N. (2016). Cuckoo Search Algorithm for Hydrothermal Scheduling Problem. In Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics. IGI Global. doi:10.4018/978-1-4666-9644-0.ch014

O'Neil, M., & Ryan, C. (2003). *Grammatical evolution: Evolutionary automatic programming in an arbitrary language*. Norwell, MA: Kluwer Academic Publishers. doi:10.1007/978-1-4615-0447-4

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., . . . Petrovic, S. (2012). Hyflex: A benchmark framework for cross-domain heuristic search. In *Lecture Notes in Computer Science: Vol. 7245. European Conference on Evolutionary Computation in Combinatorial Optimization* (pp. 136-147). Berlin: Springer. doi:10.1007/978-3-642-29124-1_12

Ong, P., & Kohshelan, S. (2016). Performances of Adaptive Cuckoo Search Algorithm in Engineering Optimization. In Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics. IGI Global. doi:10.4018/978-1-4666-9644-0.ch026

Ong, Y., Zhou, Z., & Lim, D. (2006). Curse and blessing of uncertainty in evolutionary algorithm using approximation. *Proceedings of the IEEE International Conference on Evolutionary Computation (CEC '06)*, 2928-2935.

Ouelhadj, D., & Petrovic, S. (2010). A cooperative hyper-heuristic search framework. *Journal of Heuristics*, *16*(6), 835–857. doi:10.1007/s10732-009-9122-6

Parmee, I. C., Cvetković, D., Watson, A. H., & Bonham, C. R. (2000). Multiobjective satisfaction within an interactive evolutionary design environment. *Evolutionary Computation*, *8*(2), 197–222. doi:10.1162/106365600568176 PMID:10843521

Pelta, D. A., Sancho-Royo, A., & Verdegay, J. L. (2003). On the design of metaheuristic algorithms using fuzzy rules. *Proceedings of the 3rd Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)*, 474-479.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). *A field guide to genetic programming*. Lulu Enterprises.

Poli, R., Woodward, J., & Burke, E. K. (2007). A histogram-matching approach to the evolution of bin-packing strategies. *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, 3500-3507. doi:10.1109/CEC.2007.4424926

Polprasert, J., Ongsakul, W., & Dieu, V. N. (2016). Improved Pseudo-Gradient Search Particle Swarm Optimization for Optimal Power Flow Problem. In *Sustaining Power Resources through Energy Optimization and Engineering* (pp. 177–207). IGI Global. doi:10.4018/978-1-4666-9755-3.ch008

Popa, R. (2014). Melanocytic Lesions Screening through Particle Swarm Optimization. In Handbook of Research on Novel Soft Computing Intelligent Algorithms: Theory and Practical Applications. IGI Global. doi:10.4018/978-1-4666-4450-2.ch012

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118. doi:10.1016/S0065-2458(08)60520-3

Roux, O., & Fonlupt, C. (2000). Ant programming: Or how to use ants for automatic programming. *Proceedings of ANTS 2000 from Ant Colonies to Artificial Ants: 2nd International Workshop on Ant Algorithms*, 121-129.

Shackelford, M., & Corne, D. (2001). Collaborative evolutionary multi-project resource scheduling. *Proceedings of the 2001 Congress on Evolutionary Computation*, *2*, 1131-1138. doi:10.1109/CEC.2001.934318

Shackelford, M., & Simons, C. L. (2014). Metaheuristic design pattern: Interactive solution presentation. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1431-1434.

Shakya, S. K., McCall, J. A., & Brown, D. F. (2006). Solving the ising spin glass problem using a bivariate EDA based on markov random fields. *Proceedings of the IEEE International Conference on Evolutionary Computation (WCCI '06)*, 908-915. doi:10.1109/CEC.2006.1688408

Simons, C. L., & Parmee, I. C. (2012). Elegant object-oriented software design via interactive, evolutionary computation. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *42*(6), 1797–1805. doi:10.1109/TSMCC.2012.2225103

Simons, C. L., Smith, J., & White, P. (2014). Interactive ant colony optimization (iACO) for early lifecycle software design. *Swarm Intelligence*, *8*(2), 139–157. doi:10.1007/s11721-014-0094-2

Singh, B. K. (2017). Evaluation of Genetic Algorithm as Learning System in Rigid Space Interpretation. In *Nature-Inspired Computing: Concepts, Methodologies, Tools, and Applications* (pp. 1184–1228). IGI Global. doi:10.4018/978-1-5225-0788-8.ch045

Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, *22*(1), 3–18. doi:10.1111/itor.12001

Swan, J., & Burles, N. (2015). Templar – a framework for template-method hyper-heuristics. *Proceedings of the 18th European Conference on Genetic Programming*, 205-216.

Swan, J., De Causmaecker, P., Martin, S., & Özcan, E. (2016). A re-characterization of hyper-heuristics. In F. Yalaoui & A. E.-G. Talbi (Eds.), *Recent Developments of Metaheuristics*. Springer.

Swan, J., Harman, M., Ochoa, G., & Burke, E. (2012). Generic software subgraph isomorphism. *Proceedings of the 4th Symposium on Search Based-Software Engineering*, 53. Retrieved from http://selab.fbk.eu/ssbse2012/documents/SSBSE_2012_Fast_Abstracts.pdf#page=53

Swan, J., Kocsis, Z. A., & Lisitsa, A. (2014). The 'representative' metaheuristic design pattern. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1435-1436.

Swan, J., Özcan, E., & Kendall, G. Hyperion - A Recursive Hyper-Heuristic Framework. In C. Coello (Ed.), *Learning and Intelligent Optimization* (Vol. 6683, pp. 616–630). doi:10.1007/978-3-642-25566-3_48

Swan, J., Woodward, J., Özcan, E., Kendall, G., & Burke, E. (2014). Searching the hyper-heuristic design space. *Cognitive Computation*, *6*(1), 66–73. doi:10.1007/s12559-013-9201-8

Swiss Federal Institute of Technology Zurich. (2016). *Density and approximations of μ-distribution for different test problems.* Retrieved 24 January 2017 from http://people.ee.ethz.ch/~sop/download/supplementary/testproblems/

Takagi, H. (2001). Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, *89*(9), 1275–1296. doi:10.1109/5.949485

Tibermacine, C., Sadou, S., That, M. T. T., & Dony, C. (2016). Software architecture constraint reuse-by-composition. *Future Generation Computer Systems*, *61*, 37–53. doi:10.1016/j.future.2016.02.006

Trejos, J., Villalobos-Arias, M. A., & Espinoza, J. L. (2016). Variable Selection in Multiple Linear Regression Using a Genetic Algorithm. In Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics. IGI Global. doi:10.4018/978-1-4666-9644-0.ch005

Vaessens, R. J. M., Aarts, E. H. L., & Lenstra, J. K. (1998). A local search template. *Computers & Operations Research*, *25*(11), 969–979. doi:10.1016/S0305-0548(97)00093-2

Van Gorp, J., Schoukens, J., & Pintelon, R. (1998). Adding input noise to increase the generalization of neural networks is a bad idea. *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, 127-132.

Weise, T., Chiong, R., Lassig, J., Tang, K., Tsutsui, S., Chen, W., & Yao, X. et al. (2014). Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. *IEEE Computational Intelligence Magazine*, *9*(3), 40–52. doi:10.1109/MCI.2014.2326101

Woodward, J., Swan, J., & Martin, S. (2014). The composite design pattern in metaheuristics. *Proceedings of the Companion Publication of the Conference on Genetic and Evolutionary Computation (GECCO '14)*, 1439-1444.

Woodward, J. R., & Bai, R. (2009). Canonical representation genetic programming. *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, 585-592. doi:10.1145/1543834.1543914

Woodward, J. R., & Bai, R. (2009). Why evolution is not a good paradigm for program induction: A critique of genetic programming. *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, 593-600. doi:10.1145/1543834.1543915

Woodward, J. R., & Swan, J. (2011). Automatically designing selection heuristics. *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '11)*, 583-590. doi:10.1145/2001858.2002052

Woodward, J. R., & Swan, J. (2012). The automatic generation of mutation operators for genetic algorithms. *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*, 67-74. doi:10.1145/2330784.2330796

Yuce, B., & Mastrocinque, E. (2016). Supply Chain Network Design Using an Enhanced Hybrid Swarm-Based Optimization Algorithm. In *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics* (pp. 95–112). IGI Global. doi:10.4018/978-1-4666-9644-0.ch003

Yuce, B., Mastrocinque, E., Packianather, M. S., Lambiase, A., & Pham, D. T. (2015). The Bees Algorithm and its applications. In *Handbook of Research on Artificial Intelligence Techniques and Algorithms* (pp. 122–151). IGI Global. doi:10.4018/978-1-4666-7258-1.ch004

Zadeh, L. A. (1996). Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, *4*(2), 103–111. doi:10.1109/91.493904

KEY TERMS AND DEFINITIONS

Architecture: A software architecture describes the high-level constructs of a software system, the relationships between them, and the disciplined process of creating such structures. Such structures enable and promote understanding of the software system. The notion of software architecture is a metaphor, inspired by the use of architecture in the built environment.

Design: A cognitively-demanding process used in many fields for the planning or construction of an object, artifact or product in response to a practical need or requirement. Software design considers the functional, economic, aesthetic and performance qualities of the software-to-be, and can incorporate much repeated iteration while evaluating design solution candidates.

Metaheuristic: A higher-level technique exploiting heuristics to arrive at a sufficiently satisfactory solution to an optimization problem, especially with limited information or finite computational capacity. In contrast to iterative methods such as exhaustive search, metaheuristics do not guarantee that a globally optimum solution can be arrived at for a given class of problems.

Optimization: The selection of the best element(s) (regarding some criterion(a)) from a set of available possibilities. In the computer science 'black box' model of computing, optimization refers to maximizing or minimizing input values for a given black box model of the world to achieve the best output values. More generally, optimization involves finding best available values to some objective function for a defined problem domain.

Pattern: According to Gamma, Helm, Johnson, and Vlissides (1995) software design patterns are, *"... descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context"*, and comprise four essential elements: a pattern name, a design problem, a design solution and the consequences, i.e. the results and tradeoffs of applying the pattern.

Standardization: A process of developing and implementing technical standards based on consensus among various stakeholders in the field. Standardization can greatly assist with compatibility and interoperability of otherwise disparate software components, where consistent solutions enable mutual gains for all stakeholders.

Unification: The act of bringing together disparate and heterogeneous software components into a single coherent approach, for example, by using software design patterns.