Two-B or not Two-B? Design Patterns for Hybrid Metaheuristics

Alina Patelli, Nelly Bencomo, Anikó Ekárt, Harry Goldingay and Peter R. Lewis
Aston Lab for Intelligent Collectives Engineering (ALICE)
Aston University, Birmingham, United Kingdom
{a.patelli2,n.bencomo,a.ekart,goldinhj,p.lewis}@aston.ac.uk

ABSTRACT

Real world search problems, characterised by nonlinearity, noise and multidimensionality, are often best solved by hybrid algorithms. Techniques embodying different necessary features are triggered at specific iterations, in response to the current state of the problem space. In the existing literature, this alternation is managed either statically (through pre-programmed policies) or dynamically, at the cost of high coupling with algorithm inner representation. We extract two design patterns for hybrid metaheuristic search algorithms, the *All-Seeing Eye* and the *Commentator* patterns, which we argue should be replaced by the more flexible and loosely coupled *Simple Black Box* (Two-B) and *Utility-based Black Box* (Three-B) patterns that we propose here. We recommend the Two-B pattern for purely fitness based hybridisations and the Three-B pattern for more generic search quality evaluation based hybridisations.

1. INTRODUCTION

Metaheuristics have different traits, for example Simulated Annealing promotes *diversity* by tolerating certain poor quality candidates, Genetic Programming *exploits* flexible representations, evolutionary strategies are *self-adaptive*. Regardless of their nature – stochastic or deterministic, population based or single solution, global or local – no search algorithm is perfect. The No Free Lunch theorem states that, for any non-revisiting search algorithm, "elevated performance over one class of problems is offset by performance over another class" [13]. As a result of this, search algorithms are designed to solve particular classes of problems well rather than to maximise applicability. Realistic problems, however, do not typically sit neatly into the class of problems solved well by any single search algorithm. As an example, gradient-based algorithms typically exploit a local optimum better than evolutionary approaches, but are worse at locating promising optima in a multimodal landscape. However, solving a multimodal problem requires both the location and the exploitation of optima. When attempting to design an algorithm which harnesses the opposing strengths of different search procedures, is useful to consider hybrid approaches.

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain
Copyright ©2015 ACM 978-1-4503-3488-4/15/07 ...$15.00
DOI http://dx.doi.org/10.1145/2739482.2768501.

Hybrid algorithms combine two or more search techniques and switch between them during the search process. For example, memetic algorithms employ stochastic metaheuristic global search procedures in tandem with local search methods. In this way, the advantages of the component algorithms are combined in a unified, potentially more powerful, platform. Multidimensional problems with nonlinear search spaces and/or multiple local optima stand to benefit the most from the application of hybrid metaheuristics. In such domains, the search is likely to converge prematurely (in a local optimum, due to loss of candidate solution diversity) or not at all (improperly configured search operators). To again use the memetic algorithm example, here the global, population-based search is designed to address the issue of premature convergence, while the local search procedure is specifically aimed at finding the local optimum given a predetermined vicinity.

It is desirable that the logic used to switch between component algorithms, namely determining when it is most beneficial to apply one technique or another, is enacted automatically. Ideally, this decision should be made according to the particulars of the problem (local optima, nonlinearity, etc.) as well as the internal state of the algorithm (time spent searching, population diversity, quality of individuals in the current population, etc.). Several valuable attempts at configuring hybrid algorithms are documented in the literature, yet their common traits are yet to be distilled in a design pattern.

To address this gap, we firstly extract and critique two design patterns from a range of state-of-the-art hybrid metaheuristics. We identify these as antipatterns, due to the tight coupling between components. We then propose two new design patterns to address the design flaws in the extracted patterns, both based around a black-box approach to the management of component algorithms.

The two design patterns we propose capture different ways state-of-the-art hybrid metaheuristics switch between several search techniques during the search process. The first, the *Simple Black Box* (Two-B) pattern simplifies the design, when the switching logic requires only knowledge of fitness values associated with candidate solutions during the search. The second extends this by allowing switching decisions to be made based on additional information about the state of the search (such as population statistics in an EA). This is achieved in a loosely coupled manner, through the proposed use of a generic search quality interface, based around the idea of the current, possibly multi-attribute, utility of the search. We therefore call this pattern the *Utility-based Black Box* (Three-B) pattern.

The rest of this paper is structured as follows. In Section 2 we briefly introduce example hybrid metaheuristics from the literature, which serve to illustrate the patterns subsequently described. In Section 3 we describe the extracted antipatterns, highlighting why current practice in the implementation of hybrid metaheuristics typically embodies poor design. Subsequently, in Section 4 we present our two new patterns which avoid the issues in the extracted antipatterns, and can be used depending on the type of run-time information required by the algorithm switching logic. Finally, in Section 5 we conclude the paper, with recommendations for the effective use of the two proposed patterns, and highlight required areas for future work in order to aid their realisation.

2. EXEMPLARS

To better illustrate the structure and behaviour of all four design patterns, we will use several running examples, briefly described here.

Exemplar 1: Memetic GP. Our first exemplar is a memetic algorithm that combines Genetic Programming (GP) and Orthogonally Least Square (OLS) techniques [5]. The former is employed to evolve the regressor-based structure of nonlinear models, whereas the latter is tasked with computing the optimum model parameters. The algorithm decision maker (swap) policy is fixed, as both techniques are applied at each generation, only to different parts of the model.

Exemplar 2: Memetic MOEA. Our second exemplar is an MOEA spliced with a deterministic local search technique [11] that promotes the survival of solutions situated on the knees (regions of high importance for practitioners) of the Pareto front. The local search technique (deployed at every generation) uses a metric consisting in the weighted sum of the objective values, where the weights are configured in relation to a user-prescribed parameter.

Exemplar 3: GP with Simplification. Our third exemplar applies algebraic expression simplification and genetic programming to symbolic regression [3]. Genetic programming is known to inherently lead to code growth, if no specific measures are taken to prevent this. In the domain of symbolic regression, long expressions can be simplified without affecting their quality. However, applying simplification to evolved expressions too frequently leads to loss in solution quality. Therefore a suitable frequency of applying simplification is empirically determined.

Exemplar 4: GP with Fitness Sharing. Our fourth exemplar proposes the maintenance of diversity in genetic programming populations by adaptively adjusting niche sizes for fitness sharing depending on the current state of a population [4].

Exemplar 5: ARPSO. Our fifth exemplar is a variant of Particle Swarm Optimization (PSO), Attractive and Repulsive PSO (ARPSO) [12], which consists of two phases: an attractive one in which the objective is to converge on a high quality solution and a repulsive one in which the objective is to increase the swarm's capacity to find new solutions. Both of these phases are variants of PSO and the decision to switch between them is based on the diversity of particles.

Exemplar 6: CPSO. Our sixth exemplar is a similar two-phase PSO hybrid variant of PSO, Chaotic PSO (CPSO) [8]. In CPSO, PSO is used to explore the search space and, after some user-defined criterion is met, a Chaotic Local Search (CLS) operator is used to further optimise the best known solution. In canonical CPSO, the decision to switch between PSO and CLS is based on time (number of iterations) and the cost of generated solutions.

Exemplar 7: Triggered Hypermutation. Our seventh exemplar is the use of hypermutation [9] in an EA, which is typically employed in dynamic optimisation problems [6] to ensure exploration in the face of change. Hypermutations are uncharacteristically large mutation operations, typically carried out infrequently, or for a short period of time, while during normal operation smaller, more incremental mutations are made. Hypermutations are typically triggered, for example by the detection of a change in the environment, but hypermutations generated randomly, without the use of any feedback from the search have also been shown to be useful [7]. The classic change detection trigger, which an adaptive algorithm uses to determine when to switch from a standard mutation to a hypermutation in a generational GA, is a detected drop in the running average fitness of the best solution in each generation [2].

Exemplar 8: HyFlex. Our eighth exemplar is a framework created with the goal of enabling practitioners to develop and test combinatorial optimisation algorithms [10]. The environment provides a common software interface and a collection of pre-defined, problem specific component heuristics which may be dynamically plugged into the hyper-heuristic under construction. The designer is freed to concentrate on the high-level aspects of their solution, since the problem-specific details are abstracted away.

3. EXTRACTED ANTIPATTERNS

3.1 The All-Seeing Eye Antipattern

3.1.1 Structure

The first extracted antipattern (which we call the *All-Seeing Eye*) is illustrated in Fig. 1. The **Manager** implements a custom algorithm (designed by the domain expert) that selects between N available search methods, represented by the **Searcher** $\boldsymbol{i}$, $i = 1..N$ components in the diagram. Examples of Searchers are: evolutionary techniques (e.g. genetic algorithms/programming), deterministic procedures (e.g. gradient-based, orthogonal regression, weighted sum minimisation), probabilistic methods (simulated annealing), non-evolutionary, population based algorithms (particle swarm/ ant colony optimisation).

The Manager communicates with the concrete Searchers through a consistent **Searcher** interface, allowing it to request the generation of new candidate solutions and to pass them between Searchers for further optimisation.

Each concrete Searcher i publishes a representation of its inner state (**Model** $\boldsymbol{i}$, $i = 1..N$ in the diagram) for the Manager's benefit. A Model will always contain the candidate solutions currently being acted upon by the Searcher (e.g. the population in a population-based searcher or a single solution for a gradient-based technique), but may also store some Searcher-specific state (e.g. current neighbourhood structure and particle history in PSO).

The Manager feeds model information into its preprogrammed decision algorithm and selects the most appropri-

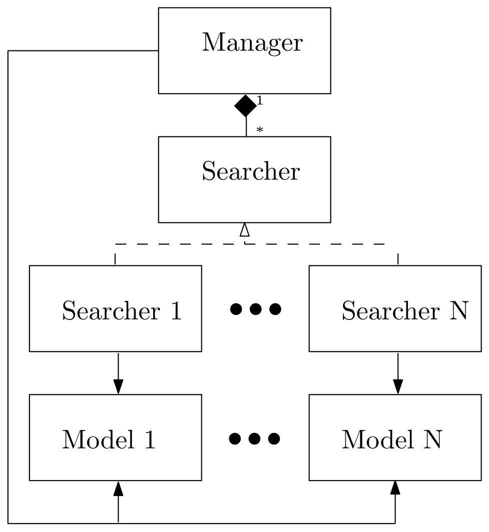

Figure 1: The *All-Seeing Eye* antipattern

ate searcher to apply at a given moment. The decision is communicated through the **Searcher** interface. The flow of information in the *All-Seeing Eye* can be illustrated through the following examples.

3.1.2 *Examples*

In **GP with Simplification**, the Searchers are standard genetic programming and arithmetic simplification, applied as a mutation operator, with some frequency pre-determined by the manager. The goal of arithmetic simplification is to transform a candidate solution to an equivalent, but more compact solution. The Manager accesses the model directly to select individuals for simplification.

In **ARPSO**, both Searchers are PSO variants and share the same Model, which the Manager accesses directly to measure swarm diversity. In **CPSO**, one Searcher (PSO) maintains a population of solutions, while the other (CLS) maintains a single solution. The Manager accesses the Models directly in order to transfer solutions between them.

3.1.3 *Consequences*

Analysing the structure of the *All-Seeing Eye*, we can see that each Searcher is coupled to its Model and that the Manager is coupled to every Model. As a result of this, we cannot replace one Searcher with another in a hybrid algorithm unless their models are identical. The negative impact on this can be seen in **ARPSO** in which the PSO-based repulsive Searcher is ineffective in discovering high quality solutions. Using another Searcher type during the repulsive phase would seem the obvious way to address this issue but, due to the coupling of the Manager and the Searchers, this would require the Manager to be modified as well. Riget and Vesterstrøm's suggestion is to neglect to evaluate the objective function until the attractive Searcher is restarted; however, if there were a pattern to combine and manage searchers with different Models, it would be simple to use a more appropriate Searcher in the repulsive phase.

In **GP with simplification**, without the Manager intervening, the application of arithmetic equivalences could lead to loops or to expressions which are not more compact than the originals.

3.2 The Commentator Antipattern

3.2.1 *Structure*

In some hybrid algorithms, the Manager does not have direct access to the Models but, instead, can request measurements of Model state through an Evaluator. This is shown in Fig. 2 which shows a single branch of the *All-Seeing Eye* diagram, modified to include an **Evaluator** $\boldsymbol{i}$, $i = 1..N$ component. Evaluators have a dual role: when the Manager requests a measurement of some Model property (e.g. the inter-generational fitness improvement rate of a population-based model), the Evaluator must be able to make and return this measurement. They also hide the details of the Model from the Manager, decreasing coupling. As the Manager no longer needs to know how to measure properties of the Model, its sole responsibility becomes Searcher selection, which it bases on information provided by the Evaluators.

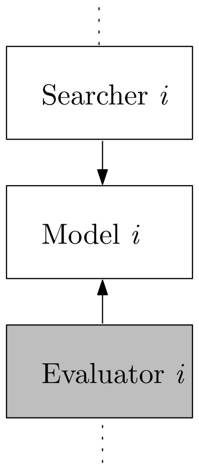

Figure 2: The *Commentator* antipattern extends the *All-Seeing Eye* with an Evaluator

3.2.2 *Examples*

Memetic GP and **Memetic MOEA** implement the *Commentator*, by deploying two search techniques in alternation. In **Memetic GP**, Searcher 1 is a GP procedure and Searcher 2 is OLS, whereas in **Memetic MOEA**, the two searchers are a GA and a weighted sum minimisation technique, respectively. In both cases, the criteria employed by the state evaluator originates from a high level policy authored by the domain expert.

More specifically, in **Memetic GP**, the fact that regressor structure and regressor parameters are evolved separately is part of the domain knowledge. Thus, at every generation, the evaluator will firstly assign the maximum score to the GP searcher and secondly to the OLS one. This way, the manager selects GP to evolve regressor structure and OLS to compute the optimum parameters, in that order, at every generation. Similarly, in **Memetic MOEA**, the criteria the evaluator block implements is also expert knowledge, namely the fact that solutions on Pareto knees are more valuable to practitioners. The GA and weighted sum minimisation searchers are applied in the same order, at every generation.

Note that the dedicated Evaluator block scores the associated searcher: Evaluator 1 assesses the GP Searcher in **Memetic GP** and the MOEA Searcher in **Memetic MOEA**, whereas Evaluator 2 measures the quality of OLS in **Memetic GP** and the weighted sum minimisation procedure in **Memetic MOEA**. The Manager only runs the static switch logic between searchers, based on the scores produced by the Evaluators. As this approach is somewhat inflexible, these metaheuristics would benefit from applying the improved design patterns described in Section 4.

GP with Fitness Sharing also implements the *Commentator*, by varying the niche size parameter for fitness sharing in genetic programming, depending on the population's diversity status. Essentially the same technique, i.e. genetic programming, is applied, but with different niche sizes. The Manager's decision is on how to vary the niche size in response to the diversity evolution. The Evaluator is responsible for determining the diversity status based on structural distances between genetic programming trees.

3.2.3 Consequences

Adding an Evaluator component decreases direct coupling by hiding Model details from the Manager. However, Evaluators represent one extra layer of abstraction since the criteria they employ need to be explicitly specified by domain experts. Thus, there is no real reduction in complexity relative to The *All-Seeing Eye*, merely a shift of complexity from the level of the Manager to that of the Evaluator. In fact, though indirectly, the Manager is still highly coupled to the Model with respect to two pieces of information: the type of Model measurement it is possible to take and how to compare measurements taken from different Models. As a result, each Manger-Model pair requires its own Evaluator, just as in the *All-Seeing Eye*, each Manger-Model pair required some custom logic in the Manager.

We call this antipattern the *Commentator*, since the Evaluator provides a commentary on relevant aspects of the state of the Model to the Manager. However, the Manager is still required to know how to interpret the information provided by the Evaluator.

4. PROPOSED DESIGN PATTERNS

In the previous Section, we presented and analysed two design patterns extracted from common implementations of hybrid metaheuristics. However, in doing so we highlighted issues relating to both coupling and cohesion of the designs. In this section, we propose two alternate design patterns which alleviate these issues. The first, the *Simple Black Box Pattern* (Two-B), simplifies the design substantially, though assumes that the switching mechanism makes decisions based *only* on fitness information provided by the searchers over time, and does not have access to any internal workings of the searcher (e.g. population statistics in a population-based algorithm). If such additional information is needed in order to inform switching decisions, then our second pattern, the *Utility-Based Black Box Pattern* (Three-B) can be used. This pattern enables internal model information to be considered in a loosely coupled way, by associating individual searchers' models with an evaluator specific to that model, responsible for providing model quality assessments to the manager, via a generic utility-based interface. This way, even though evaluators are each associated with a given searcher, they are also aligned (by implementing the same interface), thus allowing consistent and comparable assessment of different models.

4.1 The Simple Black Box (Two-B) Pattern

4.1.1 Intent

The main goal of this pattern is to enable the design of hybrid algorithms in which the Manager and Searcher components are reusable in other contexts by ensuring that the Manager does not need to know about the inner details of the Searcher.

4.1.2 Forces acting

Heterogeneous Searchers - A hybrid algorithm may consist of Searchers of arbitrary, potentially heterogeneous, types. As a heterogeneous collection of Searchers will have heterogeneous Models, a Manager cannot make the same types of detailed measurements of each Model. Allowing the Manager to make different types of measurements of each Model creates coupling and prevents reuse (as in the *All-Seeing Eye* and the *Commentator*).

4.1.3 Structure

The simplest way to reduce the coupling exhibited by the *All-Seeing Eye* and the *Commentator* and to allow component reuse is to remove the Manager's ability to take measurements of the Model (either directly or indirectly). We suggest the pattern in Fig. 3 in which the Model components have been encapsulated inside the searchers and have become transparent to the Manager. Thus, the inner state of the searchers is a "black box" with respect to the manager, hence the name *Simple **B**lack **B**ox* Pattern (Two-B). The Evaluator blocks have also been removed thus the feedback loop present in both the *All-Seeing Eye* and the *Commentator* has been removed altogether.

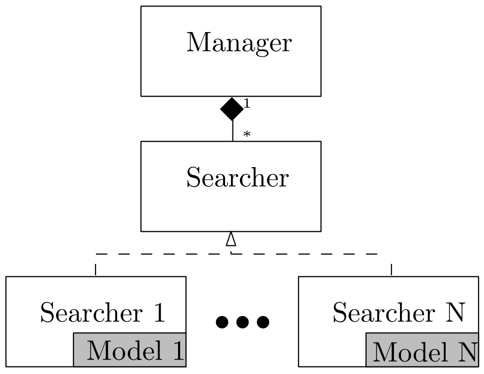

Figure 3: *Simple Black Box* Pattern

4.1.4 Examples

Triggered Hypermutation already implements the Two-B pattern. Its switching logic suits the *Simple Black Box* pattern, since the Manager does not require any knowledge of the model itself. The Manager simply keeps a running average of the fitness values which the Searcher presents to it. One could imagine alternative triggers, based on population metrics, for example hypermutation could be triggered when population diversity drops below a threshold. This would not be suitable for the simple Two-B pattern, since such information is not available.

The aim of **ARPSO** is to switch Searchers when search capacity is lost. In its standard form, it bases this decision on measurements of particle diversity and so requires knowledge of its Searchers' Models. However, Van Den Bergh proposes the Objective Function Slope measure as a way to detect loss of search capacity [1]. This measure is based solely on the cost of the best solutions in successive iterations and so a variant of ARPSO based on the Objective Function Slope measure would be suitable for implementation using the Two-B pattern. Canonical **CPSO** does not require direct model measurements, basing the decision to switch between Searchers on time (number of iterations) and fitness and so would also be suitable for the Two-B pattern.

Memetic GP and **Memetic MOEA** would benefit from implementing Two-B instead of the *Commentator* as the searcher switch logic they implement is static. Thus, the evaluator blocks can be eliminated altogether without any impact on the behaviour of the manager.

Similarly, **GP with simplification** would benefit from the Two-B pattern, as the switching logic does not need to rely on any knowledge of the model, the manager decides on applying simplification at pre-defined regular intervals and with a set probability. If the model is embedded in the searcher, it becomes the arithmetic simplifier's internal responsibility to ensure that the simplified solution is indeed more compact than the original.

4.1.5 Consequences

Encapsulating models inside searchers and eliminating evaluators decreases structural complexity as it removes two layers of abstraction, relative to the *Commentator*. This design choice also loosens coupling as there is no direct connection between the models and the manager. This decoupling would allow for the Searchers used within a hybrid algorithm to be changed without modification to the Manager. The disadvantage of Two-B is behavioural - this configuration restricts the possible types of hybridisation as the Manager cannot make decisions based on Searcher state.

4.2 The Utility-based Black Box (Three-B) Pattern

4.2.1 Intent

The main goal of this pattern is to allow a Manager to make **informed decisions** based on the suitability of its Searchers for the given problem, while allowing the reuse of Manager and Searchers as components in other hybrid algorithms, without the Manager knowing the details of the inner working of Searchers or how their quality is assessed.

4.2.2 Forces acting

As in Two-B, a hybrid may be comprised of **Heterogeneous Searchers**. Managers cannot make different types of measurements of different Models without becoming coupled to them, preventing reuse.

Performance Prediction - In some scenarios (e.g. dynamic problems, complex fitness landscapes, hybrids consisting of multiple Searchers) the fitness of solutions generated by a Searcher in the past may be a poor predictor of its future performance. Indeed, if a hybrid can only select a searcher based on past performance, then it has no basis for choosing an initial searcher. To allow high quality decisions about which Searcher to deploy given the current problem state, it may be necessary to give the Manager more information about Searcher states.

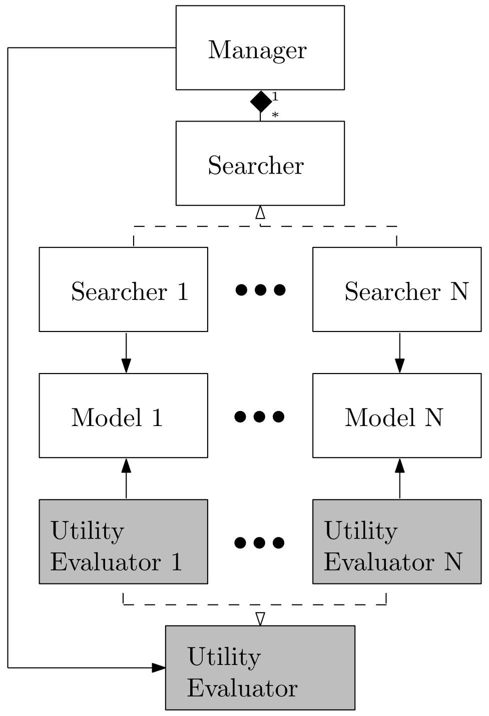

Figure 4: *Utility-based Black Box* Pattern

4.2.3 Structure

In order for a Manager to be able to manage an arbitrary collection of Searchers, it must be agnostic as to the details of their Models. While our proposed Two-B pattern allows this, it does not allow the Manager to choose between Searchers based on anything other than fitness information which may not be suitable for all hybrids. To overcome this limitation, we propose the pattern shown in Fig. 4 in which the Manager can make Model measurements through a consistent Utility Evaluator interface. Due to the potential heterogeneity of Models, the Utility Evaluator cannot be capable of making all the measurement types possible through Evaluators in the *Commentator* as this would require Utility Evaluators to know how to be able to make an arbitrary number of measurements, some inappropriate for their Model types (the Utility Evaluator of a non-population-based Searcher should not, for instance, be required to measure population diversity). Therefore, every measurement type required by the Utility Evaluator should be both *generalisable* to arbitrary Models and *comparable* between Models. It is the responsibility of each concrete Utility Evaluator to know how to make these measurements for their corresponding Model. The responsibility of the Manager becomes to decide which Searcher component to utilise given the Utility scores for each Searcher on the current problem state. Because the Searchers remain black boxes from the perspective of the Manager, we describe this as the *Utility-**b**ased **B**lack **B**ox Design Pattern* (Three-B).

4.2.4 Examples

The **HyFlex** framework can be viewed as an implementation of the Three-B pattern. The Manager has a black-box view of the Searchers, but can access information about each Searcher's qualitative *type*. The focus of the framework is on combining low-level heuristics suitable for modifying a single solution, so supported *types* include operators for local search, crossover and mutation. Knowledge of these types can be used to infer the suitability of particular low-level heuristics either at design-time or, given the algorithm and problem state, at run-time. Type information is comparable between algorithms and so could be viewed as one method of utility evaluation. Unlike the Three-B pattern, the framework in [10] requires a shared model of the problem domain for the use of all other algorithm components, whereas Searchers in Three-B maintain independent Models.

ARPSO uses population diversity as a proxy for search capacity (the capacity of the algorithm to discover new, high-quality solutions), with a low-diversity population less likely to find good new solutions in future than a high-diversity population. As discussed previously, population diversity is not an appropriate quality metric, as it is not applicable to non-population-based Models but, in principle, search capacity is. If a metric for comparing search capacity between Model types was available, then ARPSO could be implemented in terms of it.

GP with fitness sharing also measures population diversity to evaluate the search status and this serves as the basis for the decision on what niche size to use for fitness sharing. If this diversity evaluation was generalised to a utility metric, additional component algorithms could be introduced (i.e. alternative methods for limiting code growth or having other objectives) and seamlessly selected between, so that better quality solutions are obtained.

4.2.5 Consequences

Use of the Three-B pattern when creating hybrid metaheurisics would result in decreased design effort: the reusability of Managers and Searchers would allow new algorithmic hybrids to be created quickly, easily and, potentially, automatically. This less challenging design barrier would simplify the tuning of hybrids to small classes of problems and specific problem instances, increasing algorithmic performance. However, to realise Three-B, Utility metrics comparable across Model types would need to be designed.

5. CONCLUSIONS AND FUTURE WORK

We extracted two so-called antipatterns for hybrid metaheuristics, the *All-Seeing Eye* and the *Commentator*, and discussed their shortcomings in terms of complexity and coupling, using seven exemplars from evolutionary algorithms, genetic programming and particle swarm optimisation. In response to these problems, we proposed two new Black-Box patterns, Two-B and Three-B, that strike a balance between the previously mentioned opposing forces. We discussed how each exemplar would benefit from, or in the case of **HyFlex** are conceptually similar to, one of these new patterns.

We propose that Two-B is applied in situations where purely fitness based selection is applied to decide between the different searchers. However, Two-B has a behavioural disadvantage: it restricts the possible types of hybridisation as the choice of component algorithm cannot be based on Searcher state. Therefore, when Searcher state should be the basis for the decision on component algorithm and comparable utility scores for different searchers are available, the Three-B pattern should be applied. Three-B also allows for competition between various Searchers, thus simplifying the decision logic of the manager.

In the future, we plan to focus on designing a standard quality metric to solve the alignment problem between evaluator criteria and allow searchers to be arbitrarily combined within the Three-B pattern.

6. REFERENCES

[1] F. V. D. Bergh. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, 2006.

[2] H. G. Cobb and J. J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proceedings of the 5th international conference on genetic algorithms*, pages 523–530, 1993.

[3] A. Ekárt. Shorter fitness preserving genetic programs. In *Artificial Evolution*, volume 1829 of *LNCS*, pages 73–83, 2000.

[4] A. Ekárt and S. Z. Németh. Maintaining the diversity of genetic programs. In *European Conference on Genetic Programming*, volume 2278 of *LNCS*, pages 162–171, 2002.

[5] L. Ferariu and A. Patelli. Genetic programming for system identification. In P. Cong-Vinh, editor, *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification*, pages 135–168. 2012.

[6] H. Fu, P. R. Lewis, B. Sendhoff, K. Tang, and X. Yao. What are dynamic optimization problems? In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1550–1557. IEEE Press, 2014.

[7] P. R. Lewis, P. Marrow, and X. Yao. A diversity dilemma in evolutionary markets. In *Proceedings of ICEC 2011: The Thirteenth International Conference on Electronic Commerce*. ACM Press, 2011.

[8] B. Liu, L. Wang, Y. H. Jin, F. Tang, and D. X. Huang. Improved particle swarm optimization combined with chaos. *Chaos, Solitons & Fractals*, 25(5):1261–1271, 2005.

[9] R. Morrison and K. De Jong. Triggered hypermutation revisited. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 1025–1032. IEEE Press, 2000.

[10] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke. Hyflex: A benchmark framework for cross-domain heuristic search. In J.-K. Hao and M. Middendorf, editors, *Evol. Comp. in Combinatorial Optimization*, volume 7245 of *LNCS*, pages 136–147. 2012.

[11] L. Rachmawati and L. D. Srinivasan. Multiobjective evolutionary algorithm with controllable focus on the knees of the Pareto front. *IEEE Transactions on Evolutionary Computation*, 13(4):810–824, 2009.

[12] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer - the ARPSO. Tech. Rep 2, Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, 2002.

[13] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.