The ‘Representative’ Metaheuristic Design Pattern

Jerry Swan
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
jsw@cs.stir.ac.uk

Zoltan A. Kocsis
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
zak@cs.stir.ac.uk

Alexei Lisitsa
Department of Computer Science
University of Liverpool
L69 7ZF UK
A.Lisitsa@liverpool.ac.uk

Categories and Subject Descriptors

I.1 [**SYMBOLIC AND ALGEBRAIC MANIPULATION**]: General|Expressions and Their Representation|Simplification of expressions.

Keywords

Term Rewriting; Design Patterns; Evolutionary Computing; Metaheuristics.

1. PROBLEM STATEMENT

The ‘Representative’ pattern is applicable when it is desirable to eliminate redundancy in the search process:

- It is often the case that some function f of interest in optimization gives a many-to-one mapping, i.e. it induces equivalence classes over the image of f. If f is a fitness function, this can lead to plateaus in the fitness landscape.
- It may be that the elimination of redundancy allows search to be performed in a smaller (‘quotient’) space that can be searched using methods (possibly even *exact* techniques) not applicable to the original space.
- In the case of GP-trees, syntactically inequivalent but semantically equivalent representations (e.g. $x + x, 2 * x$) can lead to a lack of gradient in genotype-to-phenotype mappings, which *may* make the space of programs harder to search effectively.

2. THE SOLUTION

Transform representations into a normal or canonical form.

It is useful here to adopt the terminology of the computer algebra community in distinguishing between these forms [6]. Consider some representation $\mathcal{R}$ with equivalence relation $\sim$. Syntactic equivalence on $\mathcal{R}$ is denoted by $\equiv$.
For some distinguished element $0_{\mathcal{R}} \in \mathcal{R}$, a *normal form* of $\mathcal{R}$ is the output of a procedure $N : \mathcal{R} \to \mathcal{R}$ such that for all $x \in \mathcal{R}$

$$N(x) \sim x$$

$$x \sim 0_{\mathcal{R}} \implies N(x) \equiv N(0_{\mathcal{R}})$$

A *canonical form* of $\mathcal{R}$ is the output of a procedure $C : \mathcal{R} \to \mathcal{R}$ such that for all $x, y \in \mathcal{R}$

$$C(x) \sim x$$

$$x \sim y \implies C(x) \equiv C(y)$$

Since canonical implies normal but not conversely, even if both forms exist it is generally less computationally-expensive to compute normal forms. For example, the normal form procedure for univariate polynomials is given by recursively distributing products over sums, collecting like terms and eliminating terms with zero coefficients. In this case, the canonical form requires the additional step of applying a fixed ordering to terms (i.e. by degree-order). For boolean functions, both DNF and CNF are ubiquitous normal forms. If we fix the ordering of propositional letters and consider minimal equivalents, one may obtain a canonical form. Moreover, boolean polynomials (a.k.a. Zhegalkin polynomials [17]) may serve as an alternative algebraic normal form (ANF). Note also that functions on fixed-width integers (e.g. 32-bit) can be treated as boolean functions. Table 1 gives references for representative forms for a variety of structures. The existence of normal and canonical forms (jointly, ‘representative forms’ hereafter) depends on the algebraic properties of the representation. The representative forms of many structures of interest in metaheuristics can found via the application of *term rewriting* [2], which proceeds via the iterated application of a *Term Rewriting System* (TRS). For a set of elements of a structure of type S (e.g. polynomials, operators etc), let the set of all possible combinations of elements of S (e.g. expression trees, operator sequences etc) be denoted by Σ^*. A *rewriting system* $(\mathcal{S}, \prec)$ is then a set of ordered pairs $(l, r) \in \Sigma^* \times \Sigma^*$ subject to the condition $l \prec r$, where $\prec$ is a *reduction ordering* that determines which of two expressions is to be considered ‘more simple’. The elements of $\mathcal{S}$ are called *rewrite-rules*. and the idea is that we can replace occurrences of l in a r in the hope of that we will eventually arrive at a maximally simplified expression. Under certain conditions on $\prec$ [2], this hope is justified: the iterated application of rewrite rules will always yield a unique representative. Such a rewriting system is said to be *confluent*.

3. THE CONSEQUENCES

- Transformations into representative form can be used to reduce the size of the state-space graph. By converting solution states (or operator sequences, as per the example below) into their representative forms, one is

Algebraic Structure	Representative Algorithm
Monoid or Group	[8]
Boolean function	[3]
Polynomial	[6, 11]
Polynomials modulo an Ideal	[4]
Rational function	[6, 11]
Truncated power series	[11]

Table 1: Computing representatives of various algebraic structures

effectively working in a smaller (a.k.a. quotient) search space.

- Elimination of redundancy is not necessarily advantageous. Reduction to representative form by removing non-functional elements from evolutionary computation (e.g. $0 * s$ for any subexpression s) can reduce neutrality in genotype-to-phenotype mappings. The presence of such redundancy is argued by some to be desirable [16], since it is argued that non-functional nodes can nonetheless contribute to good quality solutions in subsequent generations. However, there is no consensus in this respect [10].

4. EXAMPLES

- In [14], a canonical form for rational functions over $\mathbb{Q}$ is given in terms of their roots. In constrast to the corresponding GP-tree, it is possible to make small changes to this representation.
- Finite state automata are a ubiquitous representation in evolutionary computation [9]. It is well-known that there is a unique canonical form for Deterministic Finite-state Automata (subject to the imposition of a specified ordering on transitions), and there are a number of DFA-minimization algorithms (having $O(n * log(n))$ average-case behaviour) for achieving this. An empirical comparison of several of these algorithms is given in [1].
- Binary decision diagrams provide a canonical form for boolean functions [3]. In [15], Genetic Programming is augmented via the use of binary-decision diagrams to share isomorphic sub-graphs across the population, thereby solving the 20-multiplexer problem.
- Techniques such as tabu search have traditionally exploited domain knowledge about operators (e.g. prohibiting the inverse of an operation for some number of iterations). Any set of operators can be considered as a *monoid* under concatenation, and we can use the *Knuth Bendix Algorithm* to convert a sequence of operators into its length-minimal canonical form [13]. By this means, as an offline activity it is possible to eliminate all operator sequences that represent cycles in the state-space graph.
- The λ-calculus with β-reduction as a rewrite rule is confluent, so normal forms are unique whenever they exist [5].Normally, uniqueness is up to renaming of bound variables only, but it can be extended to syntactic equality using the method of de Bruijn indices [7]. There are multiple algorithms for finding normal forms, their differences in reduction strategy dictated by the distinction between *lazy* and *eager* evaluation in functional programming languages. A non-deterministic optimal algorithm for finding normal forms is given in [12].

5. REFERENCES

[1] Marco Almeida, Nelma Moreira, and Rogério Reis. On the performance of automata minimization algorithms. Tech report, Universidade do Porto, 2007.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, 1998.

[3] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[4] Bruno Buchberger. An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *J. Symb. Comput.*, 41(3-4):475–511, 2006.

[5] Alonzo Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):pp. 472–482, 1936.

[6] James Harold Davenport, Y Siret, E. Tournier, and A. Davenport. Computer algebra : systems and algorithms for algebraic computation, 1993.

[7] N.G de Bruijn. Lambda calculus notation with nameless dummies. *Indagationes Mathematicae (Proceedings)*, 75(5):381 – 392, 1972.

[8] D. B. A. Epstein, D. F. Holt, and S. E. Rees. The use of Knuth-Bendix methods to solve the word problem in automatic groups. *J. Symb. Comput.*, 12(4-5):397–414, October 1991.

[9] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, Chichester, WS, UK, 1966.

[10] Edgar Galván-López, Riccardo Poli, Ahmed Kattan, Michael O'Neill, and Anthony Brabazon. Neutrality in evolutionary algorithms... what do we know? *Evolving Systems*, 2(3):145–163, 2011.

[11] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for computer algebra*. Kluwer, 1992.

[12] John Lamping. An algorithm for optimal lambda calculus reduction. pages 16–30. ACM Press, 1990.

[13] J. Swan, M. Edjvet, and E. Özcan. Augmenting metaheuristics with rewriting systems. Technical Report CSM-197, Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, Scotland, January 2014.

[14] John R. Woodward and Ruibin Bai. Canonical representation genetic programming. In *GEC Summit*, pages 585–592, 2009.

[15] Masayuki Yanagiya. Efficient genetic programming based on binary decision diagrams. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 234–239, Perth, Australia, 29 November - 1 December 1995. IEEE Press.

[16] Tina Yu and Julian Miller. Finding needles in haystacks is not hard with neutrality. In James A Foster et al, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2002.

[17] Ivan Ivanovich Zhegalkin. On the technique of calculating propositions in symbolic logic. *Matematicheskii Sbornik*, 43:9–28, 1927.