Metaheuristic Design Pattern: Candidate Solution Repair

Krzysztof Krawiec
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar St, Cambridge, MA
kkrawiec@csail.mit.edu

1. PROBLEM STATEMENT

Many problems solved with metaheuristic algorithms are *constrained*, i.e., problem formulation includes conditions (usually logical predicates) that have to be fulfilled by a candidate solution to be *feasible*. As a result, the entire search space S of candidate solutions is partitioned into *feasible solutions* $F \subset S$ and *infeasible solutions* $I \subset S$, $F \cap I = \emptyset$.

As the goal is to find the optimal *and* feasible solution (or, more often in practice, a well-performing suboptimal and feasible solution), it is desirable to traverse S using search operators that are guaranteed to produce feasible solutions (i.e., $s : S \to F$). However, designing operators with this guarantee can be challenging, for instance when feasible solutions are few and far in between (i.e., the ratio $|F|/|S|$ is small), or when the structure of S is such that the feasible solutions are frequently neighbored by the infeasible ones. Moreover, even if such an operator can be designed, then it may be biased in an undesirable way (in the sense of the probability distribution of produced candidate solutions).

This makes it necessary to equip an algorithm with some means of handling the infeasible solutions. Discarding them is rarely a good idea for at least two reasons. Firstly, with the discarded candidate solutions some knowledge gathered in hitherto search is also lost. Secondly, this may prevent the algorithm from visiting the yet unexplored and potentially promising parts of the search space. As an alternative, one may extend the definition of the objective function so that it works 'reasonably' beyond the feasible region; for instance, an infeasible candidate solution to a knapsack problem may be evaluated according to how much it violates the knapsack capacity ([3], p. 149). For many problems however coming up with such an extended and uncontroversial objective function may be difficult.

Another way of handling infeasible solutions is penalization by introducing appropriate terms into objective function f, or by defining additional objectives that implement constraint violation measures. This proceeding assumes however that f can be applied to unfeasible solutions, i.e., that the constraints are in some sense *soft*. Though this is often technically true (e.g., in many continuous optimization benchmarks f is defined in almost entire $\mathbb{R}^n$ space), f applied to an unfeasible solution may return a nonsensical value and potentially deceive the search process. This can be solved by not applying f to infeasible solutions altogether, i.e., by relying on penalties alone for such candidate solutions, but this typically implies that an infeasible solution will be always considered as worse than any feasible solution, which can again prevent exploration of certain parts of the search space.

2. THE SOLUTION

In the face of above controversies, it has been often recommended to *repair* the unfeasible solutions. A repair operator is any operator O guaranteed to fulfill

$$O(x) \in F$$

for any given infeasible solution $x \in I$.

There are at least two major ways in which such an operator can be employed:

1. In the most obvious scenario, the resulting repaired solution $x' = O(x)$ replaces the infeasible one, and the latter one is discarded. This proceeding is useful in problems where the objective function is capable of evaluating infeasible solutions. For instance, in the Traveling Salesperson Problem (TSP), one can still calculate route length even if it visits some cities more than once [9].

2. One can alternatively use the repaired solution $x' = O(x)$ only as a means for obtaining the value of $f(x)$, which otherwise could not be calculated [4]. More specifically, x remains a working solution, but receives the objective value of its repaired version, i.e.:

$$f(x) = f(O(x))$$

3. CONSEQUENCES

The availability of a repair operator for a given problem has several implications. Firstly, it gives the experimenter more freedom in designing search operators, which do not have to always produce feasible solutions anymore. In some cases, it may pay off to design search operators that *notoriously* yield infeasible solutions, in which case repair becomes a norm rather than an exception, and the repair operator gets to be a more important 'search driver' than the primary search operator(s).

Though usually a necessity, solution repair can be seen in a broader context as a way to make a search process more *responsive* to the feedback resulting from the evaluation of candidate solutions. On the other hand, the information about constraint violation can be considered as a manner of making the search process more *informed*.

Note that repair in the above sense can be seen as yet another search operator, capable of operating in I (as opposed to conventional search operators which normally are required to work only in F). Therefore, the characteristics of a repair operator becomes a (potentially important) part of algorithm's *search bias*.

In the context of evolutionary computation, the two modes of operation listed in the previous section can be respectively deemed *Lamarckian repair* and *Baldwinian repair*, as the former one employs the feedback received from the environment (i.e., the information about violating a constraint) to explicitly modify (replace) the solution, while in the latter scenario the only effect is in the objective (fitness) value.

Sometimes it may be difficult to delineate repair from other components of a metaheuristic algorithm. In some cases, repair is an indispensable element of genotype-phenotype mapping. For instance in grammatical evolution [7], solutions (genotypes) are vectors of numbers, each of them determining the choice of a grammar production to be used when deriving the phenotype (program). When such a number is greater than the number of productions available at a given stage of program derivation, it is to be treated by a modulo n operator. That operator can be seen as a repair operator in this context.

4. EXAMPLES

- Numerical optimization abounds in various methods for solution repair (see, e.g., [4] for a review).
- A simple form of solution repair is clamping (trimming) the value of a modified variable to a predefined range (domain). Though in most cases the ranges are part of problem statement, sometimes the reasons are more subtle. For instance in [8], we clamp the variables representing the parameters of Othello strategies. However, we do so not because the fitness function is *inherently* unable to handle the values outside the interval, but to avoid an indefinite growth of variables that may not bring qualitative changes in players' behavior (because candidate solutions are functionally equivalent up to linear scaling). Thus, infeasibility is not the only motivation for repair.
- Intron removal in genetic programming (e.g., [1]) can be considered as a form of repair (notwithstanding that it has been shown that some amount of introns may be beneficial for GP search).
- Simplification algorithms used in certain variants of genetic programming to reduce solution size (e.g., [5, 6]) can be seen as as a form of repair: they do not affect the semantics of a program, but change its syntax and thus make it possible for it to meet, e.g., an upper limit imposed on program size.
- Methods have been proposed that avoid solution repair by using the information on constraint violation to guide the search process [2].

Acknowledgment

The author acknowledges financial support from the Polish-U.S. Fulbright Commission and from grants no. DEC-2011/01-/B/ST6/07318 and 91507.

5. REFERENCES

[1] Thomas Haynes. Duplication of coding segments in genetic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 1, pages 344–349, Portland, USA, 4-6 August 1996. AAAI Press / MIT Press.

[2] Nanlin Jin, Edward Tsang, and Jin Li. A constraint-guided method with evolutionary algorithms for economic problems. *Applied Soft Computing*, 9(3):924–935, 2009.

[3] Sean Luke. *Essentials of Metaheuristics*. lulu.com, first edition, 2009. Available at http://cs.gmu.edu/∼sean/books/metaheuristics/.

[4] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In John R. McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Proc. of the 4th Annual Conf. on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press.

[5] Alberto Moraglio, Krzysztof Krawiec, and Colin Johnson. Geometric semantic genetic programming. In Christian Igel, Per Kristian Lehre, and Carsten Witt, editors, *The 5th workshop on Theory of Randomized Search Heuristics, ThRaSH'2011*, Copenhagen, Denmark, July 8-9 2011.

[6] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric semantic genetic programming. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31. Springer, 2012.

[7] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *First European Workshop on Genetic Programming 1998*, pages 83–95, Berlin, 1998. Springer.

[8] M. Szubert, W. Jaśkowski, and K. Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: A case study for othello. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(3):214–226, 2013.

[9] Tim Walters. Repair and brood selection in the traveling salesman problem. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 813–822, Berlin, 1998. Springer. Lecture Notes in Computer Science 1498.