A Model-Based Genetic Algorithm Framework for Constrained Optimisation Problems

Mark Lawrenson, Tommaso Urli, Philip Kilby
CSIRO Data61 & the Australian National University
Canberra, Australia
firstname.lastname@data61.csiro.au

ABSTRACT

Two major challenges are presented when applying genetic algorithms (GAs) to constrained optimisation problems: modelling and constraint handling. The field of constraint programming (CP) has enjoyed extensive research in both of these areas. CP frameworks have been devised which allow arbitrary problems to be readily modelled, and their constraints handled efficiently. Our work aims to combine the modelling and constraint handling of a state-of-the-art CP framework with the efficient population-based search of a GA. We present a new general hybrid CP / GA framework which can be used to solve any constrained optimisation problem that can be expressed using the language of constraints. The efficacy of this framework as a general heuristic for constrained optimisation problems is demonstrated through experimental results on a variety of classical combinatorial optimisation problems commonly found in the literature.

CCS CONCEPTS

•**Theory of computation** → *Random search heuristics;* •**Computing methodologies** → *Randomized search; Modeling methodologies;* •**Applied computing** → *Operations research; Decision analysis;*

KEYWORDS

Genetic Algorithms, Hybridization, Constraint Handling, Combinatorial Optimization, Modelling, Meta-heuristics

ACM Reference format:
Mark Lawrenson, Tommaso Urli, Philip Kilby. 2017. A Model-Based Genetic Algorithm Framework for Constrained Optimisation Problems. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3076041

1 INTRODUCTION

Genetic algorithms (GAs, [1, 3]) are a family of optimisation meta-heuristics based on the evolutionary principle of *survival of the fittest*. The way GAs solve an optimisation problem is by evolving a *population* of candidate solutions represented as a sequences of bits, and associated with a *fitness* function that reflects the solution quality. The population is evolved by means of operators drawn from genetics. The goal is to recombine the fittest individuals in the population to create a new generation of candidate solutions, hopefully with a higher overall fitness. GAs have been successfully used in applications ranging from logistics [5] to renewable energy [6]. Constraint programming (CP, [2]) is a paradigm for modelling and solving CSPs and COPs. In CP, a problem is modelled in terms of decision variables and constraints. Decision variables are associated with initial domains (sets of legal values), while constraints represent logic relations between variables which further restrict the allowed set of values that they can take. Problems are solved using a combination of backtracking tree-search and constraint propagation. The approach is complete: the search proceeds until the domains of the variables are singletons satisfying all the constraints, or it is proven that there is no solution. CP has had significant success in dealing with combinatorial problems in routing and scheduling. Modern CP frameworks provide large libraries of constraints, which allow to easily model arbitrary COPs.

The goal of this work is to propose a hybrid framework to combine genetic algorithms with constraint programming. By bringing together these orthogonal technologies, we hope to leverage the modelling and constraint handling capabilities of CP, and the efficient, parallel, population-based search strategies of GAs. We show the applicability of our approach by running an experimental analysis on a number of classical combinatorial optimisation problems.

GECCO '17 Companion, Berlin, Germany
 978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3076041

2 OUR FRAMEWORK

Our approach retains the overall logical structure of a classic GA, with the difference that each individual in the population is always a *feasible* solution of a constraint model of the problem being solved. By this we mean that, at all stages, an individual is represented as a CP solution, rather than as a string of values as is customary in GAs. The feasibility of the individuals is operationally guaranteed at all times by enforcing constraint propagation whenever a new individual is created or an existing solution is modified. In this section, we briefly describe how the various components of 1 are implemented to leverage the existence of an underlying constraint model and a propagation engine.

2.1 Initialiser

The Initialiser routine first creates μ unassigned CP solutions. Then, each solution is built through a depth-first tree-search in which the next variable to be assigned is chosen as the one with the highest *accumulated failure count* (AFC), and the value to assign to it is chosen uniformly at random. Constraint propagation is enforced at each step of the tree-search, so that the initial population consists only of feasible solutions.

Algorithm 1 Classical Genetic Algorithm

Require: Population size: μ
  procedure GeneticAlgorithm
    $P_I \leftarrow$Initialiser(μ) ▷ P_I is the incumbent population
    while stopping criteria is not met **do**
      Clear(P_O) ▷ P_O is the offspring population
      while $|P_O| < \mu$ **do**
        $I_1, I_2 =$Select(P_I)
        $P_O \leftarrow$Crossover(I_1, I_2)
      end while
      Mutate(P_O)
      Evaluate(P_O)
      $P_I =$ Elitism(P_I, P_O)
    end while
    return Best(P_I) ▷ Return the fittest individual
  end procedure

2.2 Selection

Since the selection of parents does not modify or generate new individuals, any operator from the GA literature can be used. In our current implementation, we use a fairly standard *roulette wheel* selection mechanism, in which the parents are chosen proportionally to their fitness.

2.3 Crossover

Once two parents have been selected, we combine them to obtain a new offspring. We initialise the offspring with an empty CP solution, and then assign its variables using a depth-first tree-search which takes the two parents as inputs, and works as follows. The variable selection heuristic chooses the next variable v to assign uniformly at random. The value selection heuristic biases the search towards the values of the variable v in the parents. In particular, the values to assign to v are attempted in one of the following orders

$$ord_1 = \langle val(v_1), val(v_2), \text{values in } dom(v) \text{ in random order} \rangle$$
$$ord_2 = \langle val(v_2), val(v_1), \text{values in } dom(v) \text{ in random order} \rangle$$

where $val(v_x)$ represents the value of the variable v in the parent x, $dom(v)$ represents the domain of v in the offspring, and the orders ord_1 and ord_2 are chosen with equal probability. Since constraint propagation is enforced at each step of the tree-search, the new generation can only consist of feasible solutions.

2.4 Mutation

After a new offspring is generated, we randomly mutate each of its variables with a *mutation probability* (p_m), a parameter of the solver. Then, we use a random AFC-driven depth-first tree-search to assign (potentially different) values to those variables. We constrain the obtained individual to be at least as fit as it was before mutation.

3 EXPERIMENTAL ANALYSIS

We conduct an experimental analysis to demonstrate the general applicability of our CP / GA framework (ConGA) to solve a variety of combinatorial optimisation problems. To this end, we model four classical combinatorial optimisation problems in the Gecode constraint programming framework, and compare the performance of our approach with the off-the-shelf B&B tree-search strategy provided by Gecode, and a freely-available CP-based large neighbourhood search (LNS) strategy based on Gecode itself (see [4]). The considered problems are: the bin packing problem (BPP), the capacitated vehicle routing problem (CVRP), the job shop scheduling problem (JSSP) and the travelling salesman problem (TSP). The first three are considered both in their original form, and in a relaxed form where some of the hard constraints are modelled as penalties.

Although there are small differences in the performances of the approaches across the various problem domains, the results paint a reasonably consistent picture: ConGA almost consistently outperforms B&B, and LNS almost consistently outperforms ConGA. This is reflected in Figure 1 which shows the distribution of normalised costs across all of the instances for the various algorithms and model variants.

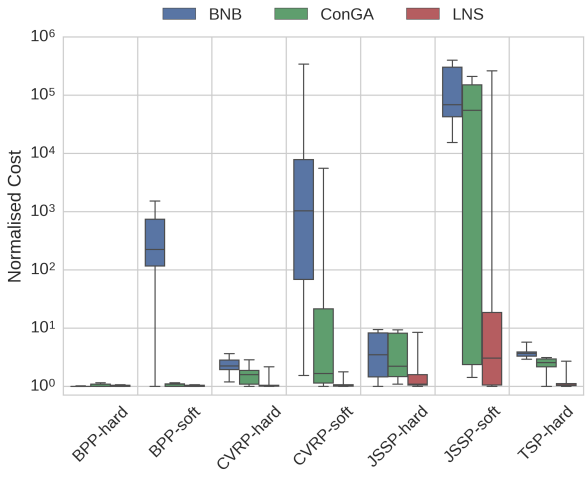

Figure 1: Performance across problems.

The advantage of ConGA over B&B was expected, since B&B is slower to converge but, being a complete method, provides guarantees. The results against LNS show that ConGA is not competitive yet as an off-the-shelf solution for solving COPs. Nevertheless, the fact that parallelisability, one of the most compelling features of GAs, was not exploited and the fact that the overall approach was based on a very simple GA scheme, motivates more research in this direction.

REFERENCES

[1] Lawrence Davis. 1991. Handbook of genetic algorithms. (1991).
[2] Francesca Rossi, Peter Van Beek, and Toby Walsh. 2006. *Handbook of constraint programming*. Elsevier.
[3] Kumara Sastry, David E Goldberg, and Graham Kendall. 2014. Genetic algorithms. In *Search methodologies*. Springer, 93–117.
[4] Tommaso Urli, Jana Brotánková, Philip Kilby, and Pascal Van Hentenryck. 2016. Intelligent Habitat Restoration Under Uncertainty.. In *AAAI*. 3908–3914.
[5] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. 2013. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research* 40, 1 (2013), 475–489.
[6] Junhua Wu, Slava Shekh, Nataliia Y Sergiienko, Benjamin S Cazzolato, Boyin Ding, Frank Neumann, and Markus Wagner. 2016. Fast and effective optimisation of arrays of submerged wave energy converters. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 1045–1052.