Instance Similarity and the Effectiveness of Case Injection in a Genetic Algorithm for Binary Quadratic Programming

Jason Amunrud and Bryant A. Julstrom
Dept. of Computer Science, St. Cloud State University
St. Cloud, MN 56301 USA
amja0201@stcloudstate.edu, julstrom@stcloudstate.edu

ABSTRACT

When an evolutionary algorithm addresses a sequence of instances of the same problem, it can seed its population with solutions that it found for previous instances. This technique is called case injection. How similar must the instances be for case injection to help an EA's search? We consider this question by applying a genetic algorithm, without and with case injection, to sequences of instances of binary quadratic programming. When the instances are similar, case injection helps; when the instances differ sufficiently, case injection is no help at all.

Categories and Subject Descriptors

G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms

Algorithms

Keywords

Case injection, population seeding, binary quadratic programming, instance similarity

1. INTRODUCTION

In *case-based reasoning*, good solutions to similar instances of the target problem are provided to a heuristic addressing a current instance. Case-based reasoning can be applied to an evolutionary algorithm as it seeks good solutions to a sequence of similar instances of a problem. In this setting, the technique is called *case injection*. Case injection arises naturally in EAs for dynamic problems; as the problem changes, previous good solutions may acquire new relevance.

We consider how similar instances must be for case injection to help an EA's search, using a genetic algorithm for binary quadratic programming. Presented with a sequence of instances of the problem, the GA can attack them independently, or it can seed its population with the best solutions of all the previous instances in the sequence.

Copyright is held by the author/owner(s).
GECCO'06, July 8–12, 2006, Seattle, Washington, USA.
ACM 1-59593-186-4/06/0007.

2. BINARY QUADRATIC PROGRAMMING

Let M be a symmetrical $n \times n$ matrix and $\vec{x}$ a binary vector of length n. *Unconstrained binary quadratic programming* (BQP) seeks an assignment of values to $\vec{x}$ that maximizes this sum:

$$f(\vec{x}) = \vec{x}^t M \vec{x} = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j M_{ij} \qquad (1)$$

Given objects that have values individually and in pairs, we seek objects to maximize the sum of their individual and pair values, which may be negative, zero, or positive. BQP has long been known to be NP-hard [2].

3. A GENETIC ALGORITHM

Binary strings of length n encode solutions to a BQP instance of size n. Each element `c[`i`]` in a chromosome `c[·]` indicates the inclusion (`c[`i`]` $= 1$) or exclusion (`c[`i`]` $= 0$) of the corresponding "object" i. A chromosome's fitness is the value of the expression (1).

A generational genetic algorithm initializes its population either with random chromosomes or with random chromosomes augmented by the GA's best solutions on all previous instances; a boolean flag determines whether case injection is enabled. The GA chooses chromosomes to be parents in 2-tournaments, and it applies two-point crossover and position-by-position mutation independently; each offspring is generated by exactly one operator. The GA is 1-elitist, and it runs through a fixed number of generations. In the tests that Section 4 below describes, on BQP instances of size n, the GA's population contains n chromosomes. The probability that crossover generates an offspring is 70%, and the probability of mutation therefore 30%. In mutation, the probability that any one bit is flipped is $2/n$. The GA runs through $2n$ generations.

4. TESTING CASE INJECTION

Beasley's OR-Library[1] contains sets of BQP instances of a variety of sizes. These instances consist of integers, and they are sparse: approximately 95% of their values are zero; the remaining values were chosen at random between -100 and 100.

What it means for instances to be similar and how to measure that similarity are non-trivial questions (e.g., [1] [3]). Here, six sets of ten instances were derived from the first of the OR-Library's BQP instances of sizes $n = 100$

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

Table 1: The trials of the genetic algorithm, without and with case injection, on three sets of five BQP instances with $n = 100$ and small, medium, and large perturbations. For each instance and both GA versions, the table lists the best value over the 30 trials and the mean and standard deviation of the trials' results. For the GA with case injection, it also lists the percent change of the best and mean values from the best and mean without CI.

		Without case injection			With case injection				
Perturbations	Inst	best	mean	stdev	best	%change	mean	%change	stdev
In [-3,3]	1	7928	7715.20	154.79	7928	0.00	7715.20	0.00	154.79
	2	7749	7401.57	149.49	7751	0.03	7583.27	2.45	155.01
	3	7748	7507.83	196.14	7809	0.79	7621.07	1.51	153.34
	4	7719	7529.43	146.01	7749	0.39	7630.20	1.34	119.12
	5	7810	7423.00	193.29	7862	0.67	7661.40	3.21	138.45
In [-10,10]	1	7928	7715.20	154.79	7928	0.00	7715.20	0.00	154.79
	2	7728	7439.80	176.83	7748	0.26	7568.47	1.73	151.74
	3	7898	7640.03	163.22	7898	0.00	7755.03	1.51	126.09
	4	7768	7506.03	204.59	7819	0.66	7643.47	1.83	144.25
	5	7837	7437.40	222.39	7837	0.00	7647.07	2.82	118.92
In [-50,50]	1	7928	7715.20	154.79	7928	0.00	7715.20	0.00	154.79
	2	9774	9524.13	181.24	9774	0.00	9486.20	-0.40	209.75
	3	8961	8657.70	233.64	8947	-0.16	8536.47	-1.40	233.68
	4	9014	8576.50	180.06	9014	0.00	8554.57	-0.26	207.78
	5	10506	10144.77	266.83	10476	-0.29	10016.47	-1.26	277.36

and $n = 500$, three sets from each of these base instances. In each set, the first instance replicates the base instance. Nine more instances perturbed the base instance's non-zero values with random values between -3 and 3, -10 and 10, and -50 and 50. This yields sets of BQP instances of the two sizes with identical structures and whose differences we characterize as small, medium, and large.

The genetic algorithm of Section 3 was implemented in C++ and executed on a Pentium 4 processor with 1 Gbyte of memory, running at 2.53 GHz under Red Hat Linux 9.0. Without and with case-injection, the GA was run 30 times on the sets of instances of size $n = 100$ and ten times on the instances of size $n = 500$. Table 1 summarizes the results of the trials on the first five in each set of smaller instances. Results on the other instances were similar.

For the first instance of every set, there are no previous instances to provide seeding solutions, so on these instances, the GA's performance is always the same without and with case injection.

On the tabled instances with small and medium perturbations, case injection induced a consistent improvement in the GA's results. On the instances with small perturbations, the average improvement in the mean solution value was 2.13%. On the instances with medium perturbations, this average was smaller, but still 1.97%. On the instances with large perturbations, case injection offered no improvement.

The present case-injection strategy requires only minimal additional time and space and so is a useful investment when a GA addresses related problem instances. Note that "related" BQP instances might be generated in other ways, as by moving non-zero values among index pairs or by changing the number of non-zero values.

5. CONCLUSION

When an evolutionary algorithm is applied to related instances of a problem, it can seed its population with good solutions found on previous instances; this technique is case injection. For case injection to assist the EA's search on instances after the first, the instances must be similar. Using a genetic algorithm for binary quadratic programming, we have investigated how similar that might be.

From posted BQP instances of sizes $n = 100$ and $n = 500$, we have generated sequences of instances of identical structure but differing a little, some, and a lot in their values. Repeated trials of the GA, without and with case injection, on these instances indicated that: (1) When the instances' values differ only slightly, the instances are similar enough that case injection offers a consistent advantage. (2) When the instances' values differ more, case injection still offers a reduced advantage. (3) Large differences in the instance's values, even when the instances' structures are the same, render them too dissimilar for case injection to help. Nonetheless, case injection requires only a small computational investment, and this suggests using it whenever problem instances in a sequence might be sufficiently related.

6. REFERENCES

[1] Sushil J. Louis and John McDonnell. Learning with case injected genetic algoritms. *IEEE Transactions on Evolutionary Computation*, 8(4):316–328, 2004.

[2] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3:262–279, 1974.

[3] Wne-Jun Yin, Min Liu, and Cheng Wu. A genetic learning approach with case-based memory for job-shop scheduling problems. In *Proceedings of the 2002 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1683–1687, 2002.