Kent Academic Repository

Full text document (pdf)

Citation for published version

Johnson, Colin G. (2004) Using tabu search and genetic algorithms in mathematics research. In: Proceedings of the Fifth International Conference on Recent Advances in Soft Computing, December 2004, Nottingham Trent University.

DOI

Link to record in KAR

http://kar.kent.ac.uk/14050/

Document Version

UNSPECIFIED

Copyright & reuse
Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (eg Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research
The version in the Kent Academic Repository may differ from the final published version.
Users are advised to check **http://kar.kent.ac.uk** for the status of the paper. **Users should always cite the published version of record.**

Enquiries
For any further enquiries regarding the licence status of this document, please contact:
researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at **http://kar.kent.ac.uk/contact.html**

KAR Kent Academic Repository

University of Kent

Using Tabu Search and Genetic Algorithms in Mathematics Research

Colin G. Johnson
Computing Laboratory
University of Kent
Canterbury, Kent, CT2 7NF, England
e-mail: C.G.Johnson@kent.ac.uk

Abstract: *This paper discusses an ongoing project which uses computational heuristic search techniques such as tabu search and genetic algorithms as a tool for mathematics research. We discuss three ways in which such search techniques can be useful for mathematicians: in finding counterexamples to conjectures, in enumerating examples, and in finding sequences of transformations between two objects which are conjectured to be related. These problem-types are discussed using examples from topology.*

Keywords: experimental mathematics, genetic algorithms, tabu search, heuristics, heuristic search, topology

1 Introduction

This paper describes an ongoing project which is concerned with the application of computational search methods, such as genetic algorithms and tabu search, to mathematical problems.

Pure mathematics is typically concerned with the properties of particular types of mathematical objects. Many topics in mathematics (particularly the more algebraic and geometric topics) begin by defining a particular kind of object: a group, a ring, a manifold, a knot, a continuous function, ... From this definition we investigate the properties held by examples of the definition. For example we might ask questions of the form: when should we regard two objects as being the same? Given a way of transforming an object, when can we transform one object into another? How many objects are there of that kind, perhaps under some constraints?

In this paper we suggest that these kinds of questions can be approached by what could be thought of as a "data mining" approach: generate a search space of lots of examples of the objects in question, then use a computational search technique to search for objects with the properties of interest.

2 An example search space: topological knots

As an example search space we have been looking at an area of topology called *knot theory* which is concerned with *knotted* shapes (see [1, 7] for an introduction). These are of interest for several reasons: as interesting pure mathematical objects, because the underlying algebraic structures turn up in physics [2, 11], and because we can use them to understand problems in chemistry such as knotted DNA strands and polymeric materials [9].

A *knot* is an embedding of a loop in space; we can think of it as a closed loop of string which is tangled in some fashion. These are most easily represented by using two dimensional diagrams, which can be thought of as a *projection* of the three-dimensional diagram onto a two-dimensional

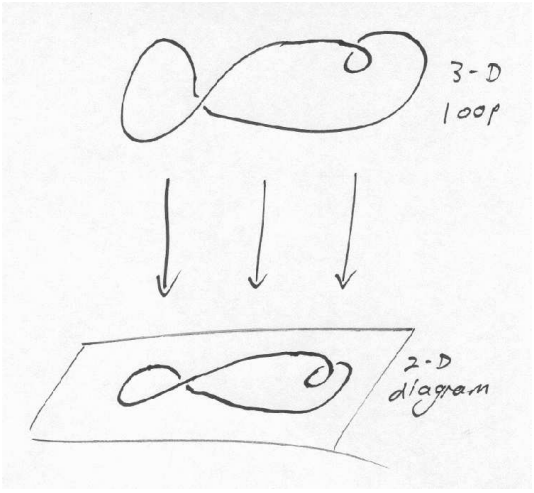

Figure 1: Forming a diagram from a knot via projection.

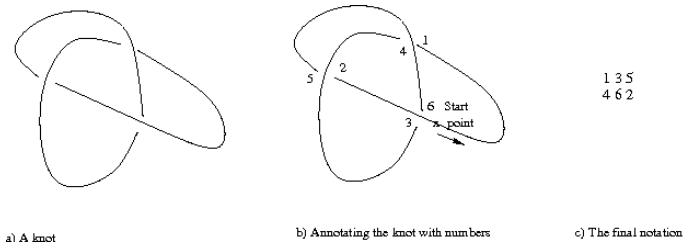

Figure 2: The Dowker-Thistlethwaite notation.

plane (diagram 1).

The collection of these diagrams, up to a certain number of crossings, forms our search space. There are a number of ways of storing these on the computer. One standard way of doing this is known as *Dowker-Thistlethwaite* notation [3]. In this notation we take a diagram and convert it into a sequence of numbers that track the over- and under-crossings as we move along the strand; each crossing point ends up with one even and one odd number associated with it. The process is illustrated in figure 2.

There are other ways of representing such knots; for example we can "stretch" the knot out into a sequence of linear strands, as illustrated in figure 3, and note the list of numbers corresponding to which strand crosses over which. To convert this sequence of strands into a conventional knot diagram we connect the two ends of each strand together to make a loop.

The aim of both of these processes is to convert a complex two-dimensional diagram into a sequence of numbers which capture the essence of the *topology* of the diagram; i.e. those parts of the diagram which are unchanged if we make continuous changes to the diagram, without cutting the strands or changing the crossovers.

Now that we have a way of notating these knots, we have a search space; this consists of all sets of numbers which define knots according to the above procedures. In order to carry out a computational search technique we need to be able to do three things.

Initialize. We need to be able to pick an initial point or population of points (depending on the technique) from the search space. In the case of the knot search space we can do this in a number of ways: e.g. choosing a list of numbers at random and using these as the crossing-

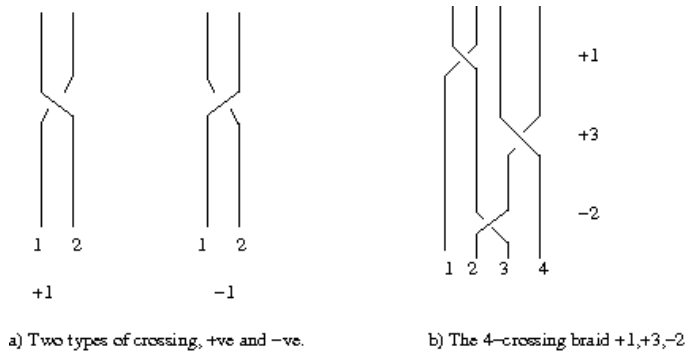

Figure 3: The braid notation.

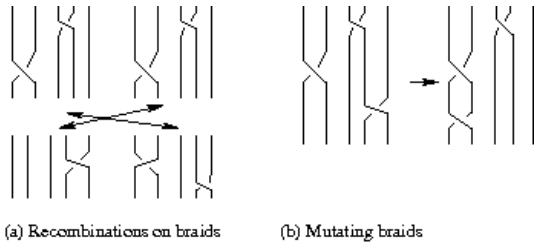

Figure 4: Genetic algorithm-style moves in knot-space

sequence of a braid, or doing a random walk on a two-dimensional grid then reading off the notation.

Move. We need to be able to do *moves* in the search space. For example in genetic algorithms we need to do *crossover* and *mutation* operators; in hillclimbing and tabu search we need to be able to do some local exploratory moves. There is a large choice of such moves for these kinds of problems (indeed choosing the best kinds of moves is an interesting challenge, beyond the scope of this paper). An example of such moves is given in figure 4.

Select We need some way of selecting the "best" example(s) after the move has been made. This will depend on the kind of problem that we are solving.

3 Counterexample Searching

The first kind of mathematical problem that we can tackle using these techniques is *searching for counterexamples*. This is where we are looking for a specific example of an object in our data-set which proves that some statement is false. For example the number 2 is a counterexample to the statement "all prime numbers are odd". Clearly to apply a search technique to counterexample finding, we need to be able to get more feedback from the system; it is unlikely that a problem like the one above would be solvable using search; knowing that, say, 3, 19, 29, ... are prime and odd doesn't give us any "direction" to find the number 2. Nonetheless we shall see below examples of other mathematical problems that do give more feedback to feed into a fitness function.

We have applied heuristic search to a counterexample finding-problem in knot theory. In particular we consider the question "when is a knot not a knot?"; i.e. when do we have a complicated tangle of strands which, nonetheless, can be untangled into an unknotted loop. This is not a simple question. One way to understand this kind of question is through the calculation of so-called *invariants* of diagrams. An invariant is an object, e.g. a number or a polynomial, which

can be calculated from the knot diagram, and which is the same regardless of which diagram is used for a particular 3-D knot. There are many such invariants. One yet-unresolved problem is to find an invariant which takes on a particular value when and only when the diagram is unknotted.

An early conjecture was that a particular invariant, called the *Alexander polynomial*, might play such a role. Eventually counterexamples were found: knots which had the same polynomial as the unknotted loop, but which were genuinely knotted. Can we use a computational search technique to rediscover this counterexample?

The fitness function for this problem needs to do two things. Firstly it needs to look for knots that have the same Alexander polynomial as the unknotted loop. However if we use that as our *only* fitness criterion, then the algorithm rapidly converges on just unknotted loops. Therefore we need to include a second fitness measure and do a multicriterion optimization. The important feature that this second measure must have is that it measures the complexity of the knot in some way, but measures that complexity in a *different* way to the Alexander invariant. To decide upon this second invariant we calculated the statistical correlation of a measure of the complexity of the polynomial (number and size of terms) against various other invariants. We then chose the *least* well correlated of these (this was a measure called the *hyperbolic volume of the knot complement*.

To summarize: to find a counterexample to the conjecture we are trying to find a knot with Alexander polynomial the same as an unknotted loop, yet which really is knotted. To do this we use a multicriterion fitness function to both *decrease* the complexity of the Alexander polynomial whilst *increasing* another measure of knot complexity: the hyperbolic volume.

The search strategy is a tabu search. We begin by randomly generating a knot with 60 crossings, and at each generation make six attempted moves, moving if possible to a solution which lowers the polynomial complexity whilst not reducing the hyperbolic volume. The algorithm was tested ten times: all runs found a counterexample between 2 and 9 rounds. Output from a typical run of the program is given in table 1.

4 Enumerating Equivalence Classes

A second type of problem is *enumerating the different types of objects*. An example of this from the knot search space is that of *tabulating* knots, that is finding one representative 2-D diagram for each possible 3-D knot. This activity dates back a long way [6], and was considered particularly important to nineteenth century mathematicians due to a popular theory [10] which suggested that atoms consisted of knots in the aether!

Again we can frame this as a search problem. In this case we are searching the set of all possible knot diagrams for diagrams which represent *different* knots to the previous stages in the search. In order to carry this out we use a GA with an adaptive fitness function which gives a positive rating to areas of the search space when there are still novel solutions being found, but applies penalties to population members which produce knots which are of the same type as those already discovered. This technique is discussed in general in [4, 5]. A comparison of this diversity-seeking GA with random search is given in figure 5.

5 Finding Sequences of Transformations

A final problem type, which is the subject of our current work, is finding sequences of transformations from one object to another. For example it is known that any two knot diagrams that represent the same underlying knot can be transformed between one another by a sequence of

```
initial fitness is Volume: 53.014083, polynomial complexity: 3528.0

About to make a move:
Fitness of move 0 is this. Volume: 31.267518, polynomial complexity: 744.0
Fitness of move 1 is this. Volume: 53.336667, polynomial complexity: 14404.0
Fitness of move 2 is this. Volume: 28.850185, polynomial complexity: 192.0
Fitness of move 3 is this. Volume: 55.256613, polynomial complexity: 21024.0
Fitness of move 4 is this. Volume: 50.544457, polynomial complexity: 3640.0
Fitness of move 5 is this. Volume: 56.492307, polynomial complexity: 33592.0
Best is 2
Lowest complexity is 192.0

About to make a move:
Fitness of move 0 is this. Volume: 52.273302, polynomial complexity: 13816.0
Fitness of move 1 is this. Volume: 49.313944, polynomial complexity: 10512.0
Fitness of move 2 is this. Volume: 56.550171, polynomial complexity: 11048.0
Fitness of move 3 is this. Volume: 51.509364, polynomial complexity: 12972.0
Fitness of move 4 is this. Volume: 34.726442, polynomial complexity: 516.0
Fitness of move 5 is this. Volume: 57.00492, polynomial complexity: 1444.0
No change
Lowest complexity is 192.0

About to make a move:
Fitness of move 0 is this. Volume: 51.924411, polynomial complexity: 9728.0
Fitness of move 1 is this. Volume: 34.008854, polynomial complexity: 40.0
Fitness of move 2 is this. Volume: 53.997257, polynomial complexity: 1484.0
Fitness of move 3 is this. Volume: 49.971012, polynomial complexity: 556.0
Fitness of move 4 is this. Volume: 56.816054, polynomial complexity: 12500.0
Fitness of move 5 is this. Volume: 44.194888, polynomial complexity: 392.0
Best is 1
Lowest complexity is 40.0

About to make a move:
Fitness of move 0 is this. Volume: 43.368439, polynomial complexity: 460.0
Fitness of move 1 is this. Volume: 36.163213, polynomial complexity: 1180.0
Fitness of move 2 is this. Volume: 27.572854, polynomial complexity: 340.0
Fitness of move 3 is this. Volume: 22.662824, polynomial complexity: 60.0
Fitness of move 4 is this. Volume: 32.233932, polynomial complexity: 160.0
Fitness of move 5 is this. Volume: 38.972726, polynomial complexity: 584.0
No change
Lowest complexity is 40.0

About to make a move:
Fitness of move 0 is this. Volume: 34.214721, polynomial complexity: 120.0
Fitness of move 1 is this. Volume: 32.755526, polynomial complexity: 36.0
Fitness of move 2 is this. Volume: 43.197662, polynomial complexity: 464.0
Fitness of move 3 is this. Volume: 41.407491, polynomial complexity: 308.0
Fitness of move 4 is this. Volume: 34.372221, polynomial complexity: 368.0
Fitness of move 5 is this. Volume: 17.551978, polynomial complexity: 16.0
Best is 5
Lowest complexity is 16.0

About to make a move:
Fitness of move 0 is this. Volume: 33.702301, polynomial complexity: 40.0
Fitness of move 1 is this. Volume: 30.393301, polynomial complexity: 184.0
Fitness of move 2 is this. Volume: 36.205037, polynomial complexity: 48.0
Fitness of move 3 is this. Volume: 34.666526, polynomial complexity: 28.0
Fitness of move 4 is this. Volume: 51.151599, polynomial complexity: 4924.0
Fitness of move 5 is this. Volume: 38.672732, polynomial complexity: 132.0
No change
Lowest complexity is 16.0

About to make a move:
Fitness of move 0 is this. Volume: 31.918501, polynomial complexity: 592.0
Fitness of move 1 is this. Volume: 21.882483, polynomial complexity: 28.0
Fitness of move 2 is this. Volume: 26.71534, polynomial complexity: 44.0
Fitness of move 3 is this. Volume: 4.555919, polynomial complexity: 12.0
Fitness of move 4 is this. Volume: 26.352331, polynomial complexity: 36.0
Fitness of move 5 is this. Volume: 9.256363, polynomial complexity: 4.0
Best is 5
Lowest complexity is 4.0

About to make a move:
Fitness of move 0 is this. Volume: 9.256363, polynomial complexity: 4.0
Fitness of move 1 is this. Volume: 20.632373, polynomial complexity: 4.0
Fitness of move 2 is this. Volume: 18.941126, polynomial complexity: 0.0
Fitness of move 3 is this. Volume: 30.629161, polynomial complexity: 104.0
Fitness of move 4 is this. Volume: 23.883052, polynomial complexity: 60.0
Fitness of move 5 is this. Volume: 24.841542, polynomial complexity: 52.0
Best is 2
Lowest complexity is 0.0
Solution found
```

Table 1: A run of the counterexample-search program.

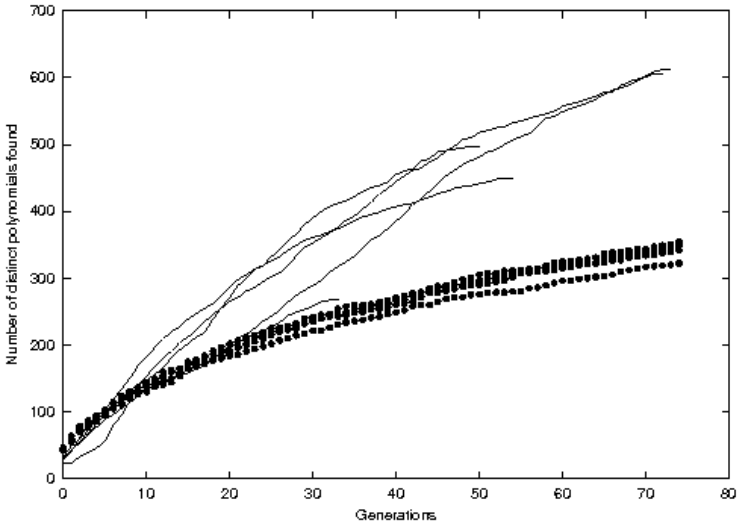

Figure 5: Number of distinct knots per generation: a comparison of the genetic algorithm and random generation for five runs (GA: line; random search: dots).

transformations drawn from a set known as *Reidermeister moves* [8]. However it is not easy to derive such a set of moves for a particular pair of knots. We are currently experimenting with heuristic search techniques for this problem.

6 Conclusions

This paper has introduced an approach to mathematics research which uses heuristic search as a way of exploring mathematical conjectures and studying mathematical objects. This is part of a general programme to study some aspects of mathematics in a data-driven fashion, and introduce techniques from heuristic search and data mining into mathematics research. In our future work we will continue to investigate problems of the three types above, both in the topological area and in other areas of mathematics such as graph theory.

References

[1] C. A. Adams. *The Knot Book*. W.H. Freeman, 1994.
[2] J. C. Baez, editor. *Knots and Quantum Gravity*. Oxford University Press, 1994.
[3] C. Dowker and M. B. Thistlethwaite. Classification of knot projections. *Topology and its Applications*, 16(1):19–31, 1983.
[4] C. G. Johnson. Finding qualitative examples with genetic algorithms. In R. John and R. Birkenhead, editors, *Developments in Soft Computing*, pages 92–99. Springer, 2001.
[5] C. G. Johnson. Understanding complex systems through examples: a framework for qualitative example-finding. *Systems Research and Information Systems*, 10:239–267, 2001.
[6] R. T. P. Kirkman. The enumeration, description, and construction of knots of fewer than ten crossings. *Transactions of the Royal Society of Edinburgh*, XXXII, 1884.
[7] K. Murasugi. *Knot Theory and its Applications*. Birkhäuser, 1996.
[8] K. Reidermeister. *Knotentheorie*. Springer, 1932.
[9] D. W. Sumners. Untangling DNA. *Mathematical Intelligencer*, 12(3):71–80, 1990.
[10] S. W. Thompson. On vortex motion. *Transactions of the Royal Society of Edinburgh*, XXV, 1867.
[11] E. Witten. Quantum-field theory and the Jones polynomial. *Communications in Mathematical Physics*, 121(3):351–399, 1989.