Solving the Quadratic Assignment Problem with BOA

(Bachelorproject)

Alex van der Bie, s1419137,
Supervisor: Marco Wiering
University of Groningen, Department of Artificial Intelligence

February 26, 2011

Abstract

Genetic Algorithms have been used throughout the years for a large number of optimization problems. However, it has become ever more clear that these algorithms are powerful but not always very efficient. The biggest problem is that GAs often take a long time to find a near-optimal solution. Martin Pelikan has proposed an optimization technique called the Bayesian Optimization Algorithm (BOA). BOA uses the structure of the best solutions to model the data in a Bayesian Network, using the building blocks of these solutions. Then new solutions can be extracted from the network and proposed for evaluation. While Pelikan used this algorithm to solve simple, mostly binary, problems, we have used the same techniques for the Quadratic Assignment Problem, an NP-Hard problem. Our findings are that with BOA we get better solutions, more so for greater problem sizes.

1 Introduction

Genetic Algorithms (GAs) [6, 9] show great promise as tools for solving optimization problems. The general strategy for a GA is to make a population of randomly generated proto-solutions and through a number of generations evolve them to the optimal solution. There are however a few problems with these algorithms. The one we wish to address here is that they can take a long time to find the optimal solution (if it finds the solution at all). The reason for this is that GAs use the crossover mechanic for making new proto-solutions. The idea here is similar as in nature: randomly combining the characteristics (genes) of two parent solutions to form a new solution. The problem is that the good part of these solutions often lies in combinations of certain genes, we call these combinations 'building blocks'. Random recombinations will often cause the structure of the building blocks to be copied only partly, so the newly generated solution is often worse than the parents are.
Many studies try to use methods to hold the building blocks together. There are a few statistical methods. An example is PBIL [2] that tries to calculate the probability for the occurrence of every specific gene and builds a solution from those probabilities. Another example is UMDA [11] that uses a function to calculate the fitness of potential offspring of a pair of solutions to get higher quality offspring. What we will be studying though is BOA [13] BOA is a method that uses Bayesian Networks [12] to build a model of the solutions. Pelikan has booked great results with using this optimization technique on problems containing strings of bits. We are going a step further and we will try this same method on the Quadratic Assignment Problem (QAP).

Outline: In section 2 the background knowledge will be explained, starting with Genetic Algorithms, going through Bayesian Networks, the BOA algorithm and finally the Quadratic Assignment Problem. Then, in section 3, we will describe the used method and the experimental setup. Section 4 will discuss the results and finally section 5

gives the conclusion and a discussion.

2 Background

2.1 Genetic Algorithms

The idea of Genetic Algorithms [5] is, as the name implies, derived from genetics. The next pseudo-code might illustrate its working:

Generate initial solutions
do until solution found
    Sort solutions by fitness
    Delete the worst solutions
    Generate new solutions from
    kept ones with crossover
    and mutation to
    replace the deleted solutions
end

In this way the typical 'survival of the fittest' idea is used to incrementally optimize a population of possible solutions. First a random population is created. With each iteration the worst solutions are replaced with newly generated solutions that are probably better than the ones that are replaced. Because of this the solutions should get better and better with each iteration and eventually reach an optimal solution.
The way this is most often done is with the methods of crossover and mutation. These methods work just like in normal genetics. Crossover means that two strands of DNA (from two parents) are combined into a new one. This means that some of the genes from two successful members of the species are copied and some combinations of these will be even more successful than one of the two individual parents. There are many methods to create a crossover operation [10]. A method that is most often used is to pick two crossover points and switch out everything between those points. Figure 1 might illustrate this operation.

This method is used often because it keeps the parts of the two solutions which might be the best for the newly created solution.
Mutation is the process in which some of the genes are copied ineffectually and are replaced by new,

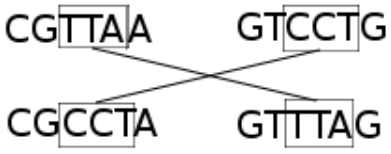

Figure 1: A crossover example

Figure 2: A mutation example

unrelated ones. Figure 2 illustrates this.

Mutation does not always have positive effects (as we can easily see in nature, where harmful mutations occur frequently), but mutations have to occur in order to assure that new information is continuously added to the population.
There are also a number of selection mechanisms that can be used in GAs [7]. A method that is often used is tournament selection. In this method a random number of individuals is selected from the population. The best individual from this group is then selected to produce offspring.
Another method is ranking selection. The idea here is simple: sort the population from best to worst and choose individuals to reproduce and individuals to be replaced based on their individual ranks. The offspring from some of the best individuals will replace some of the worst individuals and the population will gradually improve.

2.2 Bayesian Networks

A Bayesian Network [12] is a network of nodes connected by edges that represent a conditional probability. This can be used to build a probabilistic model of the data. The network has to be directed and acyclic.

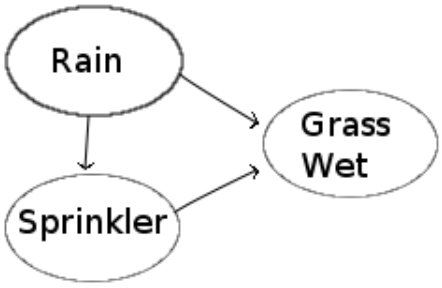

Figure 3: Example of a Bayesian Network

The above network in Figure 3 is a model of grass being wet. The rule is that the probability of any node is dependent upon its parent nodes. This means that the probability of the grass being wet is dependent upon the sprinklers working and/or any rain that might have fallen. The example is a very simple network, but it complicates a bit when you add the probabilities to the edges. The purpose of these kinds of networks is to infer information from the network with the following formula:

$$p(X) = \prod_{i=0}^{n-1} p(X_i|p(X_i)) \quad (1)$$

Where $p(X_i)$ denotes the parent variable of x_i. In short this formula tells us: (1) that the probability of any node is dependent only on the probability of the parent nodes and (2) the probability of all nodes having particular values can be computed with it. This technique is useful in inferring any type of information from a Bayesian Network.
The difficulty of Bayesian Networks is in learning the structure of the Network from the data. Unfortunately this problem is itself a complex problem which has not been completely solved [8]. However, we do not have to find the optimal network. We only have to find a network that is sufficient for modeling our data. There are however a number of known methods for obtaining a network from the data. We look at two, that have been proposed by Wu and Shapiro [16]. The first method is Greedy Search. In this method, for every pair of nodes it is calculated which edge addition would improve the network the most. The edge is added to the network and edges are added in this same way until no more additions can be found that improve the network or certain stop criteria are met. A second method is to just pick two nodes at random and check if the network would improve if a connection were to be added between these nodes. If this is the case, add the connection and keep adding in this way until the network does not improve for a set number of attempts or certain stop criteria are met.
Both these methods rely on obtaining a good score for the quality of the network. There are a number of good methods for obtaining a score but we will not go too far into that here. Since our goal is to obtain a network that is a good model for the population it is sufficient to add any edge that represents data that is also in the population. Therefore scoring the network might be as simple as just counting the occurrences of a connection in the population and comparing it to the score of the previous network.
This is just a small sample from all the possible methods for building a network from the data. An optimal method has not been found yet, but we can work with any method that leads to a sufficiently optimized network from the data.

2.3 The Bayesian Optimization Algorithm

The Bayesian Optimization Algorithm (BOA) is based on normal GAs. The only significant difference is that it uses a Bayesian Network instead of the techniques of crossover and mutation to obtain the next generation. The crossover method has the problem that building blocks are often ripped apart as a result of the operation, leaving the next generation with a genome that is often worse than either of its parents. Therefore we use a different method for obtaining next generations. This method is to build a model of the data using a Bayesian Network. This is done by trying to find common structures in the genome of the best solutions. As nodes for the network, we choose the specific value of a particular gene and connect it to other values for other genes to try and obtain a structure that works well. We now have a network in which the paths can be seen as possible solutions from the data. Now, following a path through this network will give us a possible solution that shares features with the best solutions in the population.

This is exactly what we were striving for in the first place. These new solutions can be inserted into the population as a new generation, building a new population for our evolutionary algorithm without ever having to use crossover operations.

Martin Pelikan used this method on linearly separable problems [14]. This seems unneccesarily restrictive though. BOA should be an important technique for any optimization problem. It would therefore be interesting to use BOA on harder problems. We chose the class of NP-Hard problems specifically to test the limits of BOA. This is an area that GAs have had some success with and we believe BOA would be a good optimization method for these problems.

2.4 The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is an optimization problem. The formal definition of the problem is as follows: There are a set of n facilities and a set of n locations. For each pair of locations, a distance is specified and for each pair of facilities a weight or flow is specified (e.g., the amount of supplies transported between the two facilities). The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows. Or, more generally: given a flow matrix W of size $n * n$ and a distance matrix D of size $n*n$, find an assignment f so that:

$$Cost(f) = \sum_{i,j \in P} w(i,j) * d(f(i), f(j))$$

is minimal.
The problem is NP-Hard [3], which means that there is no known algorithm that can solve this problem in polynomial time. In fact, even a standard approach like local search has proven to be problematic [15]. Genetic Algorithms have been applied for this problem, with reasonable amounts of success [1]. We use a variant of the solution that has been proposed by Wu and Ji [17] in which they propose a GA with a new replacement strategy, as explained in section 3.

This is a general form of more specific problems like scheduling or manufacturing of microchips.

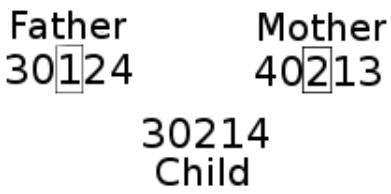

Figure 4: One crossover step

The principles of BOA and GAs however can be used in any problem, regardless of the specifics.

3 Methods & Setup

To solve the QAP with BOA we first need to have a fair comparison. To make sure this is the case we have two experimental setups. At first we will generate our own random problems. Each problem can be described with two $n * n$ matrices. We fill these up with random numbers and feed them into both algorithms. This way there will always be a fair comparison between the two algorithms.
Secondly there is QAPlib [4], an online library of Quadratic Assignment Problems with real-world problems as well as computationally hard problems.
We will compare two separate algorithms. The first one is a regular GA using a crossover operation as described by Ahuja et al [1]. The crossover operation needs to take into account that after the operation every number can appear only once. The procedure of Ahuja et al is to select two parent solutions from the population and pick a number of crossover points. For each of the crossover points, replace the value at that point with the value of that point in the other parent. Now all we have to do is search in the solution for the other instance of the value we just inserted in the solution and replace it with the value we deleted. Figure 4 illustrates how one of these steps takes place. This procedure is then repeated for each crossover point.

When using this crossover method mutation is unneccesary as each number can only appear once in each solution. This would usually be accomplished by flipping the values of two positions. As explained above, the crossover operation already does this, so the mutation operation becomes

Problem Size	GA	BOA
30	15561	15031
35	21827	21121
40	29850	28906
45	36445	35519
50	46215	45647
55	56317	55761

Table 1: Performance on random problems

superfluous.

Next we explain the BOA algorithm. BOA is similar to the GA but instead of using crossover we build a Bayesian Network from the best solutions in the population. When building the network, for each position a number could take, we make a node for that position. This way, the network has n^2 nodes by default. We initially build just the nodes and create all connections from there. Our method for making these connections is based on the stochastic search method of Wu and Shapiro [16]. The strategy is to pick two nodes at random and make a connection between these two nodes. After this we test the network and see if the addition has improved the network. If the network has not improved, remove the connection. Repeat this until the network does not change.
When the network is finished it is simply a question of following a path through the network to obtain a possible solution for the problem. The probability of any solution can be calculated by multiplying all the values of the connections between the nodes of that solution in the network. These solutions are fed into the population and the rest of the algorithm works similar as the GA.

4 Results

Table 1 shows our results for the experiment on the random problems. In these experiments we generated a random problem and presented it to both algorithms. A population size of $5n$ was found to be sufficient. Both algorithms are made to run for 30000 generations and offspring percentages are set to 50

Table 2 shows a run of a few of the QAPlib problems. It is clear that BOA outperforms the GA in all instances of the problems.

Problem	GA	BOA	Optimal
Bur26a	5446751	5426670	5426670
Nug25	4204	4120	3744
Rou12	249574	235528	235528

Table 2: Performance on QAPlib problems

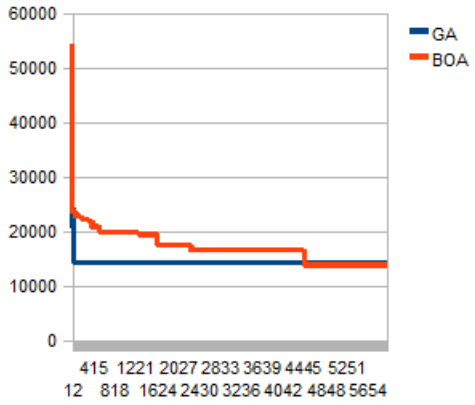

Figure 5: Learning curves for the algorithms

Finally Figure 5 shows the leaning curves for GA and BOA. It can be seen that the GA is intially a bit faster but quickly stops improving because the population gets stuck on a particular solution. BOA keeps on improving until it surpasses the GA.

These results look very promising. BOA often finds the optimal solution, where GA does not. We have also found in smaller experiments that for smaller problems, where both algorithms find the optimal solution, BOA is generally faster to find the optimum.

5 Conclusions

We believe that we have demonstrated that even in NP-Hard problems, an evolutionary algorithm can be optimized with the Bayesian Optimization Algorithm in such a way that it produces results more efficiently. In both the random problems and the QAPlib problems BOA outperformed the regular GA. Not always by a large margin, but in these kinds of optimization problems relatively small fit-

ness improvements like this are already worth a lot. If we can optimize combinatorial optimization algorithms to consistently give better results, this is already worth a lot of effort.
The biggest strike against BOA is that it takes longer to generate new solutions. GAs can generate a new solution in linear time while BOA takes n^3 to build the network and generate the solutions from it.

References

[1] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research 27*, pages 917–934, 2000.

[2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *Tech. Rep. No. CMU-CS-94-163*, pages 94–163, 1994.

[3] R.E. Burkard. Quadratic assignment problems. *European Journal of Operational Research, 15*, pages 283–289, 1984.

[4] R.E. Burkard, S.E. Karisch, and F. Rendl. http://www.opt.math.tu-graz.ac.at/qaplib/.

[5] Harik G. and F. Lobo. A parameter-less genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) I*, pages 258–265, 1999.

[6] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley, 1989.

[7] D. E. Goldberg and K. Deb. *A comparative Analysis of Selection Schemes Used in Genetic Algorithms.* University of Alabama, The Clearinghouse for Genetic Algorithms, Department of Engineering Mechanics, 1991.

[8] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning, 20*, pages 197–243, 1995.

[9] J.H. Holland. *Adaptation in natural and artificial systems.* Universitu of Michigan Press, Ann Arbor, MI, 1975.

[10] M. Mitchell. *An introduction to Genetic Algorithms.* First MIT Press, 1998.

[11] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. in parallel problem solving from nature (ppsn iv). In *PPSN IV*, pages 178–187. Springer Verlag, Berlin, 1996.

[12] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence 29*, pages 241–288, 1986.

[13] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. Boa: The bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), vol.I*, pages 525–532, 1999.

[14] M. Pelikan, K. Sastry, and D.E. Goldberg. Scalability of the bayesian optimization algorithm. *International Journal of Approximate Reasoning 31*, pages 221–258, 2002.

[15] T. Stutzle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research, 174 (3)*, pages 1519–1539, 2006.

[16] H. Wu and J. L. Shapiro. Choosing search algorithms in bayesian optimization algorithm. *World Academy of Science, Engineering and Technology 7*, pages 51–55, 2005.

[17] Y. Wu and P. Ji. Solving the quadratic assignment problems by a genetic algorithm with a new replacement strategy. *World Academy of Science, Engineering and Technology 30*, pages 310–314, 2007.