Journal of Statistics and Computer Science
Vol. 1, Issue 2, 2022, pp. 111-118
https://doi.org/10.47509/JSCS.2022.v01i02.01

Formal Analysis of Penalty Functions in Constrained Optimization with Genetic Algorithms

Wim Hordijk
SmartAnalytiX, Vienna, Austria
wim@WorldWideWanderings.net

Abstract

It is well known that using a penalty function can significantly improve the search performance of a genetic algorithm (GA) on constrained optimization problems. However, although it is intuitively clear, there is very little theory for why and how this actually works. Here, a formal investigation into this phenomenon is presented in terms of a schema analysis and a Walsh function analysis. This explicitly elucidates two particular mechanisms by which a penalty function can improve GA search performance.

Keywords: Machine learning; Schema analysis; Walsh function analysis

1 Introduction

Genetic algorithms (Holland, 1975; Goldberg, 1989b; Mitchell, 1996) are a well-known machine learning technique for finding good approximate solutions to difficult optimization problems. When dealing with constrained optimization, the use of a penalty function was already suggested early on (Goldberg, 1989b; Davis, 1991). Subsequently, various guidelines have been provided for how to use penalty functions in a genetic algorithm (GA) in an efficient way (Richardson et al., 1989; Deb, 2000), and experiments have shown that this indeed improves the search performance (Siedlecki and Sklansky, 1989; Powell and Skolnick, 1993; Smith and Tate, 1993). Other methods for handling constraints have also been proposed. However, they are usually more complicated, requiring special genetic operators that preserve feasibility (Michalewicz and Janikow, 1991), or using several (computationally expensive) steps to identify the feasible region in the search space (Schoenauer and Xanthakis, 1993). A comparison of several such methods was presented by Michalewicz (1995).

A penalty function thus provides a simple and cost effective method that, when carefully applied, significantly improves the GA search performance. However, although it is intuitively clear why and how this works, there is very little mathematical theory underlying this. Here, a GA is applied to a simple instance of the knapsack problem, both with and without a penalty function. The differences in search performance between the two methods are then formally analyzed in terms of schemata and Walsh functions. This formal analysis explicitly elucidates two possible mechanisms by which a penalty function can help improve GA search performance.

This paper is organized as follows. First, section 2 describes the methodology used, in particular the instance of the knapsack problem, the penalty function, and the GA parameters. Next, section 3 presents the formal analyses in terms of a schema analysis and a Walsh function analysis, respectively, to explain the differences in search performance. Finally, section 4 summarizes the main conclusions, and some directions for further research are suggested.

2 Methodology

2.1 The knapsack problem

The knapsack problem is a well-known and easy to define constrained optimization problem (Moret and Shapiro, 1991). Imagine a number N of objects, each with a particular value v_i and weight w_i, $i = 1, \ldots, N$.

Article History
Received : 25 September 2022; Revised : 08 November 2022; Accepted : 08 November 2022; Published : 15 December 2022

To cite this paper

Wim Hordijk (2022). Formal Analysis of Penalty Functions in Constrained Optimization with Genetic Algorithms. *Journal of Statistics and Computer Science.* 1(2), 111-118.

The goal is to put a subset of these objects in a knapsack, such that the sum of their values is maximized, but with the sum of their weights not exceeding a given maximum weight M. Formally, the problem is stated as follows:

$$\text{maximize} \sum_{i=1}^{N} v_i x_i \quad \text{subject to} \sum_{i=1}^{N} w_i x_i \leq M$$

where $x_i \in \{0, 1\}$. Solutions that obey the maximum weight constraint are called *feasible*, and solutions that violate this constraint are called *infeasible*.

Here, an instance of the knapsack problem with $N = 10$ objects is considered, where the values and weights are as shown in Table 1. The values are chosen at random, and the weights are the same as the values, but at random positions two weights are swapped. The sum of the values (and thus also of the weights) of all 10 objects is equal to 97. The maximum allowed weight M is set to 80.

Table 1: The values and weights in the used instance of the knapsack problem.

Object	1	2	3	4	5	6	7	8	9	10
Value	10	13	11	8	2	12	19	5	7	10
Weight	8	5	19	10	2	12	11	13	10	7

For this knapsack instance it is easy to find the optimal solution, which turns out to be all objects included except for the third one, yielding a total value of 86 and a total weight of 78. In that sense there is no reason to apply a GA, but this instance is small enough to allow for a complete formal analysis. In the GA, the genotypes are bit strings of length N, where the i^{th} bit indicates whether object i is included (1) or not (0). The fitness of a bit string is then simply the sum of the values of all the included objects (i.e., all the objects that have a 1 at their corresponding bit positions).

However, because of the maximum weight constraint, the GA could create infeasible solutions. One way of dealing with this, is to just discard infeasible solutions and give them a fitness of zero. Another way is to use a penalty function, and "punish" infeasible solutions by decreasing the fitness value they would have when the constraint is ignored.

2.2 The penalty function

The most common and straightforward way of incorporating a penalty for infeasible solutions is to let the penalty depend on how many constraints are violated, and how badly they are violated (Richardson et al., 1989). It has been shown that this method will indeed improve GA search performance on constrained optimization problems (Siedlecki and Sklansky, 1989; Powell and Skolnick, 1993; Smith and Tate, 1993).

Since the knapsack problem has only one constraint, the penalty function used here is very simple. For each infeasible solution in the GA population, the unconstrained fitness will be decreased by a fixed fraction, regardless of how much the weight restriction is violated. In other words, each infeasible solution will be assigned a fitness value of $c \cdot f$, where c is a fixed value between 0 and 1, and f is the fitness value of the solution without any constraints (i.e., the sum of the values of all objects with a 1 in their corresponding bit position). A feasible solution will just be assigned its unpenalized fitness value.

The parameter c can be tuned, which will influence the search performance of the GA. Care should be taken, however, that the value of this parameter is not set too high, otherwise an infeasible solution might end up with a higher fitness than the optimal (feasible) solution. With this in mind, the value of c is (otherwise arbitrarily) set to 0.75.

2.3 GA search performance

Two variants of a GA were applied to the given instance of the knapsack problem. In the first variant, the fitness of infeasible solutions is simply set to 0. In the second variant, the penalty function described in the previous section is used. Otherwise the implementation of both variants is exactly the same: a population size of 50, roulette wheel selection, one-point crossover with a probability of 0.75, and a per-bit mutation

rate of 0.01. With each GA variant, 100 runs were performed for 100 generations each. (Note that a fairly low mutation rate is used, as the analyses below focus primarily on crossover mechanisms.)

The number of runs (out of 100) where the optimal solution was found is 70 for the variant that discards infeasible solutions (i.e., gives them a fitness of 0), while it is 95 for the variant with the penalty function. Clearly, the penalty function significantly improves the search performance of the GA. Increasing the number of generations in a run does not make a difference, as the optimal solution is typically found in about 20 generations, or not at all.

As a reminder, the optimal solution for the particular instance used here is 1101111111, i.e., all objects except the third one included, with a total value of 86. It turns out that for those runs where the GA did not find the optimal solution, it converged to a sub-optimal solution 1111111001, i.e., all objects except the eighth and ninth included, with a total value of 85, just one less than the optimal value. The question now is: How did the penalty function help the GA to find the optimum significantly more often instead of converging to the sub-optimal solution?

3 Results and Discussion

3.1 Schema analysis

One way of trying to answer this question is to look at the schemata (Holland, 1975) that characterize the optimal and sub-optimal solutions. Since these two solutions differ in the third, eighth and ninth bit, the value of the other seven bits do not matter for this comparison. This gives rise to the following two schemata:

optimal solution schema: $**0****11*$
sub-optimal solution schema: $**1****00*$

Since there are only $2^{10} = 1024$ possible bit strings of length $N = 10$, it is easy to calculate the fitness values of these schemata, which is the average of the fitness values of all their respective instances. Note that neither schema contains any infeasible instances, so these average fitness values, presented in Table 2 are the same for both GA variants.

Table 2: The fitness values of the relevant schemata.

schema	optimal $**0****11*$	sub-optimal $**1****00*$
fitness	49.0	48.0
st.dev.	15.4	15.4

The fitness of the sub-optimal schema is just one less than the fitness of the optimal schema, both having a rather high variance. Therefore, because of sampling biases caused by a finite population size, sometimes the GA will converge to the sub-optimal schema. Once the population is dominated by the sub-optimal schema, crossover is not able to produce the optimal solution anymore, and with a low mutation rate the probability of mutating the three relevant bits all at once is too low.

So, it might be useful to have an "intermediate" schema that can compete with the sub-optimal schema, and help the GA to find the optimal solution via a crossover between instances of the sub-optimal schema and the intermediate schema.

Two possible candidates for such an intermediate schema are $**0****00*$ and $**1****11*$. Calculating their average fitness values yields a value of 37.0 and 46.6 respectively. Note, however, that the second intermediate schema does have infeasible instances, and the average fitness of 46.6 is for the first GA variant, where infeasible solutions have a fitness of 0. It seems that only the intermediate schema $**1****11*$ would be able to compete with the sub-optimal schema, since its fitness is close to that of the sub-optimal one.

Figure 1 shows a particular run of the first GA variant (i.e., without the penalty function), where it eventually converges to the sub-optimal schema. The plot shows the percentages of individuals in the

population that are an instance of any of the three relevant schemata (optimal $**0****11*$, sub-optimal $*1****00*$, and intermediate $*1****11*$). The optimal and intermediate schemata die out rather quickly, and only the sub-optimal schema eventually survives. Clearly, the intermediate schema is not able to compete sufficiently with the sub-optimal schema, and there is no opportunity for a crossover between instances of the intermediate and sub-optimal schemata.

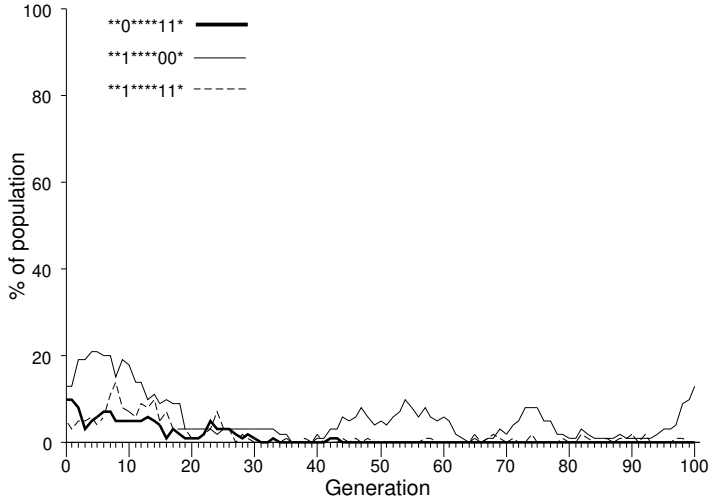

Figure 1: A particular GA run *without* the penalty function.

However, using the penalty function will increase the fitness of this intermediate schema, as it increases the fitness values of its infeasible instances from zero to some positive number. Recalculating the fitness value of the intermediate schema, but with the penalty function, gives a value of 56.7, i.e., higher than that of both the optimal and sub-optimal schema. This should certainly make the intermediate schema capable of competing with the sub-optimal one.

Figure 2 shows a particular run of the second GA variant (i.e., with the penalty function), where the GA initially converges to the sub-optimal schema, with the optimal schema quickly dying out. However, the intermediate schema is now able to survive, and after almost 50 generations the optimal schema suddenly re-appears, and eventually dominates the population. Clearly the penalty function, as opposed to just discarding infeasible solutions, helps the GA to find the optimal solution by making the intermediate schema competitive enough, thus providing an opportunity to have a crossover event between instances of the sub-optimal and the intermediate schemata.

This provides one possible and explicit explanation for the intuitive understanding of why penalty functions improve GA search performance on constrained optimization problems. In this particular case, the penalty function makes an intermediate schema competitive enough to stay alive, even if the population becomes dominated by a schema that belongs to a sub-optimal solution. This enables the GA to still find the optimal solution, even after initially converging to a sub-optimal one.

3.2 Walsh function analysis

Walsh functions (Walsh, 1923) can be used to analyze the difference in search performance of the GA with and without the penalty function in even more detail. In particular, the method presented by Goldberg (1989a) and Forrest and Mitchell (1993) is used here.

A *Walsh function* ψ_j maps a bit string x, $x \in \{0,1\}^l$, to $\{-1,+1\}$:

$$\psi_j(x) = \begin{cases} +1 & \text{if } x \wedge j \text{ has even parity} \\ -1 & \text{otherwise} \end{cases}$$

where $\wedge$ denotes bitwise AND, and the parity of a bit string is the number of 1s in the string. The index j, represented as a bit string of length l, denotes a partition of the space of all possible bit strings of length l.

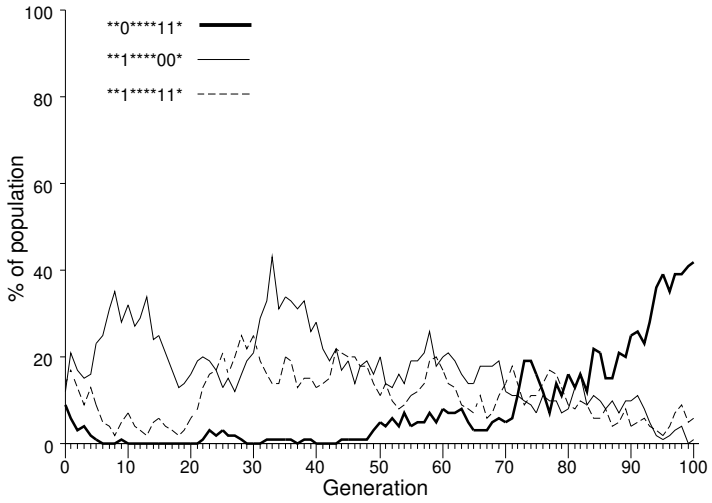

Figure 2: A particular GA run *with* the penalty function.

The order of a partition j is the number of "defined" bits in the partition, i.e., the number of 1s in the bit string j.

Analogous to a Fourier transform, each function f defined on $\{0,1\}^l$ can now be written as a linear combination of Walsh functions:

$$f(x) = \sum_{j=0}^{2^l-1} \omega_j \psi_j(x)$$

where the ω_j are real valued *Walsh coefficients*, which can be calculated by:

$$\omega_j = \frac{1}{2^l} \sum_{x=0}^{2^l-1} f(x)\psi_j(x)$$

The function $f(x)$ can be calculated as a sum of Walsh coefficients, where each term contributes to a better approximation of the exact value $f(x)$.

These Walsh coefficients can also be used to calculate or approximate the fitness of schemata. Since a schema itself is not a bit string, first a function is defined that transforms a schema h to a bit string:

$$\beta(h_i) = \begin{cases} 0 & \text{if } h_i \in \{0, *\} \\ 1 & \text{if } h_i = 1 \end{cases}$$

The fitness of schema h is now calculated as:

$$f(h) = \sum_{j \in J(h)} \omega_j \psi_j(\beta(h))$$

Here, $J(h)$ is the set of all indices of the partitions of which schema h is a subset. As a simple example, taken from Forrest and Mitchell (1993), the fitness of the schema $*11$ can be calculated as:

$$f(*11) = \omega_{000} - \omega_{001} - \omega_{010} + \omega_{011}$$

An exact calculation of $f(h)$ can be obtained by letting j run over all elements of $J(h)$. A *first order* approximation is obtained by only considering partitions of order 1 or smaller. An i^{th} order approximation is obtained by considering all partitions of order i or smaller.

Forrest and Mitchell (1993) give the following interpretation of how a GA works in terms of schemata and Walsh coefficients:

The GA's estimate of a given schema s can be thought of as a process of gradual refinement, where the algorithm initially bases its estimate on information about the low-order schemata containing s, and gradually refines this estimate from information about higher and higher order schemata containing s. Likewise, the terms in the sum above represent increasing refinements to the estimate of how good the schema $*11$ is. The term ω_{000} gives the population average (corresponding to the average fitness of the schema $***$), and the increasingly higher-order ω_j's in the sum represent higher-order refinements of the estimate of $*11$'s fitness, where the refinements are obtained by summing ω_j's corresponding to higher and higher order partitions j containing $*11$.

Thus, such a Walsh coefficients analysis can be applied to the optimal and sub-optimal schemata as defined in the previous section. The relevant Walsh coefficients for calculating the fitness of the optimal and sub-optimal schemata can be obtained as a function of the parameter c in the penalty function. Using these coefficients, a first order estimate f^1 of the fitness of these schemata is given by:

$$\begin{aligned} f^1(**0****11*) &= \omega_{0000000000} + \omega_{0010000000} - \omega_{0000000100} - \omega_{0000000010} \\ &= 45.66 + 3.34c \\ f^1(**1****00*) &= \omega_{0000000000} - \omega_{0010000000} + \omega_{0000000100} + \omega_{0000000010} \\ &= 47.15 + 0.87c \end{aligned}$$

It is clear that without the penalty function, i.e., when $c = 0$, the first order estimate of the fitness of the sub-optimal schema (47.15) is higher than that of the optimal schema (45.66). So it is not surprising that the GA tends to converge to the sub-optimal solution so often, since its schema will on average get a higher fitness estimate early on in the search. Only for $c = 0.60$ are both first order estimates equal, and for higher values of c, the first order estimate of the optimal schema is higher than that of the sub-optimal schema.

A second order estimate f^2 of the fitness of both schemata is given by:

$$\begin{aligned} f^2(**0****11*) &= f^1(**0****11*) - \omega_{0010000100} - \omega_{0010000010} + \omega_{0000000110} \\ &= 47.76 + 1.24c \\ f^2(**1****00*) &= f^1(**1****00*) - \omega_{0010000100} - \omega_{0010000010} + \omega_{0000000110} \\ &= 49.24 - 1.24c \end{aligned}$$

Again, for $c = 0$ the estimate of the fitness of the sub-optimal schema (49.24) is higher than that of the optimal schema (47.76), and they become equal for $c = 0.60$. So even the second order estimate will on average prefer the sub-optimal schema. A third order estimate, which is in this case also an exact one, gives the actual fitness values of the two schemata as given in the previous section.

This provides another explicit and formal explanation for how a penalty function can help the GA to find the optimal solution more often. Since the GA initially bases its estimates of schema fitnesses on lower-order estimates, it will on average prefer the sub-optimal schema when no penalty function is used ($c = 0$). However, using a penalty function will change these estimates, making the (lower-order) estimates of the fitness of the optimal schema higher than those of the sub-optimal schema.

4 Conclusions

Experiments have shown that when a GA is applied to a constrained optimization problem, the use of a penalty function can significantly improve its search performance. However, although it is intuitively clear, surprisingly little theory is available for why or how this actually works. In this paper, two explicit and formal explanations are provided for how a penalty function can help in improving the GA search performance.

The first explanation shows that the use of a penalty function can increase the fitness of an intermediate schema that is necessary to be able to find the optimal solution after the population has converged to a sub-optimal one. Without the use of a penalty function, i.e., just discarding infeasible solutions, this intermediate schema would not be competitive enough to survive.

The second explanation shows that the use of a penalty function can increase the lower-order estimates of the fitness of a schema of which the optimal solution is an instance, compared to that of a sub-optimal schema. Since the GA initially bases its estimates of these schema fitnesses on lower-order estimates, this can prevent the GA from convergence to a sub-optimal solution.

It is expected that these formal explanations generalize to a wider range of problems, and thus contribute to a real theory of how and why penalty functions improve GA search performance in constrained optimization. Suggested further research includes investigating how often the two mechanisms occur over a large number of GA runs (and other problems), and whether they can also work together, or always strictly independently.

References

Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338.

Forrest, S. and Mitchell, M. (1993). What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. *Machine Learning*, 13:285–319.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, 3(2).

Goldberg, D. E. (1989b). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization, and contraints. In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 151–158. Morgan Kaufmann.

Michalewicz, Z. and Janikow, C. Z. (1991). Handling constraints in genetic algorithms. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Moret, B. M. E. and Shapiro, H. D. (1991). *Algorithms from P to NP*, volume 1. Benjamin Cummings.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann.

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann.

Schoenauer, M. and Xanthakis, S. (1993). Constrained GA optimization. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580. Morgan Kaufmann.

Siedlecki, W. and Sklansky, J. (1989). Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 141–150. Morgan Kaufmann.

Smith, A. E. and Tate, D. M. (1993). Genetic optimization using a penalty function. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 499–505. Morgan Kaufmann.

Walsh, J. L. (1923). A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45(1):5–24.