Towards Automatic Domain Knowledge Extraction for Evolutionary Heuristics*

Márk Jelasity

Research Group on Artificial Intelligence,
University of Szeged, Hungary
`jelasity@inf.u-szeged.hu`
and
Leiden Institute of Advanced Computer Science,
Leiden University, The Netherlands

Abstract

Domain knowledge is essential for successful problem solving and optimization. This paper introduces a framework in which a form of automatic domain knowledge extraction can be implemented using concepts from the field of machine learning. The result is an encoding of the type used in most evolutionary computation (EC) algorithms. The approach focuses on whole problem domains instead of single problems. After the theoretical validation of the algorithm the main idea is given impetus by showing that on different subdomains of linear functions the method finds different encodings which result in different problem complexities.

1 Introduction and Motivation

Domain knowledge plays a key part in today's machine learning applications. Though in many cases relatively simple heuristics combined with the brute force of available fast processors and millions of test samples seems to be the best available solution — like hidden Markov models used in speech recognition systems rather than an expert knowledge of phonemes [11], or simple Bayesian models employed in natural language processing applications instead of knowledge of grammar [6] — it is now generally accepted that for instance the performance of evolutionary heuristics depends heavily on the applied encoding and operators. In fact this is consistent with what the "no free lunch" theorems [12] tell us: there are no algorithms that are the best in each domain, so for the best performance in each domain one has to find the best algorithm for each separately. Hence the extraction of domain knowledge is essential.

Our previous research into the question also supports this view [4, 2, 3]. We have shown that the usual practice of simply characterizing domains by giving them labels

such as "NP-hard" or "subset-sum problem" is not necessarily useful or even misleading. The actual structure and complexity of a domain (i.e. a set of functions defined over a common search space) depends on the source of these functions. For example a domain containing subset sum problems (i.e. "NP-complete" combinatorial optimization problems) may turn out to be a trivial domain due to the *structure of the parameters* of the particular functions in the domain. So even when a characterization is available (the function is not a black-box) the extraction of domain knowledge is still essential. In Section 4 it will be shown that even when we know that a domain contains only linear functions the performance can still be significantly improved using domain analysis.

The problem is that extracting domain knowledge in general is quite a difficult problem; scientific researchers and engineers do this for a living and it is not one of the easiest jobs available. However, in systems where knowledge is explicitly and separately represented, it is possible to perform some kind of meta optimization over the domain of possible knowledge content. Evolutionary heuristics are good examples since knowledge is expressed in the encoding and operators while the basic algorithm remains the same. A lot of methods can be found in the literature that tackle the problem of dynamic problem structure analysis. A survey of methods using probabilistic models can be found in [8]. Other approaches concentrate on linkage detection [7]. A common feature of these methods is that they concentrate on single functions. Our goal here is different in that we would like to extract knowledge that characterizes a whole domain and is reusable and interchangeable. One area of research is relevant from this point of view, namely the work of Radcliffe [10]. Their basic ideas on the nature of knowledge to be extracted are not unlike those presented here (the differences being emphasized later on) but they did not tackle the problem of extracting knowledge automatically in a systematic way.

This paper introduces a framework in which automatic domain knowledge extraction is possible. In our case domain knowledge means the representation or encoding of the search space. Here, binary representations will be generated that are optimal in a sense to be described later. A binary representation is a mapping of the search space to a set of the binary strings $\{0,1\}^n$. Though it is now widely accepted that an *arbitrary* binary representation is not necessarily better than an *arbitrary* non-binary encoding, our problem is a little different here. We are looking for the *optimal* binary representation in the space of all binary representations of a search space. Note that e.g. a search space of size 2^n has $2^n!$ different n-bit binary representations which is an enormous number. It is still possible that the *optimal* binary representation is not optimal in the space of all representations but here we do not tackle this problem.

In a binary representation every position of this string contains a 0 or a 1 which means that every position defines a *concept* over the search space. The term concept is used as in machine learning, i.e. a concept over a space is a subset of the space. Very briefly, our method is based on finding such concepts with the help of a measure over the concept space.

The outline of the paper is as follows. In Section 2 the basic concepts of the framework are defined. In Section 3 a useful property of the method is proved which supports applicability. Section 4 provides an illustrative but interesting example of the possible advantages of the approach on the class of linear functions. In Section 5 it will be shown that this method is in fact a generalization of some probabilistic methods used to model the distribution of good solutions of a function (see [8]). Finally, Section 6 discusses the possibilities and limitations of implementation of the framework in real-world, large-scale problems.

2 Framework

It is important to emphasize that this section will define only a general framework which may have many implementations depending on problem size, available time and data, etc. These details will be discussed in the following sections.

2.1 Basic Terms

Let us first define a problem. A problem is given by its *search space*, and a real valued *objective function* defined over it. The notation of the search space is S. The notation of the function is $f : S \longrightarrow \mathbb{R}$. In other words $f \in \mathbb{R}^S$. In evolutionary computation the objective function is usually called the *fitness function*. In this paper I will adopt this convention.

The *problem domain* is a subset of all possible fitness functions. The problem domain will be denoted by $\mathcal{D} = \{f_1, f_2, \ldots\} \subseteq \mathbb{R}^S$. This notion is crucial from our point of view. In practice the problem domain is given by the problem situation, e.g. a university which needs schedules for organizing its activity. The particular variables of the particular university — i.e. the number of employers, students, rooms, the sizes of rooms, the habits of each lecturer (who get up early/late, work at home/in their office etc.) and so on — will make the scheduling task special. The fitness functions in the domain will probably have a lot of features in common. At the same time, the scheduling task is an NP-complete combinatorial optimization problem in general. But this mathematical definition includes many more functions which are very diverse compared to the ones actually encountered at our university. To handle problem domains as an actual sample of functions and trying to describe them instead of using a given definition is therefore a main constituent of the philosophy of the present approach (see also [4, 2, 3]).

A *concept* over S is a subset of S. The notation will be $C \subseteq S$ while $\overline{C} = S \setminus C$. In other words, using a function notation $C \in \{0, 1\}^S$. An *encoding of S* is given by an ordered list of concepts. The encoding will be denoted by $\mathcal{C} = (C_1, \ldots, C_n)$. Using this notation, the *code* of a solution $x \in S$ is given by $\mathcal{C}(x) = (C_1(x), \ldots, C_n(x)) \in \{0, 1\}^n$. For the sake of simplicity these binary codes will be used throughout the paper noting that generalization is possible to other kinds of codes.

Next let us define two properties of encodings. The first is very important from a practical point of view: an encoding has to be *invertible*, i.e. given a code c of a solution, we should be able to effectively compute solutions $x \in S$ for which $\mathcal{C}(x) = c$. Note that it is possible that x is not unique. The second is related to the efficiency of the encoding. We want as few concepts as possible. To express this we call an encoding *independent* if its concepts are stochastically completely independent, i.e.

$$\forall k, i_1, \ldots, i_k \quad Pr(x \in C_{i_1}, \ldots, x \in C_{i_k}) = Pr(x \in C_{i_1}) \ldots Pr(x \in C_{i_k})$$

This seemingly contradicts other results from the GA literature, which report that non-coding segments (introns) may improve the search (e.g. [5]). This may be true under the assumption that the encoding is not optimal and so the genetic drift introduced by the small population-size has a larger impact. With optimal encodings which will be defined later on the genetic drift is a smaller problem while with few concepts the search space size reduces significantly.

2.2 Automatic Generation of Codes

The task is to generate the optimal encoding for a problem domain. If we define the notion of optimality then the problem reduces to a search problem over the possible encodings.

First let us define the optimality of a concept over a domain. We need a concept that separates good and bad solutions as clearly as possible in all of the functions in the domain. Good (or bad) solutions are defined as being in the upper half (or the lower half) of the search space with respect to a given fitness function. Let us denote the concept representing exactly the good solutions for the fitness function f by G_f. Clearly every fitness function will define a different notion of good and bad solutions. For measuring the separation of good and bad solutions over a given fitness function *information gain* is an ideal choice. Information gain is a measure of "goodness" of cutting a space. Before cutting the space, the entropy of the space tells us how many bits are needed on average to encode if a random solution is good or bad. In the worst case one bit is needed (if the number of good solutions equals the number of bad ones) and in the best case no information is needed (0 bits) if all the solutions are good or bad. After cutting the entropy of the two resulting subspaces can be calculated. If the cut is good, these entropies will be smaller than the original entropy of the whole space. The difference of the average of the entropies of the two half spaces and the original entropy is the gain.

We use information gain as defined in the classical ID3 algorithm [9]. For a given fitness function f from the domain the information gain of a concept C is defined as follows:

$$\text{gain}(G_f, C) = E(\frac{|G_f|}{S}) - \frac{|C|}{|S|}E(\frac{|G_f \cap C|}{|C|}) - \frac{|\overline{C}|}{|S|}E(\frac{|G_f \cap \overline{C}|}{|\overline{C}|})$$

where function E is the entropy defined by $E(p) = -p\ln p - (1-p)\ln(1-p)$. Here p is the proportion of a given concept over the space under consideration. The natural logarithm was chosen because natural logarithm is equivalent to $\log_2$ as a measure of information according to information theory but our formulas will become simpler using $\ln$. $E(0) = E(1) = 0$ while $E(0.5)$ is maximal. This means that the information gain is maximal if $C = G_f$ or $\overline{C} = G_f$, and minimal (0) if C and G_f are independent. The measure we are seeking will be the average information gain of the concept over the functions in the domain. This measure is denoted by $\text{gain}(C)$. This means that a concept is an *optimal concept of the domain* $\mathcal{D}$ if it maximizes the average information gain over $\mathcal{D}$.

Since a useful encoding contains several concepts we need a method for finding additional concepts while preserving the mutual independence between the concepts. A good heuristic for doing this is to find the concepts iteratively, one by one, and then applying the definition of optimality to the subdomains defined by the inverse sets of the possible codes determined so far. For example two concepts define four possible codes. The inverse sets of these codes yield a classification of the search space defining four subsets. These subsets define four subdomains by restricting the functions of the original domain. An optimal concept can be found in each of these subdomains. Now let the third concept of the encoding be the union of these four optimal concepts. The rationale behind this heuristic is that this recursive construction ensures independence if the optimal concepts divide the search space in two equal parts. Of course this will not be true in general.

$$C_1 = \arg\max \operatorname{gain}(C)$$
$$g = \operatorname{gain}(C_1)$$
$$i = 1$$
$$\text{while}(g > \text{random gain})$$
$$\quad C_i = \arg\max_C \operatorname{gain}(C \mid C_1, \ldots, C_{i-1})$$
$$\quad g = \operatorname{gain}(C_i \mid C_1, \ldots, C_{i-1})$$
$$\quad i \longleftarrow i + 1$$

Figure 1: The algorithm for finding the optimal encoding.

This method provides us with a definition of the information gain of the concept C_{i+1} given that $C_1, \ldots, C_i$ are known (denoted by $\operatorname{gain}(C_{i+1} \mid C_1, \ldots, C_i)$). For this let us take the information gain values of C_{i+1} restricted to each of the subspaces defined by the known concepts as described above and let the information gain of C_{i+1} be the weighted average of these gains where the weights are proportional to the sizes of the corresponding subspaces. The algorithm for finding the optimal encoding is given in Figure 1. The algorithm stops when the gain of the new concept is not greater than the optimal information gain over a domain containing only random functions.

3   Theoretical Foundations

In this section I will show that the algorithm described in Section 2 is optimal in an important sense: random domains are never divided by any concept if some assumptions hold. This means that the subdomains of the original domain defined by the inverses of the codes are either random or empty. We say that a subdomain is random if it contains only random functions i.e. the values of the functions are drawn from the same distribution. Besides this the random subdomains are maximal i.e. every larger domain becomes non-random. In other words it is impossible to gain more information from the space by refining the encoding and the information contained in the encoding cannot be expressed using fewer random classes.

According to our algorithm we have to find the optimal concept on a domain, the concept with the maximal information gain. Let as assume that our domain with search space S contains a random subdomain S_2 (see Figure 2 for an illustration). I will show that for every concept that splits this random space there exists another concept which has a larger gain and which does not split S_2.

Now let us choose an arbitrary concept C. The dotted line in Figure 2 is the boundary of the concept. The subspace $S_1 = C \setminus S_2$ is the non-random part of C and $S_3 = \overline{C} \setminus S_2$ is the non-random part of $\overline{C}$. The sizes of the classes are $|S_i| = N_i$, $i = 1, 2, 3$. π_1 is the value for which

$$E(\pi_1) = \frac{1}{|\mathcal{D}|} \sum_{f \in \mathcal{D}} E\left(\frac{|G_f \cap C|}{|C|}\right)$$

There are two such values since $E(p) = E(1 - p)$. Let π_1 be the smaller one. With a similar definition of π_3 the gain of C now can be written as

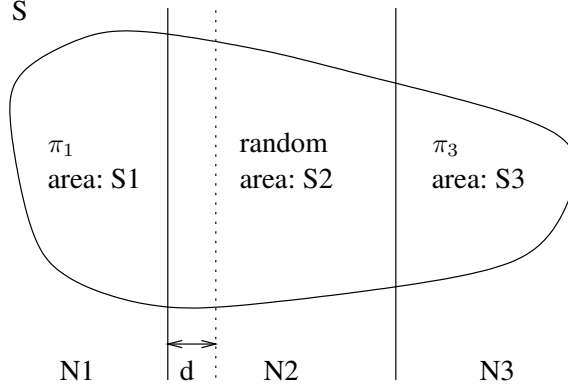$$\operatorname{gain}(C) = K - \frac{|C|}{|S|} E(\pi_1) - \frac{|\overline{C}|}{|S|} E(\pi_3)$$

Figure 2: Illustration of the notations.

where K is a constant independent of C. Using this value we introduce a simplification assumption:

$$\text{gain}(C_d) = K - \frac{|C_d|}{|S|}E(p_1(d)) - \frac{|\overline{C}_d|}{|S|}E(p_2(d)) \tag{1}$$

where

$$p_1(d) = \frac{\pi_1 N_1 + 0.5d}{N_1 + d}, \quad p_2(d) = \frac{0.5(N_2 - d) + \pi_3 N_3}{N_2 - d + N_3}$$

and C_d is any concept that was created from C by adding d elements from S_2. The point is that we replace the average of the entropies with the entropy of the average of $|G_f \cap C|/|C|$ over the domain. This introduces some error since it is possible only for linear functions of probability variables and entropy is not linear. This error depends on the actual distribution of $|G_f \cap C|/|C|$ over the domain. The lower the variance the higher the accuracy of the approximation. Furthermore, recall that we have chosen the smaller values for π_1 and π_3 so we use only the first half of the entropy function (over the interval $[0, 0.5]$) and here (apart from the neighborhood of the ends of the interval) it is not very far from linearity, so the accuracy depends also on the actual values of π_1 and π_2.

Now we can prove the following theorem:

Theorem 1. *Using the assumption in (1)*

$$C_i = \arg \max_{C \in \{C_0, \dots, C_{N_2}\}} \text{gain}(C)$$

holds only for $i = 0$ or $i = N_2$.

Proof. We are looking for the maximum of the information gain

$$\text{gain}(C_d) = K - \frac{(N_1 + d)E(p_1(d)) + (N_2 - d + N_3)E(p_2(d))}{N_1 + N_2 + N_3}$$

The problem is equivalent to finding the minima of the counter of the fraction, the average entropy. I will show that this function is concave on the interval $[0, N_2]$ which directly proves the theorem. It is sufficient to show that the first term $(N_1+d)E(p_1(d))$

6

is concave, the other term has a symmetrical structure and the sum of concave functions remains concave. The second derivative of the first term is

$$\frac{-1}{2(1-\pi_1)N_1 + d} + \frac{-1}{\pi_1 N_1 + 0.5d}$$

which is negative so the proof is complete. □

This theorem means that either S_2 is included in the optimal concept or it is excluded completely. Applying this to every subdomain that arises during the running of the algorithm we get the result mentioned in the introduction of this section. It is very interesting to briefly relate this finding to Radcliffe's notion of a good encoding [10]. According to his model, the equivalence classes of the encoding should have a low fitness variance. This can be applied not only to single functions but to domains as well since it is possible to take e.g. the average of the variances of the functions of the domain. This is a special case of our approach since if the low variance property holds over a subdomain then according to our approach it will be a good candidate for being an optimal concept since all the solutions will tend to be good or bad due to low variance and therefore the entropy will tend to be low. However in the case of random domains the variance is not necessarily low.

4   Structure in the Linear Domain

For illustration of the potentials of the approach let us take a closer look at a domain of special significance: the linear functions over the binary search space $S = \{0, 1\}^n$. A function $f : S \longrightarrow \mathbb{R}$ is linear with the coefficient vector $\mathbf{a} \in \mathbb{R}^n$ if

$$f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} = \sum_{i=1}^n a_i x_i$$

In this section three subdomains of the general linear functions will be studied. Every subdomain will have prototypical functions, but the definitions are intended to be fuzzy. The closer examination of these subdomains is useful for two reasons. The first is that we will see that on certain subdomains the efficiency of the search can be significantly improved. The second is that this discussion will illustrate a major point of this work: different domains may have dramatically different structure and thus different optimal encodings even if the mathematical description of the functions of the domains have the same form.

Some claims of the section are based on experimental data. In all the experiments 8 bit domains were used and a concept was implemented as an explicit characteristic function (i.e. a list of 256 truth values). The concepts were optimized using a simple multistart hillclimber run until 10000 evaluations restarted when no one-bit change resulted in improvement. The other specific details are given in the subsections.

4.1   Orderable Problems

The coefficient vector of an *orderable* function contains numbers that differ in their order of magnitude significantly. In other words the coefficients (and thus the bits of a solution) can be ordered according to dominance. Prototypical examples are vectors with $|a_i| = 2^i$, $(i = 1, \ldots, n)$. It is easy to see that for orderable domains the optimal encoding will be the collection of schemata of length 1. As an additional benefit, the dominance order is also given by the algorithm.

4.2 Counting Problems

Here the coefficients do not differ in magnitude and they have the same sign. The coefficients of the prototypes of such problems are equal to a given constant: $a_i = c$, $(i = 1, \ldots, n)$. We have run experiments with domains containing 100 functions where the coefficients of a particular function were drawn from $[100, 120]$ (or $[-120, -100]$) to introduce some noise. Note that the value of solutions which have the same number of 1s is similar. Thus they generate random subdomains in the sense of Section 3.

The experiments confirmed our theoretical assumptions in that the concepts found during search never divided such a subdomain in any single run, only when the whole domain to divide was random. Surprisingly (to me), when trying to divide such a random domain the algorithm did find structure consistently. Closer analysis showed that this structure is due to the noise we introduced and can approximately be translated into an additional heuristic which says that in a space of solutions containing the same number of 1s divide the space using the bit which has the smallest coefficient on average.

Note that the length of the optimal encoding is proportional to $\log n$ so a significant reduction of the search space can be achieved while the fitness variance of the inverse sets of the codes is low therefore this subdomain with the optimal encoding is much easier than the general linear domain.

4.3 Hamming Problems

Here the coefficients do not differ in magnitude but they may have different signs. A prototypical example could be $a_i = (-1)^n$, $(i = 1, \ldots, n)$. The value of a Hamming function depends on the Hamming distance from a given binary vector.

Experiments were run using a 100 function domain where the coefficients of a particular function were drawn from $[100, 120]$ and their sign was random. The maximal information gain of the first concept that was found by the hillclimber was 0.048 with a variance of 0.003 (from 10 experiments). This value is quite low given that on completely random domains the expected maximal gain is around 0.01 according to our simulations, and in the case of counting problems this value is 0.39 on average. Analyzing the optimal first concepts we can define the following heuristic: divide the space according to a one bit schema. Applying this heuristic explicitly we get a gain of 0.046 on average with a variance of 0.002 (10 experiments).

The conclusion is that the optimal encoding is the natural encoding as in the case of orderable problems but the information gain is significantly lower. This indicates that Hamming problems are harder then orderable problems since the fitness variance is much larger and the problems are much more sensitive to sampling error and genetic drift.

5 Probabilistic Models

The approach presented here can be considered as a generalization of search techniques that use dynamic probabilistic models to generate good solutions [8]. A probabilistic model of the good region of the space has a close relationship to our notion of concept. As mentioned earlier, a concept has to be invertible; we have to be able to generate solutions that satisfy a given concept. A probabilistic model is in fact a fuzzy concept which is of course invertible.

A practical implementation of the algorithm applied to a domain containing only a single function may be very similar to algorithms using probabilistic models since during the recursive building of the optimal encoding the gain of new concepts can be evaluated on solutions generated using the inverses of available codes. Furthermore — as a trivial extension — every subdomain can be labeled positive if the functions over it contain good solutions consistently (recall that large information gain requires only homogeneity), and emphasis can be moved to explore those regions further, even if the domain contains several functions.

6 Conclusions and Future Work

The purpose of this paper was to motivate, to theoretically ground and to illustrate an automatic encoding generation technique. We have seen that the method cuts search spaces along their "natural joints" in the sense that random domains are never cut in half. This also means — considering the structure of the algorithm as well — that the non-empty inverses of the optimal codes define either random domains or low fitness variance domains. It was demonstrated that even in the case of the linear functions three subdomains can be defined that have significantly different complexity. This also implies that similar or identical mathematical structure is not necessarily sufficient to characterize a domain: the distribution of the parameters of the functions is also essential [4, 2, 3].

Here I would like to touch on some problems of practical, real world applications and its limitations. One main problem to solve when implementing the system is to chose the actual representations of the abstract notion of concept. In the case of big spaces this representation is naturally a function class. The literature on machine learning provides us with an endless number of opportunities, the class of feedforward artificial neural networks (ANNs) is a good example. The only important constraint is the invertibility condition of the encoding.

Another important issue is the bias introduced by the chosen representation. When restricting ourselves to a specific function class we risk the possibility that we cannot describe the structure of the domain under consideration. For example if parity of bits plays an important role in a binary domain then there is practically no chance to capture this using feedforward ANNs. However this is not a specific problem of the present approach: it is the problem of machine learning in general.

Finally, for finding good concepts samples of the functions in the domain are needed. Two natural approaches seem to be reasonable. The first is to use the trajectories that were produced by search algorithms on the functions of the domain. The other is to recursively generate new solutions based on the available concepts and to evaluate them. Both methods assume a larger time-scale than ordinary optimization methods but the output, the interchangeable and reusable knowledge about important problem domains may pay off in the long term.

References

[1] Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, 1999. Morgan Kaufmann.

[2] Márk Jelasity. A wave analysis of the subset sum problem. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 89–96, San Francisco, California, 1997. Morgan Kaufmann.

[3] Márk Jelasity. The adaptationist stance and evolutionary computation. In Banzhaf et al. [1], pages 1859–1864.

[4] Márk Jelasity and József Dombi. Implicit formae in genetic algorithms. In W. Ebeling, Ingo Rechenberg, Hans-Paul Schwefel, and Hans-Michael Voigt, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computational Science*, pages 154–163. Springer-Verlag, 1996.

[5] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hillclimbing? In J. D. Cowan et al., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994.

[6] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[7] Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In Banzhaf et al. [1], pages 433–440.

[8] Martin Pelikan, David E. Goldberg, and Fernando Lobo. A survey of optimization by building and using probablistic models. Technical Report 99018, Illinois Genetic Algorithms Laboratory, 1999.

[9] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[10] Nicholas J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384, 1994.

[11] Alexander I. Rudnicky, Alexander G. Hauptmann, and Kai-fu Lee. Survey of current speech technology. *Communications of the ACM*, 37(3):52–57, March 1994.

[12] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.