Efficient Evolutionary Optimization through the use of a Cultural Algorithm

Carlos A. Coello Coello and Ricardo Landa Becerra

CINVESTAV-IPN

Evolutionary Computation Group

Departamento de Ingeniería Eléctrica

Sección de Computación

Av. IPN No. 2508

Col. San Pedro Zacatenco

México, D. F. 07300

ccoello@cs.cinvestav.mx

rlanda@computacion.cs.cinvestav.mx

August 1, 2003

Abstract

This paper introduces a cultural algorithm that uses domain knowledge to improve the performance of an evolutionary programming technique adopted for constrained optimization. The proposed approach extracts domain knowledge during

the evolutionary process and builds a map of the feasible region to guide the search more efficiently. Additionally, in order to have a more efficient memory management scheme, our implementation uses 2^n-trees to store this map of the feasible region. Our results indicate that the approach is able to produce very competitive results with respect to other optimization techniques at a considerably lower computational cost.

Keywords: Cultural algorithms, evolutionary programming, engineering optimization.

1 Introduction

The use of evolutionary algorithms for solving optimization problems has become very extensive in the last few years [14, 2]. This popularity is mainly due to the robustness, ease of use and wide applicability of evolutionary algorithms [12].

However, it is commonly the case that evolutionary algorithms are seen as "blind heuristics" in the sense that they do not use or require any specific domain knowledge. Nevertheless, several researchers have proposed different mechanisms to extract knowledge (or certain design patterns) from an evolutionary algorithm in order to improve convergence of another evolutionary algorithm (see for example [35, 20, 27]).

In this paper, we propose the use of a biological metaphor called a "cultural algorithm" as a global optimization technique. Cultural algorithms are based on the following notion: in advanced societies, the improvement of individuals occurs beyond natural selection; besides the information that an individual possesses within his genetic code (inherited from his ancestors) there is another component called "culture".

Culture can be seen as a sort of repository where individuals place the information acquired after years of experience. When a new individual has access to this library of information, it can learn things even when it has not experienced them directly. Humankind as a whole has reached its current degree of progress mainly due to culture.

In this paper, we propose an approach in which domain knowledge (using the concept of a cultural algorithm) extracted during a run of an evolutionary algorithm is used to guide the search more efficiently in constrained optimization problems [24, 7].

The remainder of this paper is organized as follows. In Section 2, we provide some basics of cultural algorithms. Section 3 discusses the most important previous related work. The use of cultural algorithms in constrained optimization is discussed in Section 4. The way in which constraints are handled as a belief space is discussed in Section 5. Our proposed approach is described in Section 6. The mathematical description of the examples used to validate our approach are provided in Section 7. Results are compared with respect to other approaches in Section 8. Finally, our conclusions and some possible paths for future research are provided in Section 9.

2 Basics of Cultural Algorithms

Cultural algorithms were developed by Robert G. Reynolds as a complement to the metaphor used by evolutionary algorithms, which had focused mainly on genetic and natural selection concepts [30].

Cultural algorithms are based on some theories originated in sociology and archaeology which try to model cultural evolution. Such theories indicate that cultural

evolution can be seen as an inheritance process operating at two levels: (1) a micro-evolutionary level, which consists of the genetic material that an offspring inherits from its parents, and (2) a macro-evolutionary level, which consists of the knowledge acquired by individuals through generations. This knowledge, once encoded and stored, is used to guide the behavior of the individuals that belong to a certain population [29, 11].

Culture can be seen as a set of ideological phenomena shared by a population. Through these phenomena, an individual can interpret its experiences and decide its behavior. In these models, we can clearly appreciate the part of the system that is shared by the population: the knowledge, acquired by members of a society, but encoded in such a way that such knowledge can be accessed by every other member of the society. And then there is an individual part, which consists of the interpretation of such knowledge encoded in the form of symbols. This interpretation will produce new behaviors as a consequence of the assimilation of the corresponding knowledge acquired combined with the experiences lived by the individual itself.

Reynolds attempts to capture this double inheritance phenomenon through his proposal of cultural algorithms [30]. The main goal of such algorithms is to increase the learning or convergence rates of an evolutionary algorithm such that the system can respond better to a wide variety of problems [13].

Cultural algorithms operate in two spaces. First, we have the population space, which consists of (as in all evolutionary algorithms) a set of individuals. Each individual has a set of independent features that are used to determine its fitness. Through time, such individuals can be replaced by some of their descendants, which are obtained

from a set of operators applied to the population.

The second space is the belief space, which is where we store the knowledge acquired by individuals through generations. The information contained in this space must be accessible to each individual, so that they can use it to modify their behavior.

In order to join the two spaces, it is necessary to provide a communication link, which dictates the rules regarding the type of information that must be exchanged between the two spaces. The pseudo-code of a cultural algorithm is shown in Algorithm 1.

Algorithm 1 Pseudo-code of a cultural algorithm.

```
Generate the initial population
Initialize the belief space
Evaluate the initial population
Repeat
    Update the belief space (with the individuals accepted)
    Apply the variation operators (under the influence of the belief
    space)
    Evaluate each child
    Perform selection
While the end condition is not satisfied
```

Most of the steps of a cultural algorithm correspond with the steps of a traditional evolutionary algorithm. It can be clearly seen that the main difference lies in the fact that cultural algorithms use a belief space. In the main loop of Algorithm 1, we have the update of the belief space. It is at this point in which the belief space incorporates the individual experiences of a select group of members of the population. Such a group is obtained with the function *accept*, which is applied to the entire population. On the other hand, the variation operators (such as recombination or mutation) are modified by the function *influence*. This function applies some pressure such that the children

resulting from the variation operators can exhibit behaviors closer to the desirable ones and farther away from the indesirable ones, according to the information stored in the belief space.

These two functions (*accept* and *influence*) constitute the communication link between the population space and the belief space. Such interactions can be appreciated in Figure 1 [31].

In [30], Reynolds proposed the use of genetic algorithms to model the micro-evolutionary process, and Version Spaces [25] to model the macro-evolutionary process of a cultural algorithm. This sort of algorithm was called the *Version Space guided Genetic Algorithm* (VGA). The main idea behind this approach is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. Therefore, if we apply a cultural algorithm for global optimization, the acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level [22].

3 Related Work

Reynolds et al. [32] and Chung & Reynolds [6] have explored the use of cultural algorithms for global optimization with very encouraging results. Chung and Reynolds [6] use a hybrid of evolutionary programming and GENOCOP [23] in which they incorporate an interval constraint-network [8] to represent the constraints of the problem at hand. An individual is considered as "acceptable" when it satisfies all the constraints of the problem. When that does not happen, then the belief space, *i.e.*, the intervals associated with the constraints, are adjusted. This approach is really a more sophisti-

cated version of a repair algorithm in which an infeasible solution is made feasible by replacing its genes with a different value between its lower and upper bounds. Since GENOCOP assumes a convex search space, it is relatively easy to design operators that can exploit a search direction towards the boundary between the feasible and infeasible regions.

In further work, Jin and Reynolds [17] proposed an *n*-dimensional regional-based schema, called a *belief-cell*, as an explicit mechanism that supports the acquisition, storage and integration of knowledge about non-linear constraints in a cultural algorithm. This *belief-cell* can be used to guide the search of an Evolutionary Computation (EC) technique (evolutionary programming in this case) by pruning the instances of infeasible individuals and promoting the exploration of promising regions of the search space. The key aspect of this work is precisely how to represent and save the knowledge about the problem constraints in the belief space of the cultural algorithm.

The idea of Jin and Reynolds' approach is to build a map of the search space similar to the "Divide-and-Label" approaches used for robot motion planning [19]. This map is built using information derived from evaluating the constraints of each individual in the population of the EC technique. The map is formed by dividing the search space in sub-areas called *cells*. Each cell can be classified as: feasible (if it lies completely in a feasible region), infeasible (if it lies completely in an infeasible region), semi-feasible (if it occupies part of a feasible and part of an infeasible region), or unknown (if that region has not been explored yet). This map is used to derive rules about how to guide the search of the evolutionary algorithm (avoiding infeasible regions and promoting the exploration of feasible regions).

This previous work, however, has an important drawback: the authors do not indicate how to implement the belief space and, from their publications, one can infer that static data structures were adopted in their work. This has some important scalability issues since a relatively low dimensionality (about 20 decision variables) can become impractical in terms of the cell representation needed (i.e., we would run out of memory).

4 Constrained Optimization

In this paper, we use cultural algorithms with evolutionary programming (CAEP) [6]. The basic idea is to "influence" the mutation operator (the only operator in evolutionary programming) so that current knowledge about the properties of the search space can be properly exploited.

As indicated above, in a cultural algorithm there are two main spaces: the normal population adopted with evolutionary programming and the belief space. The shared acquired knowledge is stored in the belief space during the evolution of the population. The interactions between these two spaces are detailed below [6]:

1. Select an initial population of p candidate solutions, from a uniform distribution within the given domain for each parameter from 1 to n.

2. Assess the performance score of each parent solution by a given objective function f.

3. Initialize the belief space with the given problem domain and candidate solutions.

4. Generate p new offspring solutions by applying a variation operator, V, as modified by the influence function, *Influence*. Now there are $2p$ solutions in the population.

5. Assess the performance score of each offspring solution by the given objective function f.

6. For each individual, select c competitors at random from the population of size $2p$. Conduct pairwise competitions between the individual and the competitors.

7. Select the p solutions that have the greatest number of wins to be parents for the next generation.

8. Update the belief space by accepting individuals using the acceptance function.

9. Go back to step 4 unless the available execution time is exhausted or an acceptable solution has been discovered.

As we saw before, in this case, most of the steps previously described are the same as in the evolutionary algorithm adopted (evolutionary programming [12]). The acceptance function accepts those individuals that can contribute with their knowledge to the belief space. The update function creates the new belief space with the beliefs of the accepted individuals. The idea is to add to the current knowledge the new knowledge acquired by the accepted individuals.

The function to generate offspring used in evolutionary programming is modified so that it includes the influence of the belief space in the generation of offspring. Evolutionary programming uses only mutation and the influence function indicates the most

promising mutation direction. The remaining steps are the same as used in evolutionary programming.

For unconstrained problems, Chung [5] proposes the use of two types of knowledge: (1) situational, which provides the exact point where the best individual of each generation was found; and (2) normative, which stores intervals for the decision variables of the problem that correspond to the regions where good results were found.

5 Beliefs as Constraints

As we mentioned before, Jin and Reynolds [17] modified Chung's proposal so as to include in the belief space information about feasibility of the solutions. We will explain next the changes performed in more detail, since our current proposal is an extension of Jin & Reynolds' algorithm.

First, Jin and Reynolds eliminated the situational knowledge and added constraints knowledge. Taking advantage of the intervals of good solutions that are stored in the normative portion of the belief space, they created what they called "belief cells". These belief cells are a subdivision of the search space within the intervals of good solutions, such that feasibility of the cells can be determined. When the intervals of the variables are modified, the cells are also modified. As indicated before, there are 4 types of cells (see Figure 2)[1]: (1) feasible, (2) infeasible, (3) semi-feasible (contain part of both areas) and (4) unknown.

The influence that the belief space has on the generation of offspring consists of

[1] Other authors have also proposed the use of a map of the feasible region. See for example [21].

moving individuals that lie in infeasible cells towards feasible cells. Actually, in this process, semi-feasible cells are given preference because in most difficult constrained problems, the optimum lies on the boundary between the feasible and infeasible regions. However, Jin & Reynolds [17] do not modify the rules used to update the normative part of the belief space proposed by Chung [5]: the intervals are expanded if the accepted individuals do not fit within them; conversely, they are tightened only if the accepted individuals have a better fitness. This may reduce the intervals towards infeasible regions in which the objective function values are higher.

6 Description of our Approach

The approach proposed here is a variation of Jin & Reynolds' technique [17]. However, in our case, we incorporate spatial data structures (2^n-trees) in order to store the map of the feasible region more efficiently. Next we will describe the main differences between traditional evolutionary programming and our approach.

6.1 Initialization of the Belief Space

The lower and upper boundaries of the promising intervals for each variable are stored in the normative part of the belief space, together with the fitness for each extreme of the interval. This part is initialized by putting in the boundaries of the variables the values given in the input data of the problem. The initial fitnesses in all cases are set to $+\infty$.[2]

[2]This is assuming a minimization problem.

Regarding the constraints of the problem, the interval given in the normative part is subdivided into s subintervals such that a portion of the search space is divided in hypercubes (see Figure 3). The following information about each hypercube is stored: number of feasible individuals (within that cell), number of infeasible individuals (within that cell), and the type of region. The type of region depends on the feasibility of the individuals within. Four types are defined:

- **if** $feasible\ individuals = 0$ **and** $infeasible\ individuals = 0$, **then** $cell\ type = unknown$
- **if** $feasible\ individuals > 0$ **and** $infeasible\ individuals = 0$, **then** $cell\ type = feasible$
- **if** $feasible\ individuals = 0$ **and** $infeasible\ individuals > 0$, **then** $cell\ type = infeasible$
- **if** $feasible\ individuals > 0$ **and** $infeasible\ individuals > 0$, **then** $cell\ type = semi - feasible$

To initialize this part, all counters are set to zero and the cell type is initialized to "unknown" (other values could be used in this case, but that would obviously affect the performance of the algorithm).

6.2 Updating the Belief Space

The constraints part of the belief space is updated at each generation, whereas the normative part is updated every k generations. The update of the constraints part consists

only of adding any new individuals that fall into each region to the counter of feasible individuals. The update of the normative part is more complex (that is the reason why it is not performed at every generation). When the interval of each variable is updated, the cells or hypercubes of the restrictions part are changed and the counters of feasible and infeasible individuals are reinitialized. Furthermore, this update is done taking into consideration only a portion of the population. Such a portion is selected by the function *accept()*, taking as a parameter (given by the user) the percentage of the total population size to be used. We set this percentage to 25% in our experiments, based on some empirical testing. Note that changing this value does not significantly affect the computational cost of the algorithm, but it may affect the results that it produces. The interpretation of this percentage in terms of its role in the algorithm is that it regulates the rate at which the knowledge gets specialized. As this percentage approaches 100%, the knowledge gets specialized at a slower rate, and viceversa. We found that 25% was a good compromise. The function *accept()* selects the best individuals, based on their number of victories obtained during the selection process.

In the approach proposed in this paper, the conditions to reduce the intervals are stronger than those in previous approaches (e.g., [17]): an interval is reduced only if the accepted individual has a better fitness AND it is feasible. In order to make this mechanism work, it is necessary to modify the acceptance function so that feasible individuals are preferred and fitness is adopted as a secondary criterion. If this is not done, then the condition for interval reduction will not hold most of the time because the accepted individuals are more likely to be infeasible.

6.3 Influence of Beliefs on the Mutation Operator

Mutation takes place for each variable of each individual, with the influence of the belief space and in accordance with the following rules:

- If the variable j of the parent is outside the interval given by the normative part of the constraints, then we attempt to move within this interval through the use of a random variable.
- If the variable is within a feasible, a semi-feasible or an unknown hypercube, the perturbation is made trying to place it within the same hypercube or very close to it.
- Finally, if the variable is in an infeasible cell, we try to move it first to the closest semi-feasible cell. However, if none is found, we try to move it to the closest feasible or unknown cell. If that does not work either, then we move it to a random position within the interval defined by the normative part.

6.4 Tournament Selection

The rules for updating the belief space may result in that knowledge becoming specialized at a slower rate. To improve the speed of the algorithm, we take advantage of the rules for performing tournament selection. After performing mutation, we will have a population of size $2p$ (p parents generate p children). The tournament is performed considering the entire population (i.e., we use $(\mu+\lambda)$ selection with $\mu=\lambda=p$). Tournaments consist of c confrontations per individual, with the c opponents randomly chosen from the entire population. When the tournaments finish, the p individuals with

the largest number of victories are selected to form the following generation. The tournament rules adopted for the current proposal are very similar to those adopted by Deb in his penalty approach based on feasibility [10].

The new tournament rules adopted by our approach are the following:

1. If both individuals are feasible, then the individual with the best objective function value wins.

2. If both individuals are infeasible, then the individual with the lowest constraint violation wins. The constraint violation is measured using:

$$sumg(\vec{x}) = \sum_{j \in J} \frac{g_j(\vec{x})}{gmax_j}$$

   where $gmax_j$ is the largest value of the constraint g_j found during the evolutionary process, and $J = \{ j \mid g_j(\vec{x})$ is a constraint violated in $\vec{x}\}$.

   In words, we are saying that the winner is the individual that presents a lower constraint violation, considering normalized constraints (this normalization is done to avoid problems with the use of different units for each of the constraints used).

3. Otherwise, the feasible individual always wins.

6.5 Use of 2^n-Trees

One of the main drawbacks of Jin & Reynolds' approach [17] is its intense memory usage. Since the belief maps of each decision variable have to be stored, the approach

runs out of memory very fast and cannot possibly handle problems with more than a few decision variables (memory requirements grow exponentially with the number of decision variables of the problem). This led us to develop a scheme in which 2^n-trees are used to partition the feasible region into cells so that with higher-dimensionality problems the memory usage is not exponentially increased. The idea was inspired by the popularity of spatial data structures to store efficiently navigation maps in robotics [19] and to represent efficiently 3D objects in computer graphics [16].

In order to be able to use 2^n-trees within our implementation, we have to partition only the projection of the search space in some dimensions, since 2^n-trees have practical use only when $n \leq 4$, where n corresponds to the number of decision variables of our problem [19]. An example of how to partition a 2D space using a quadtree with a depth of 2 is shown in Figure 4.

Note that the decision of how to partition decision variable space so as to comply with this restriction is very important since the number of nodes used may be incremented rather than reduced! For example, if an octree is adopted, using a node division we will divide three dimensions and our tree will have $2^3 + 1 = 9$ nodes in total. However, if we use a tree that divides only one dimension and through 3 successive divisions we partition a 3D space, the leaf nodes will give the same result as for the octree but using 15 nodes.

From the previous discussion we can infer that we should use a 2^n-tree with the largest possible n, but being careful not to use too much memory. Our conjecture is that $n = 3$ is the largest number with which the problem remains manageable.

Once the number of dimensions to be partitioned has been decided, we have to de-

cide which are the dimensions to be partitioned. The idea is to choose the 3D projection that best divides the search space into a feasible and an infeasible region (or regions). However, since the number of possible combinations of three dimensions grows exponentially with the number of variables, it soon becomes impossible to try them all. Therefore, we can choose a group of combinations to be tried such that the size of this group grows linearly with the number of variables of the problem. In order to determine the "goodness" of a certain partition, we have to count the number of feasible and infeasible individuals in each leaf node. A node will be considered good as long as one of these two values (i.e., feasible and infeasible individuals) tends to zero. For example, let's assume that n_f is the number of feasible individuals in a certain node and that n_i is the number of infeasible individuals in that same node. Thus, a small number $\min(n_f, n_i)$ in each node will indicate a good partition. From the previous discussion, we can say that we are looking for a partition that minimizes:

$$\lambda = \sum_{leaf\,nodes} \min(n_f, n_i)$$

Having this partition, we can continue partitioning with the same method until reaching the maximum allowable depth. Since we have tried several partitioning methods for a single node division, it is better to choose a small depth limit so that not much time is spent in the creation of the tree.

The method described to expand nodes is only done for nodes corresponding to semi-feasible cells, and it stops when it reaches the maximum depth of the tree. The tree is rebuilt every time the normative part is updated.

7 Examples

To validate our approach, we have used some test functions from the well-known benchmark proposed in [24] which has been often used in the literature to validate new constraint-handling techniques. Additionally, we also used some well-known engineering optimization problems. All the problems are described in Appendix A.

8 Comparison of Results

For all the experiments reported next, we performed 10 independent runs per problem, and we used the following parameters in our approach: population size = 20, maximum number of generations = 2500, the normative part is updated every 20 generations with 25% of the population (acceptance %), tournaments consist of 10 encounters per individual (half the population size), the maximum depth of the octree is equal to the number of decision variables of the problem. These parameters were derived empirically after numerous experiments.

8.1 Example 1 : g01

In this case, the global optimum is at $x^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ where $f(x^*) = -15$. The constraints g_1, g_2, g_3, g_4, g_5 and g_6 are active. The results of our approach and the homomorphous maps of Koziel and Michalewicz [18] are shown in Tables 1 and 2.

8.2 Example 2: g02

The global maximum is unknown; the best reported solution is [33]: $f(x^*) = 0.803619$. Constraint g_1 is close to being active ($g_1 = -10^{-8}$). The results of our approach and the homomorphous maps of Koziel and Michalewicz [18] are shown in Tables 1 and 2.

8.3 Example 3: g04

In this example, the optimum solution is $x^* = (78, 33, 29.995256025682, 45, 36.775812905788)$ where $f(x^*) = -30665.539$. Constraints g_1 and g_6 are active. The results of our approach and the homomorphous maps of Koziel and Michalewciz [18] are shown in Tables 1 and 2.

8.4 Example 4: g08

In this case, the optimum solution is located at $x^* = (1.2279713, 4.2453733)$ where $f(x^*) = 0.095825$. The results of our approach and the homomorphous maps of Koziel and Michalewciz [18] are shown in Tables 1 and 2.

8.5 Example 5: g12

In this test function, the global optimum is located at $x^* = (5, 5, 5)$ where $f(x^*) = 1$. The results of our approach and the homomorphous maps of Koziel and Michalewciz [18] are shown in Tables 1 and 2.

Note that the homomorphous maps approach of Koziel & Michalewicz [18] is one of the best constraint-handling techniques for evolutionary algorithms known to date. Also, it is important to indicate that the results of Koziel and Michalewicz were ob-

tained with 1,400,000 fitness function evaluations, whereas our approach required only 50,020 fitness function evaluations. Note that our approach has been able to deal with problems that have several variables (**g01** has 13 decision variables and **g02** has 20 decision variables).

As can be seen in Tables 1 and 2, our approach produces very competitive results with respect to the homomorphous maps (which is considerably more difficult to implement) at a fraction of its computational cost (in some cases, we converge to the global optimum). The main reason for this cost reduction is that the belief cells are used to guide the search of the evolutionary algorithm very efficiently, avoiding the algorithm moving to unpromising regions of the search space.

Let us analyze now the results for the engineering optimization problems chosen for this comparative study.

8.6 Example 6: Design of a Welded Beam

This problem was solved before by Deb [9] using a simple genetic algorithm with binary representation, and a traditional penalty function as suggested by Goldberg [14]. It has also been solved by Ragsdell and Phillips [26] using geometric programming. Ragsdell and Phillips also compared their results with those produced by the methods contained in a software package called "Opti-Sep" [34], which includes the following numerical optimization techniques: ADRANS (Gall's adaptive random search with a penalty function), APPROX (Griffith and Stewart's successive linear approximation), DAVID (Davidon-Fletcher-Powell with a penalty function), MEMGRD (Miele's memory gradient with a penalty function), SEEK1 & SEEK2 (Hooke and Jeeves with 2

different penalty functions), SIMPLX (Simplex method with a penalty function) and RANDOM (Richardson's random method).

The results of the techniques previously indicated are compared against those produced by the approach proposed in this paper (see Table 3). In the case of Siddall's techniques [34], only the best solution produced by the techniques contained in "Opti-Sep" is displayed. The mean from the runs performed with our approach was $f(\vec{x}) = 1.9718091$, with a standard deviation of 0.4431313. The worst solution found was $f(\vec{x}) = 3.1797085$, although this solution appeared only once in the runs performed.

8.7 Example 7: Minimization of the Weight of a Tension/Compression Spring

This problem was solved before by Belegundu [3] using the following numerical optimization techniques: Feasible directions (CONMIN and OPTDYN), Pshenichny's Recursive Quadratic Programming (LINRM), Gradient Projection (GRP-UI), Exterior Penalty Function (SUMT), and Multiplier Methods (M-3, M-4 and M-5). Only the best feasible result reported by him is shown in Table 4. Additionally, Arora [1] also solved this problem using a numerical optimization technique called Constraint Correction at constant Cost (CCC). It is important to notice that Arora's solution is actually infeasible because it violates one of the constraints slightly. In the experiments reported here, our approach handled all constraints as hard, so that the solutions produced were considered valid only if all of them were fully satisfied. Nevertheless, the proposed approach was able to find a better (feasible) solution than Arora's technique, as can be

seen in Table 4.

The mean from the runs performed with our approach was $f(\vec{x}) = 0.0135681$, with a standard deviation of 0.00084152. The worst solution found was $f(\vec{x}) = 0.0151156$.

We can see that also in the engineering problems chosen, our approach produced very competitive results at a low computational cost (the computational costs of the other approaches against which we compared our algorithm were not available).

9 Conclusions and Future Work

We have presented an approach based on cultural algorithms and evolutionary programming for constrained optimization. The approach proposed has provided good results at a relatively low computational cost both in some well-known test functions used with evolutionary algorithms and in some engineering optimization problems.

We argue that our results suggest that the proper use of domain knowledge can certainly improve the performance of an evolutionary algorithm when such domain knowledge is properly handled. Also, we argue that our results suggest that this domain knowledge can be extracted during the evolutionary process in which we aim to reach the global optimum of a problem. This contrasts with the more conventional approach of using domain knowledge extracted from previous runs of an evolutionary algorithm (see for example [15, 20]).

One of the main drawbacks of cultural algorithms in constrained search spaces (i.e., memory usage) is attacked using spatial data structures that can efficiently store the belief space. To illustrate this point, we will briefly discuss a simple example. With the

2^n-trees adopted in this paper, we defined a maximum depth of 5. Since an octree (such as those used in our approach) has exactly eight child nodes, using a maximum depth of 5, the maximum number of nodes of a tree will be: $8^0 + 8^1 + 8^2 + 8^3 + 8^4 = 4681$ nodes. However, this number is not always reached in practice. If we now consider a static data structure, to divide the space with 12 decision variables, if we just split each dimension in half, we will need $2^{12} = 4096$ nodes. This number is slightly lower than the one used. However, if we now assume 20 decision variables (as in g02), our approach still requires the same number of nodes, whereas a static data structure would require $2^{20} = 1,048,576$ nodes. This would introduce obvious memory management problems. Thus, the mechanism for memory management introduced in our approach is one of our main contributions and it constitutes the main difference with respect to previous proposals.

As part of our future work, we are considering the possibility of using self-adaptation or online adaptation mechanisms that make it unncessary to fine tune the parameters required by our approach. Additionally, we are also considering the possibility of using additional rules in the tournaments performed, so that we can provide more feasibility information to our evolutionary algorithm so as to guide the search in a more effective way (for example, in g02 we were unable to converge to the best known solution). Finally, we are also considering the possible use of alternative data structures for representing the belief space (e.g., *k*-d trees [4]).

Acknowledgements

The authors thank the anonymous reviewers for their comments which greatly helped them to improve the contents of this paper.

The first author acknowledges support from the Consejo Nacional de Ciencia y Tecnología (CONACyT) through project number 32999-A.

The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department of CINVESTAV-IPN.

References

[1] Jasbir S. Arora. *Introduction to Optimum Design*. McGraw-Hill, New York, 1989.

[2] Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*, volume 1. IOP Publishing Ltd. and Oxford University Press, 1997.

[3] Ashok Dhondu Belegundu. *A Study of Mathematical Programming Methods for Structural Optimization*. Department of civil and environmental engineering, University of Iowa, Iowa, Iowa, 1982.

[4] Jon Louis Bentley and Jerome H. Friedman. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.

[5] Chan-Jin Chung. *Knowledge-Based Approaches to Self-Adaptation in Cultural Algorithms*. PhD thesis, Wayne State University, Detroit, Michigan, 1997.

[6] Chan-Jin Chung and Robert G. Reynolds. A Testbed for Solving Optimization Problems using Cultural Algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, Cambridge, Massachusetts, 1996. MIT Press.

[7] Carlos A. Coello Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002.

[8] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[9] Kalyanmoy Deb. Optimal Design of a Welded Beam via Genetic Algorithms. *AIAA Journal*, 29(11):2013–2015, November 1991.

[10] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2/4):311–338, 2000.

[11] W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity*. Stanford University Press, Stanford, California, 1994.

[12] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., New York, 1999.

[13] Benjamin Franklin and Marcel Bergerman. Cultural algorithms: Concepts and experiments. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1245–1251, Piscataway, New Jersey, 2000. IEEE Service Center.

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[15] Eduardo Islas Pérez, Carlos A. Coello Coello, and Arturo Hernández Aguirre. Extraction of Design Patterns from Evolutionary Algorithms using Case-Based Reasoning. In Yong Liu, Kiyoshi Tanaka, Masaya Iwata, Tetsuya Higuchi, and Moritoshi Yasunaga, editors, *Evolvable Systems: From Biology to Hardware (ICES'2001)*, pages 244–255. Springer-Verlag. Lecture Notes in Computer Science No. 2210, October 2001.

[16] C.L. Jackins and S.L. Tanimoto. Octrees and Their Use in Representing Three-Dimensional Objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[17] Xidong Jin and Robert G. Reynolds. Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, D.C., July 1999. IEEE Service Center.

[18] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[19] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[20] Sushil J. Louis and Judy Johnson. Solving Similar Problems using Genetic Algorithms Case-Based Memory. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290, San Francisco, California, 1997. Morgan Kaufmann Publishers.

[21] Carlos E. Mariano and Eduardo F. Morales. Distributed Reinforcement Learning for Multiple Objective Optimization Problems. In *2000 Congress on Evolutionary Computation*, volume 1, pages 188–195, Piscataway, New Jersey, July 2000. IEEE Service Center.

[22] Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. The MIT Press, Cambridge, Massachusetts, 1995.

[23] Zbigniew Michalewicz and Cezary Z. Janikow. Handling Constraints in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[24] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[25] Tom Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

[26] K. M. Ragsdell and D. T. Phillips. Optimal Design of a Class of Welded Structures Using Geometric Programming. *ASME Journal of Engineering for Industries*, 98(3):1021–1025, 1976. Series B.

[27] Connie Loggia Ramsey and John J. Grefenstette. Case-Based Initialization of Genetic Algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California, 1993. Morgan Kauffman Publishers.

[28] Singiresu S. Rao. *Engineering Optimization*. John Wiley and Sons, third edition, 1996.

[29] A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaelogy*. Edinburgh University Press, Edinburgh, Scotland, 1994.

[30] Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.

[31] Robert G. Reynolds. Cultural algorithms: Theory and applications. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 367–377. McGraw-Hill, London, UK, 1999.

[32] Robert G. Reynolds, Zbigniew Michalewicz, and M. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 298–305. MIT Press, Cambridge, Massachusetts, 1995.

[33] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[34] James N. Siddall. *Analytical Design-Making in Engineering Design*. Prentice-Hall, 1972.

[35] Zhiming Zhang and T. Warren Liao. Combining Case-Based Reasoning with Genetic Algorithms. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 305–310, Orlando, Florida, 1999.

A Test Problems

1. **Example 1: g01**:

   Minimize:

$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

   subject to:

$$\begin{aligned}
g_1(\vec{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(\vec{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(\vec{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(\vec{x}) &= -8x_1 + x_{10} \leq 0 \\
g_5(\vec{x}) &= -8x_2 + x_{11} \leq 0 \\
g_6(\vec{x}) &= -8x_3 + x_{12} \leq 0 \\
g_7(\vec{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(\vec{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(\vec{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned}$$

   where the bounds are $0 \leq x_i \leq 1$ ($i = 1, \ldots, 9$), $0 \leq x_i \leq 100$ ($i = 10, 11, 12$) and $0 \leq x_{13} \leq 1$.

2. **Example 2: g02**:

Maximize:

$$f(\vec{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to:

$$g_1(\vec{x}) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g_2(\vec{x}) = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

where $n = 20$ and $0 \leq x_i \leq 10$ $(i = 1, \ldots, n)$.

3. **Example 3: g04**:

Minimize:

$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 +$$
$$37.293239x_1 - 40792.141$$

subject to:

$$g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 +$$

$$0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0$$

$$g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 -$$

$$0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$$

$$g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 +$$

$$0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0$$

$$g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 -$$

$$0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0$$

$$g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 +$$

$$0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0$$

$$g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 -$$

$$0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0$$

where: $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$.

4. **Example 4: g08**

   Minimize:

$$f(\vec{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

   subject to:

$$g_1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0$$

$$g_2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

where $0 \leq x_1 \leq 10$, $0 \leq x_2 \leq 10$.

5. **Example 5: g12**

   Maximize:

$$f(\vec{x}) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100$$

   subject to:

$$g(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$$

   where: $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$, and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds.

6. **Example 6: Design of a welded beam**

   A welded beam is designed for minimum cost subject to constraints on shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), end deflection of the beam (δ), and side constraints [28]. There are four design variables as shown in Figure 5 [28]: h (x_1), l (x_2), t (x_3) and b (x_4).

   The problem can be stated as follows:

Minimize:

$$f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$

Subject to:

$$g_1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \leq 0$$

$$g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \leq 0$$

$$g_3(\vec{x}) = x_1 - x_4 \leq 0$$

$$g_4(\vec{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leq 0$$

$$g_5(\vec{x}) = 0.125 - x_1 \leq 0$$

$$g_6(\vec{x}) = \delta(\vec{x}) - \delta_{max} \leq 0$$

$$g_7(\vec{x}) = P - P_c(\vec{x}) \leq 0$$

where

$$\begin{aligned}
\tau(\vec{x}) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} \\
\tau' &= \frac{P}{\sqrt{2}x_1x_2}, \tau'' = \frac{MR}{J}, M = P\left(L + \frac{x_2}{2}\right) \\
R &= \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \\
J &= 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \\
\sigma(\vec{x}) &= \frac{6PL}{x_4x_3^2}, \delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4}
\end{aligned}$$

$$
\begin{aligned}
P_c(\vec{x}) &= \frac{4.013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}\left(1-\frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \\
P &= 6000\,lb,\ L=14\,in,\ E=30\times10^6\,psi,\ G=12\times10^6\,psi \\
\tau_{max} &= 13{,}600\,psi,\ \sigma_{max}=30{,}000\,psi,\ \delta_{max}=0.25\,in
\end{aligned}
$$

7. **Example 7: Minimization of the Weight of a Tension/Compression Spring**

   This problem was described by Arora [1] and Belegundu [3], and it consists of minimizing the weight of a tension/compression spring (see Figure 6) subject to constraints on minimum deflection, shear stress, surge frequency, limits on outside diameter and on design variables. The design variables are the mean coil diameter D, the wire diameter d and the number of active coils N.

   Formally, the problem can be expressed as:

   Minimize:

$$f(\vec{x}) = (N+2)Dd^2$$

   Subject to

$$
\begin{aligned}
g_1(\vec{x}) &= 1-\frac{D^3N}{71785d^4} \le 0 \\
g_2(\vec{x}) &= \frac{4D^2-dD}{12566(Dd^3-d^4)}+\frac{1}{5108d^2}-1 \le 0 \\
g_3(\vec{x}) &= 1-\frac{140.45d}{D^2N} \le 0 \\
g_4(\vec{x}) &= \frac{D+d}{1.5}-1 \le 0
\end{aligned}
$$

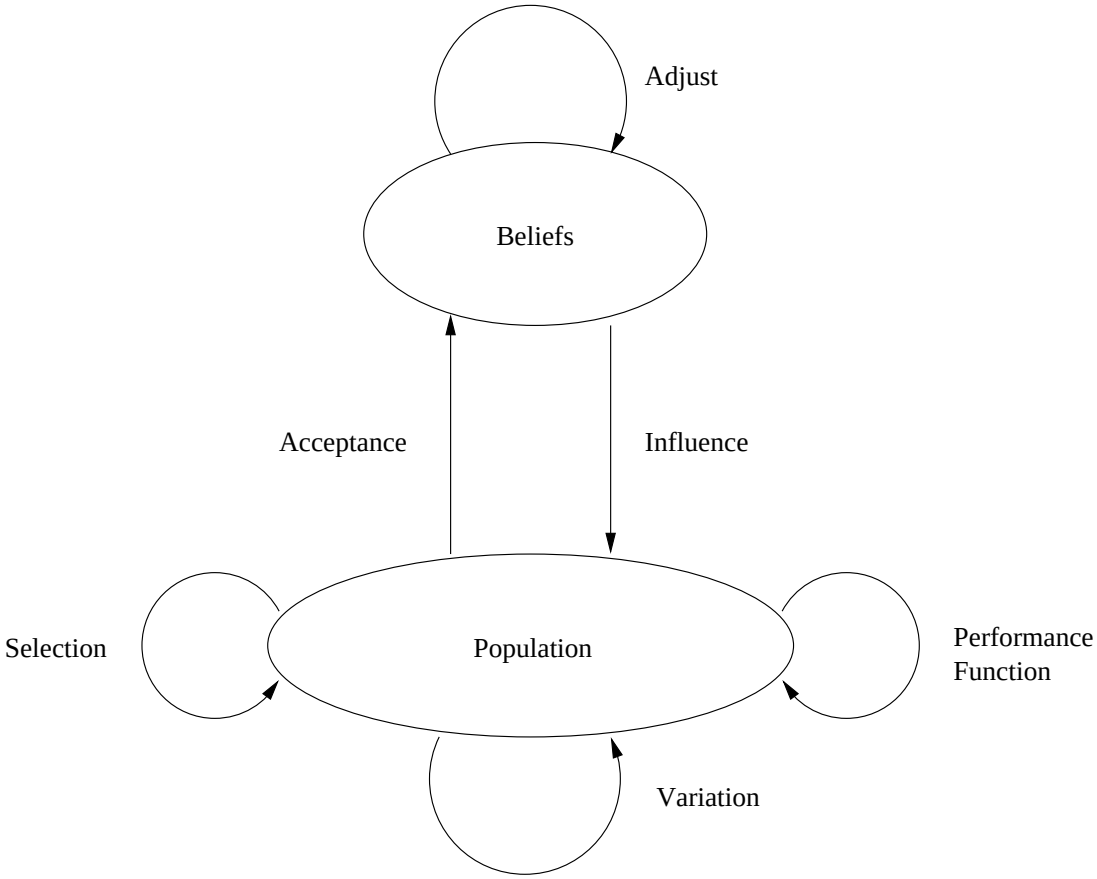

Figure 1: Spaces used by a cultural algorithm.

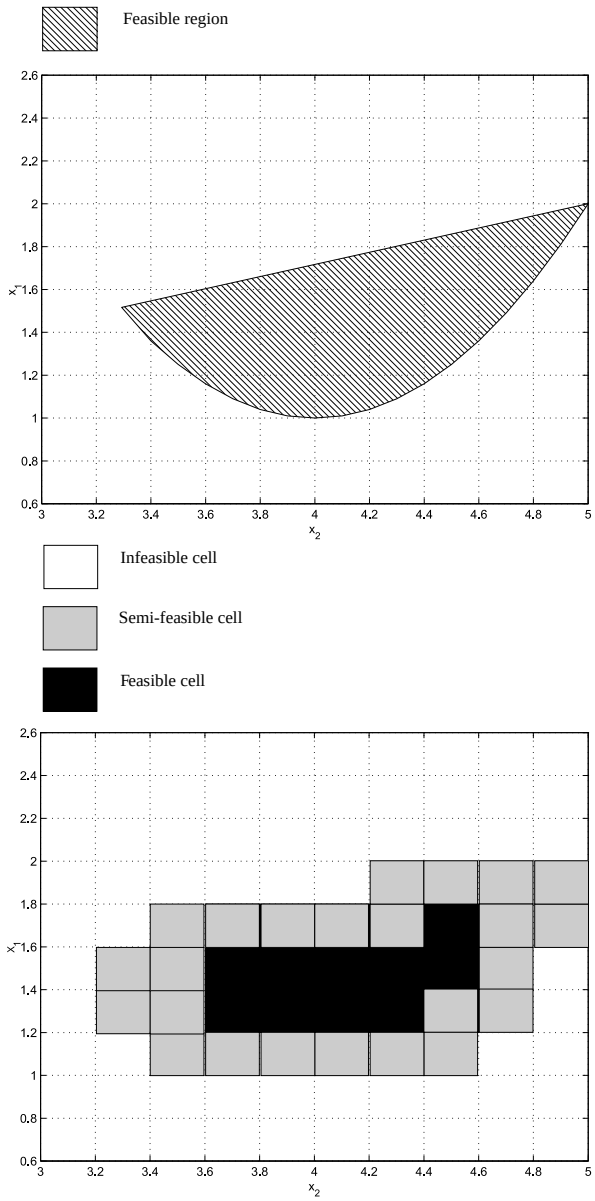

Figure 2: The figure at the top illustrates the feasible region of a problem. The figure at the bottom illustrates the representation of the constraints part of the belief space for the search space of the same problem. In this example, the intervals stored in the normative part must be [0.6, 2.6] for x_1, and [3, 5] for x_2.

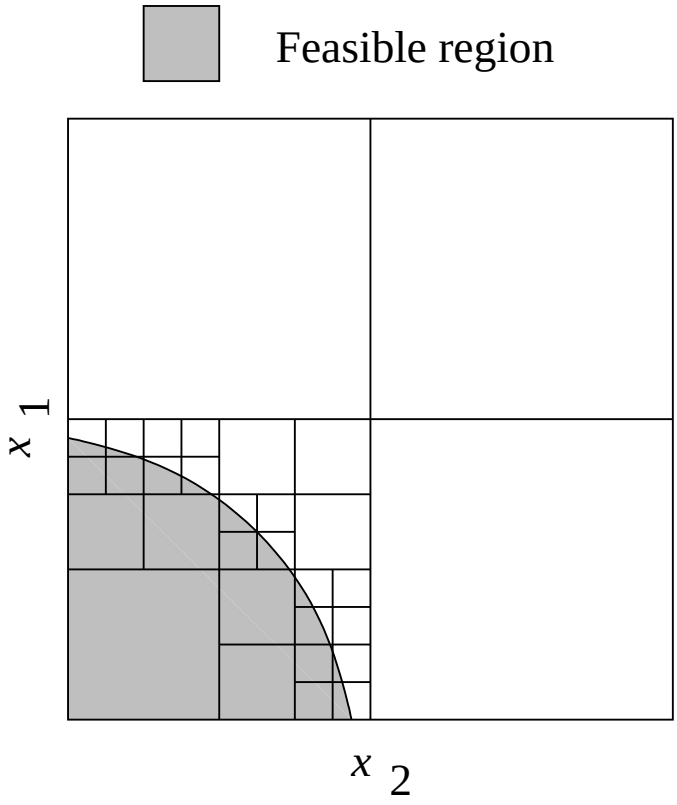

Figure 3: Graphical representation of the division of the semi-feasible cells.

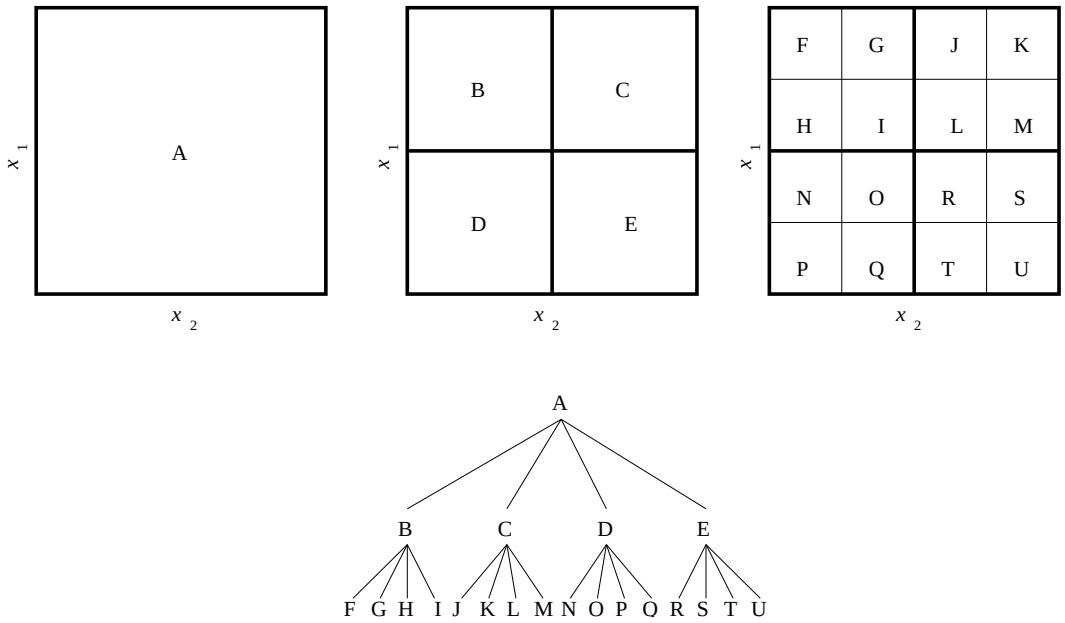

Figure 4: Example of the partition of a 2D space using a quadtree of depth two.

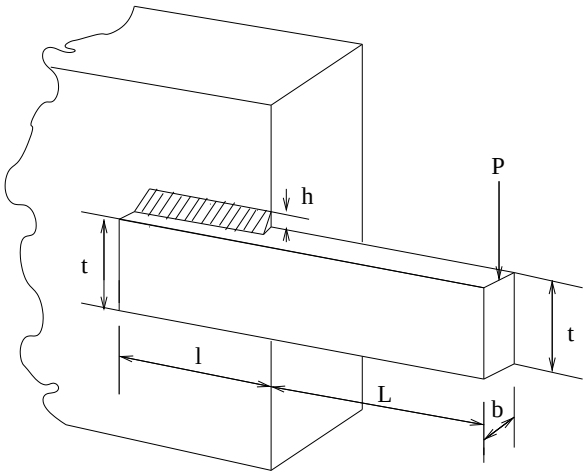

Figure 5: The welded beam used for the sixth example.

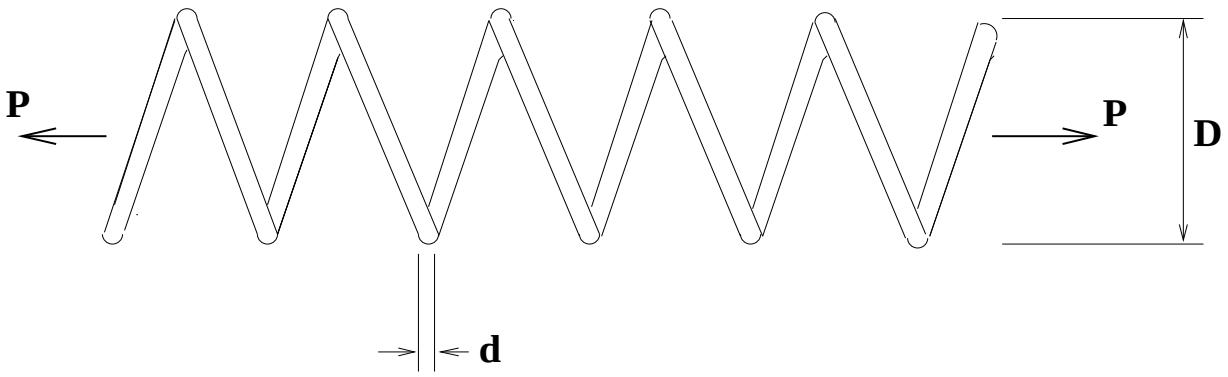

Figure 6: Tension/compression string used for the seventh example.

TF	optimal	Best	Mean	Worst	Std Dev
g01	-15.0	-15.0000	-14.4999	-12.0000	1.0801
g02	0.803619	0.77351	0.66995	0.51762	0.09456
g04	-30665.539	-30665.5	-30662.5	-30636.2	9.3
g08	-0.095825	-0.095825	-0.095825	-0.095825	0.000000
g12	1.0	1.000000	0.996375	0.969375	0.009650

Table 1: Results produced by our CAEP using 2^n-trees.

TF	optimal	Best	Mean	Worst	Std Dev
g01	-15.0	-14.7864	-14.7082	-14.6154	N.A.
g02	0.803619	0.79953	0.79671	0.79119	N.A.
g04	-30665.539	-30664.5	-30655.3	-30645.9	N.A.
g08	-0.095825	-0.095825	-0.089157	-0.029144	N.A.
g12	1.0	0.999999	0.999135	0.991950	N.A.

Table 2: Results produced by the homomorphous maps of Koziel and Michalewicz [18]. N.A. = Not Available

Design Variables	Best solution found			
	CAEP	**Deb [9]**	**Siddall [34]**	**Ragsdell [26]**
$x_1(h)$	0.2057	0.2489	0.2444	0.2455
$x_2(l)$	3.4705	6.1730	6.2189	6.1960
$x_3(t)$	9.0366	8.1789	8.2915	8.2730
$x_4(b)$	0.2057	0.2533	0.2444	0.2455
$g_1(\vec{x})$	-0.000472	-5758.603777	-5743.502027	-5743.826517
$g_2(\vec{x})$	-0.001561	-255.576901	-4.015209	-4.715097
$g_3(\vec{x})$	0.000000	-0.004400	0.000000	0.000000
$g_4(\vec{x})$	-3.432984	-2.982866	-3.022561	-3.020289
$g_5(\vec{x})$	-0.080730	-0.123900	-0.119400	-0.120500
$g_6(\vec{x})$	-0.235540	-0.234160	-0.234243	-0.234208
$g_7(\vec{x})$	-0.000779	-4465.270928	-3490.469418	-3604.275002
$f(\vec{x})$	**1.7248523**	**2.4331160**	**2.3815434**	**2.3859373**

Table 3: Comparison of the results for the sixth example (optimal design of a welded beam).

Design Variables	Best solution found		
	CAEP	**Arora [1]**	**Belegundu [3]**
$x_1(d)$	0.050000	0.053396	0.050000
$x_2(D)$	0.317395	0.399180	0.315900
$x_3(N)$	14.031795	9.185400	14.250000
$g_1(\vec{x})$	0.000000	0.000019	-0.000014
$g_2(\vec{x})$	-0.000075	-0.000018	-0.003782
$g_3(\vec{x})$	-3.967960	-4.123832	-3.938302
$g_4(\vec{x})$	-0.755070	-0.698283	-0.756067
$f(\vec{x})$	**0.0127210**	**0.0127303**	**0.0128334**

Table 4: Comparison of results for the seventh example (minimization of the weight of a tension/compression spring).