IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 2, July 2011
ISSN (Online): 1694-0814
www.IJCSI.org

Intelligent Optimization Techniques for Industrial applications

E. Raj Kumar[1]

[1] **School of mechanical and building science, VIT University**
Vellore, Tamil nadu,632014, India

Abstract

Product recovery and waste management techniques are in popular demand as important elements of environmentally conscious manufacturing. Once a product is received at a recovery facility, it is analyzed to determine its recovery option (i.e., recycling or remanufacturing). Disassembly is considered a key step to retrieving the desired parts and/or subassembly is a systematic product (i.e. partial disassembly); or separating a product into all of its parts (i.e. complete disassembly) for a given purpose. Performing disassembly in an optimized way is properties of the product and the limitations set by the disassembly system. In this paper optimal disassembly sequence can be attained by means of implying Genetic Algorithm and Simulated Annealing technique. A genetic algorithm that generates and evaluates disassembly plans is proposed. This algorithm is able to generate near – optimal disassembly plans through purposely-developed crossover and mutation operators starting from a randomly initialized population of disassembly sequences.

Keywords: Genetic Algorithm, Disassembly plan, simulated annealing,

1. Introduction

Most traditional optimization methods used in engineering applications can be divided into two broad classes: direct search methods requiring only the objective function values and gradient search methods requiring gradient information either exactly or numerically. One common characteristic of most of these methods is that they all work by point-by-point basis. An algorithm starts with an initial point (usually supplied by the user) and depending on the transition rule used in the algorithm a new point is determined. Essentially, algorithms vary according to the transition rule used to update a point.

Thus, we observe that most of the traditional optimization methods are apt for well-behaved, unimodal, simple objective functions. When applied to multimodal problems or problems where gradient information is not available, most of these methods either can not be used or not very efficient. This suggests that in order to solve complex, multimodal, discrete or discontinuous problems, better methods are required. In general, we may be interested in robust search techniques that can be applied to wide variety of problems with ease. As seen from the above discussions that most of the traditional methods are not robust, because each of them is specialized to solve a particular class of problems. That is why there are so many different types of optimization methods each suitable for a particular class of problems. For different problems different algorithms need to be applied.

However, it is to be mentioned here that this discussion is not to say that these traditional algorithms are useless, in fact they have been extensively used in many engineering optimization problems. The suggestion here is that if the solutions obtained by some traditional methods are satisfactory, there is no problem. But if the solutions obtained are not satisfactory or some known methods can not be applied, then the user either has to learn and use some other optimization suitable to solve that problem (which is by no means an easy) matter) or the user has to know some robust search algorithm which can be applied to a wide variety of problems without much problem.

In this chapter, the following intelligent optimization techniques which have been reportedly successful in solving a wide variety of search and optimization problems in sciences, engineering, and commerce are described:

1) Genetic Algorithm (GA)
2) Simulated Annealing (SA)

2.1. Genetic Algorithm (GA)

Genetic algorithms (GAs) are adaptive search and optimization algorithms that mimic the principles of natural genetics. Gas are very different from traditional search and optimization methods used in engineering design problems. Because of their simplicity, ease of operation, minimal requirements and global perspective, GAs have been successfully used in a wide variety of problem domains. Gas were developed by John Holland of University of Michigan in 1965.

2.2. WORKING PRINCIPLE OF GA

Genetic algorithms are search and optimization procedures that are motivated by the principles of natural genetics and natural selection. Some fundamental ideas of genetics are borrowed and used artificially to construct search algorithms that are robust and require minimal problem information. The working principles of GAs are very different from that of most of traditional optimization techniques. A list of them is outlined later. Here, we first describe the working principles of GAs. Unconstrained, single-variable optimization problem given below:

Maximize f(x)
Variable bound Xmin ≥ X ≥ Xmax

In order to use GA to solve the above problem, the variable x is typically coded in some string structures. Binary coded strings are mostly used. The length of the string is usually determined according to the accuracy of the solution desired. For example, if five bit binary strings are used to code the variable x then the string (0 0 0 0 0) is decoded to the value xmin, the string (1 1 1 1 1) is decoded to the value xmix and any other string is decoded to a value in the range (xmin, xmax) uniquely. It is worthwhile to mention here that with five bits in a string there could be only 2 or 32 different strings possible, because each bit position can take a value 0 or a 1. In practice, strings of size hundred or a few hundreds are common, recently a coding with string size equal to 16,384 has also been used. Thus with five bit strings used to code the variable x, the accuracy between two consecutive strings is only (Xmax - Xmin)/32. If more accuracy is desired, longer strings may be used. It is also noteworthy that as the string length increases the minimum possible accuracy in the solution increases exponentially. With a known coding, any string can be decoded to an x value, which can then be used to find the objective function value. A string's objective function value (f(x)) is known as the string's fitness.

2.3A pseudo-code for a simple Genetic algorithm.

Begin
Initialize population;
Evaluate population;
Repeat
Reproduction;
Crossover;
Mutation;
Until (termination criteria);
End.

GAs begin with a population of string structures created at random. Thereafter, each string in the population is evaluated. The population is then operated by three main operators' reproduction, crossover and mutation to create a hopefully better population. The population is further evaluated and tested for termination. If the termination criteria are not met, the population is again operated by above three operators and evaluated. This procedure is continued until the termination criteria are met. One cycle of these operators and the evaluation procedure is known as a generation in GA terminology.

Reproductions usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. There exist a number of reproduction operators in GA literature, but the essential idea is that above average strings are picked from the current population and duplicates of them are inserted in the mating pool. The commonly used reproduction operator is the proportionate selection operator, where a string in the current population is selected with a probability proportional to the string's fitness. Thus, the i-th string in the population is selected with a probability proportional to f. Since the population size is usually kept fixed in a simple GA, the cumulative probability for all strings in the population must be one. Therefore, the probability for selecting i-th string is f i/f avg

where N is the population size. One way to achieve this proportionate selection is to use a roulette wheel with the circumference marked for each string proportionate to the string's fitness. The roulette wheel pointer in the mating pool. Since the circumference of the wheel is marked according to a string's fitness, this roulette wheel mechanism is expected to make copies of the i^{-th} string, where f is the average fitness of the population. Even though this version of roulette wheel selection is somewhat noisy, it is usually used. There exists other more stable versions of this roulette wheel selection.

Crossover operator is applied next to the strings of the mating pool. Like reproduction operator. There exists a number of crossover operators in GA literature but in almost all crossover operators, two strings are picked from the mating pool at random and some portion of the stings are exchanged between the strings. In a single point crossover operator, this is performed by randomly choosing a crossing site among the string and by exchanging all bits on the right side of the crossing site as shown.

0 0 0 0 0 0
1 1 1 1 1 1

It is intuitive from this construction is that good substrings from either parent string can be combined to form a better child string if an appropriate site is chosen. Since the knowledge of an appropriate site is usually not known, a random site, the children strings produced may

or may not have a combination of good substrings from parent string depending on whether the crossing site falls in the appropriate place or not. But we do not worry about this aspect very much, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by the reproduction operator. But if good strings are not created by crossover, they will not survive beyond next generation, because reproduction will not select those strings for the next mating pool. In a two-point crossover operator, two random sites are chosen and the contents bracketed by these sites are exchanged between two parents. This idea can be extended by these sites are exchanged between two parents. This idea can be extended to create a multi-point crossover operator and the extreme of this extension is what is known as a uniform crossover operator. In a uniform crossover for binary strings, each bit from either parent is selected with a probability of 0.5. It is worthwhile to note that the purpose of the crossover operator space. Other aspect is that the search needs to be performed in a way that the information stored in the parent strings are maximally preserved, because these parent strings are instances of good selected using the reproduction operator. In the single-point crossover operator, the search is not extensive, but the maximum information is preserved from parent to children. On the other hand, in the uniform crossover, the search is very extensive but minimum information is preserved between parent and children strings. Even though there exists some studies to find an optimal crossover operator, considerable doubts prevail whether those results can be generalized to be used in all problems. Before any results from theoretical studies are obtained, it is still a matter of personal preference on the choice of crossover operator. However, in order to preserve some good found in the mating pool, not all strings in the population are used in crossover. If a crossover probability of Pc-1 is used then 100% of the population are simply copied to the new population.

Crossover operator is mainly responsible for the search aspect of genetic algorithms, even though mutation operator is also used for this purpose sparingly. Mutation operator changes a 1 to a 0 and vice versa with a small mutation probability, Pm. The need for mutation is to keep diversity in the population. For example, if a particular position along the string length all strings in the population have a value 0, and a 1 is needed in that position to obtain the optimum then neither reproduction nor crossover operator described above will be able to create a 1 in that position. The inclusion of mutation introduces some probability of turning that 0 into a 1. Furthermore, for local improvement of a solution, mutation may be found useful.

These three operators are simple and straight forward. Reproduction operator selects good strings and crossover operator recombines good substrings from two good strings together to hopefully form a better substring. Mutation operator alters a string locally to hopefully create a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be emphasized.

2.4 GA PARAMETERS

The building block hypothesis give an intuitive and qualitative reasoning to what might cause GAs to work. But it tells nothing about for what values of various GA parameters would work. In this subsection, we present some guidelines to determine values for some GA parameters.

The choice of the string length is the first decision to be made. The string length is usually chosen depending on the accuracy needed in the solution. For example, if binary strings of length 1 are used, then the search space would contain 21 strings. The minimum accuracy in the solution that may be expected using GAs would then be of order.

Apparently, the building block hypothesis may seem to suggest that the problems that can be solved successfully using GAs must be linearly separable in terms of clusters of bits. But this is not the case. The above hypothesis can also be applied in problems having higher order nonlinearities with one requirement that either all competing building blocks are supplied in the initial population (by means of biased initial population) or by making the initial random large so that high order scheme competitions can take place in the population. Other important aspect is that even if the building blocks are supplied blocks. Messy GAs were developed to supply building blocks without stray bits and found to be successful in solving many difficult problems.

The other important issue is the balance between the exploitation and exploration aspects of GA operators. Reproduction is responsible for exploring the current population by making many duplicates of good strings and crossover and mutation are responsible for exploring a set of good strings for better strings. GA's success is, therefore, depends on a nice balance between the two. If too many copies of the good strings are allocated in the mating pool then the diversity of the mating pool reduces which in turn reduces the extent of search that can be accomplished using crossover and mutation operators. Even though it has been discussed earlier that this aspect of GAs provide flexibility in their design, this may cause

some potential problem if GA operators are not properly designed to have a proper balance between the two. Recently, a control map is available for values of the selection pressure, S (the number of copies allocated to the best string in the population) versus the crossover probability, (the extent of search) for bit wise linear problems using a computational model that equates the degree of characteristic time of convergence of the selection and the crossover operators alone.

3.1 The optimization procedure using Genetic algorithm

The genetic search process used here in is outlined below :

Step1: Generate a random initial population of chromosomes of size p.

Step 2: decode all chromosomes and evaluate the objective function of their corresponding candidate solutions.

Step 3: if the elitism policy is employed , insert the best chromosomes in to the new generation pool.

Step 4. choose a pair of parent chromosomes from the current population without replacement , apply the crossover and mutation operators to yield a pair Of new chromosomes.

Step 5. Insert the new chromosomes into the new population . If the new population is smaller than p, return to step4.

Step 6. If the pre- specified stopping criterion has been met, then stop the search Process. Select and decode the overall best chromosome to be the final solution. Otherwise, proceed to the next generation and replace the population with the new one, and return to step 2.

3.2. The optimization procedure using SA

Step 1: Choose an initial point x1, a termination criteria Set T a sufficiently high value, number of iterations to be performed at a particular temperature n, and set t =0.

Step 2: Calculate a neighbouring point x2. Usually, a random point in the neighbourhood is created.

Step 3: If $\Delta E = E(x2)-E(x1) < 0$, set t = t+1; Else create a random number 'r' in the range (0,1). If r exp (-ΔE/T) set t = t+1;Else go to step 2.

Step 4: If x2-x1 < and T is small, Terminate. Else if (t mod n) = 0 then lower T according to a cooling schedule. Go to step 2;

Else go to step 2.

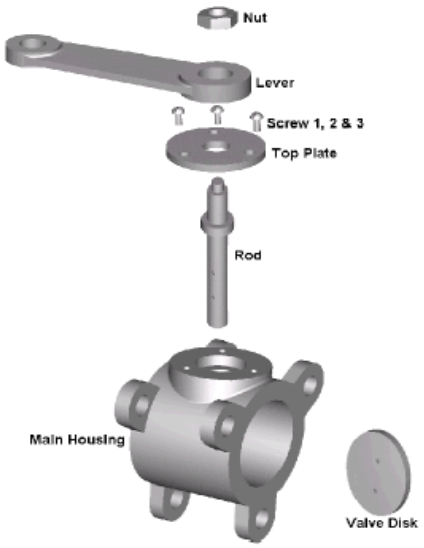

Figure 1 : Disassembly Sequence

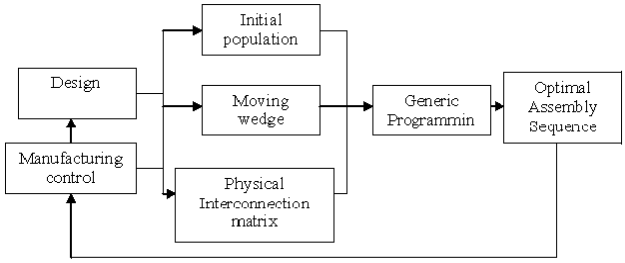

Figure 2: Disassembly Sequence Planner

4. Conclusion

In this paper, optimization techniques for handling of mechanical components in disassembly sequence planning presented. It is an efficient approach of dealing with mechanical components interactively during disassembly process. Optimal solution using GA and SA for Valve Assembly is :1→5→7→2→6→4→9→3→8→1. Path which is used to find the optimal soln is:A→E→G→B→F→D→I→C→H→A. Future work will focus on the application of this methodology to complex products with a large number of parts. We are currently working on a methodology to identify the best way of performing disassembly for the products with large number of parts so as to satisfy some optimization criteria, e.g., minimization of the cost of material.

References

[1] Gupta, S.M and K.Taleb, "Scheduling disassembly," *International journal of Production Research*, Vol.32,No.8,pp.1857-866,1994.

[2] F. Bonneville, C Perrard, J. M. Henrioud. A genetic algorithm to generate and evaluate Disassembly plans. *Proc. of IEEE Symposium on Emerging Technologies and Factory Automation,* Paris, 1995, pp. 231-239.

[3] K.Taleb and S.M Gupta,"Environmental concerns and recycling/disassembly efforts in the electronic industry," *Journal of Electronic Manufacturing*, vol.32,No.4,pp.949-961,1997.

[4] A. Mahanti and A. Bagchi. AND/OR Graph Heuristic Search Methods. *Journal of the Association for Computing Machinery*, Vol. 32, No. I, January 1985, pp. 28-5 I.

[5] Homem de Mello, L.S. and A.C. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Trans. on Robotics and Automat*. Vol. 7, No. 2, 1991, pp.228-240.

[6] Luiz. S. Homem de Mello and A. C. Sanderson. Planning Repair Sequences Using the And/Or Graph Representation of Assembly Plans. *Proceedings of IEEE Int. Conference Robotics Automat, April* 1988, pp.1861-1862.

[7] Luiz.S.Homem de Mello and A.C.Sanderson. Representations of Mechanical Assembly Sequences *IEEE Transactions on Robotics and Automation* ,Vol 7, No. 2, April 1991, pp.211-227.

[8] N.Boneschanscher, Delft University of Technology. Task Assignment for a Small Batch Flexible Assembly Cell Incorporating Multiple Robots. *IEEE* 1990, pp.746-750.

[9] Xining Li and Wei Fan. An Object-Oriented And/Or Graph Inference Engine. *IEEE* 1993, pp. 615- 618.

[10] Michael H. Goldwasser and Rajeev Motwani. Complexity Measures for Disassembly Sequences. *International Journal of Computational Geometry & Applications* Vol. 9, Nos. 4 & 5 (1999) 371-417.

[11] S. Balakrishnan, N. Popplewell, M. Thomlinson. Intelligent Robotic Assembly. *International Journal of Computers and Industrial Engineering*. 38(2000) pp. 467-478.

About the authors

E. Raj Kumar is working as Assistant Professor (Senior) Design division, School of Mechanical and Building Science, VIT UNIVERSITY, Vellore. Tamil nadu, INDIA. He has received B.E. Mechanical Engineering, M.E. CAD/CAM from Anna University, and currently pursuing Ph.D at VIT UNIVERSITY. His main research interest includes CAD/CAM, Optimization Techniques, Virtual Reality and Application of Evolutionary Algorithms in Engineering Applications.