International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-2, Issue-1, Jan 2014

Application of Genetic Algorithms to Solve Modified Load Distribution Problem

Shriranga Kulkarni, Sathasivaiyer Nadarajasarma, George Gao Yong, Wang Yingwei

***Abstract*— As we all know, Cargo services now-a-days have been playing a really important role in our day-to-day operations of business. Proper load weight distribution could allow the transport to operate at peak efficiency which costs less fuel consumption and less maintenance expenses and it could also make the transport to be balanced during its travelling. So, Load Distribution on a container such as a ship/air cargo/or even trucks, has become a very interesting study to many researchers.**

In the standard load distribution problem, there are 64 identical size containers to be arranged in dimension 4x4x4. We define level 1 as the set of containers that lay in the bottom of the ship and level 4 as the set of containers that lay on top. We specify 16 groups each of which contains 4 containers of the same location but with 4 different levels. The weight of each container is given.

The objective is to apply Genetic Algorithms (G.A) concepts and find a way to arrange these containers so that the weight is evenly distributed among 16 groups and satisfy the problem constraint that in each group the lighter packages always lay on top of heavier packages.

This objective is obtained by minimizing the following function:

F = (w1 – AV) 2 + (w2 – AV) 2 + … + (w16 – AV) 2

Where wi denotes average weight of stack i

AV denotes the average weight of the 64 packages.

The objective of this research study is to extend and generalize the above standard Load Distribution Problem and apply genetic algorithm techniques for finding the optimal solution.

***Index Terms*—Fitness Function, Genetic Algorithms, Load Distribution Problem, Mutation**

I. INTRODUCTION

The extensions to the load distributor problem are defined as follows:

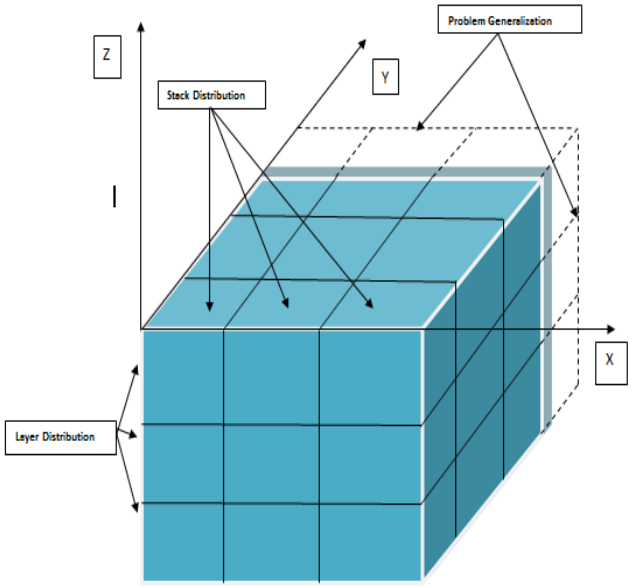
Problem generalization: [container vs. package extension], In the original problem, the number of packages was fixed to 64. As part of extension, now the number of packages can be selected by the user. This is problem generalization (as part of problem extension). So now, the number of packages can be selected as X * Y * Z (denoting Container Length * Breadth * Height).

*Assumption***:** the size/shape of the packages is uniform.

Manuscript received Jan 09, 2014.

Shriranga Kulkarni, M.tech - Knowledge Engineering, Institute of System Sciences, National University of Singapore, Singapore, +65 84823354.

Fitness function/Constraint modeling: In this part of problem extension, fitness functions are applied in 2 areas. The 2 areas are stack-based distribution and layered distribution. As shown in the following diagram, the packages are arranged in a 3-D co-ordinate system.



The Fitness functions consist of both stack based and layer based distribution. This serves as the extension to the given problem. As part of extension, for analysis purposes, there are 4 representations of fitness functions being used.

Choice of Different Selection methods: Currently there are 3 major choices of selection methods available. We can select different selection methods and compare/analyze the results obtained.

Elite or *elitist selection* / Rank Selection / Roulette wheel (as Wheel of Fortune)

Adaptive GA: To prevent premature convergence, the mutation rate and crossover rates can be manipulated at difference stages. We can provide varying values of mutation rate and crossover rate at different intervals of time (during the load convergence) and observe the convergence distribution over the number of iterations (generations).

II. Design and Knowledge Representation

Terminologies: Following are the problem modeling units to represent the problem scenario and problem resolution in Genetic Algorithm problem modeling paradigm.

Stack Distribution: It is the horizontal uniformity of all the stacks in the container.

Layer Distribution: It is the vertical uniformity of all the layers in the container.

Unloading Order: It is an order of integers from 1-K assigned to each package which indicates the relative unloading sequence of package inside a stack. The smaller number indicates a higher priority of unloading.

Package: Each package is uniquely identified by an integer ID. Each package has the properties of weight, unloading order as well as its geometrical location on the container.

Knowledge Representation – Problem Representation based on Genetic Algorithms

Problem Data: The Package Id's, Weight of packages, the number of packages, the length, breadth, height of the container.

Problem Solution: Package ID, Location of the Packages (in a 3-D space), Weight of each package in the container.

Fitness Function: There are 3 components which constitute the overall fitness function being used in implementing the GA techniques for the Load Distributor System.

Stack Based Distribution (SBD): In the 3-D Block diagram above, the distribution of packages along the Z-axis (i.e., Horizontal distribution of packages).

Weight of stack-based distribution (WSBD): This is the input parameter (over the number of generations) which decides the degree of stack based distribution (weight) applied over number of generations.

Layer-based distribution (LBD): In the 3-D Block diagram above, the distribution of packages along the X-axis(i.e., Vertical distribution of packages).

Weight of layer-based distribution (WLBD) : This is the input parameter selected by the user (over the number of generations) which decides the degree of layer based distribution (weight) applied over number of generations.

Unloading disorder incurred penalty (UDIP): The unloading dis-order parameter is basically the unloading order of packages. Each package has an unloading rank order, the unloading of the packages; occur based on the ascending order of unloading rank.

Weight of unloading disorder penalty (WUDP): The input parameter, which signifies, the degree up to which, the unloading dis-order penalty function influences the overall distribution of packages (over the number of iterations).

All the above factors are used to calculate the overall fitness function and following is the mathematical representation of the fitness function.

*Fitness Function = (SBD * WSBD) + (LBD * WLBD) - (UDIP * WUDP)*

Both the Stack and Layer-based distributions are applied using 4 different mathematical calculations:

1. Sum of Delta (∑|Xi-Xa|)
2. Logarithm of Sum of Delta (log(∑|Xi-Xa|))
3. Sum of Square of Delta(∑(|Xi-Xa|)2)
4. Standard Deviation(Sqrt((∑(|Xi-Xa|)2)/N-1))

The constraint of ordered stack is silenced by repairing the order from raw chromosome piece. The unloading disorder is implemented as hard constraint as penalty.

III. Results and Analysis

For the purpose of this experiment, the Elite selection method is assumed,
Set the no. of iterations to 2000,
Population size of 100,
A crossover rate of 60% and
A mutation rate of 10%.
The screenshot below shows the results of the genetic algorithm optimization. The 4 line graphs at the center represent the fitness value of the best chromosome of a generation versus the no. of generations. The 4 bar graphs to the right show the final distribution of the stack weights. The length of the bar represents the weight of a package, so the longer the bar the heavier the package.

The results show that using any of the 4 fitness functions; the best chromosome produces stacks that are uniformly distributed. If you look at the diagram above, the fitness value is much improved from the original value.
(1) Taking the example of fitness function one (Sum of Delta), the initial sum of the differences between the individual stack weights and the average stack weight is about **6.394**.
After optimization with Genetic Algorithms, the sum of differences drops to **0.1352** which is a tremendous improvement on the initial sum. This is what we are trying to achieve as we want differences of weight between the stacks to be as low as possible.
(2)We also observe that regardless of the fitness function, all final stack weights are more uniformly distributed than the

initial distribution, which goes to show the effectiveness of Genetic Algorithm.

(3)Another observation that we can infer from the results is that genetic algorithm allows us to obtain an optimal solution at an efficient pace. The expectations were that the solution should get better as the number of generations increase and we can observe from the results above that this is the case.

(4)We can also notice that with the default **crossover rate** set at 0.6 and **mutation rate** of 0.1, the solutions converge to a maximum optimal fitness value within 2000 generations. Comparing that to an exhaustive calculation of all possible permutations which would take a very long time, the implementation of genetic algorithm in solving this problem is yielding very positive results.

(5) The decision to use 4 different formulae as the fitness function is also giving us some insights about Genetic Algorithm. We can say that using different representation of the fitness functions such as using sum of differences or standard deviation or the log of differences is giving us different rates of convergence as well as different optimal solutions. While we are unable to compare the absolute optimal value of the 4 fitness functions as they are calculated differently, it is interesting to note that the choice of our fitness function actually has a vital impact to distinguish good and relatively poor chromosomes to obtain the optimal solution.

(6) One of the difficulties or problems faced during this implementation is implementing the hard constraint of having no heavier package on top of a lighter package within a stack. To rectify this problem, a **sorting repair algorithm** was used at the end to redistribute the package to the right position in the stack. The results verify that , the sorting repair method is working as you can see that there are no longer bars above shorter bars in each of the 4 diagram above. As a gain, we avoid the adverse effects of prematurely rejecting good chromosomes that do not meet the criteria should a hard constraint be applied instead.

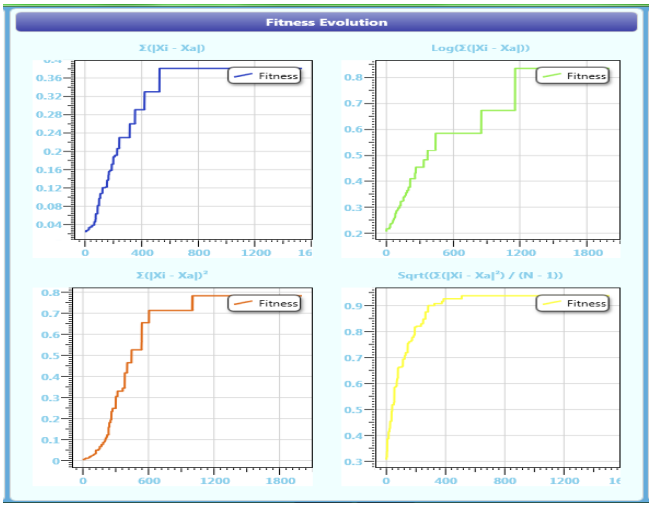
IV. ADAPTATION

To prevent premature convergence, manipulation of the mutation rate and crossover rate was done at difference stages. The user can provide varying values of mutation rate and crossover rate at different intervals of time (during the load convergence) and observe the convergence distribution over the number of iterations (generations).

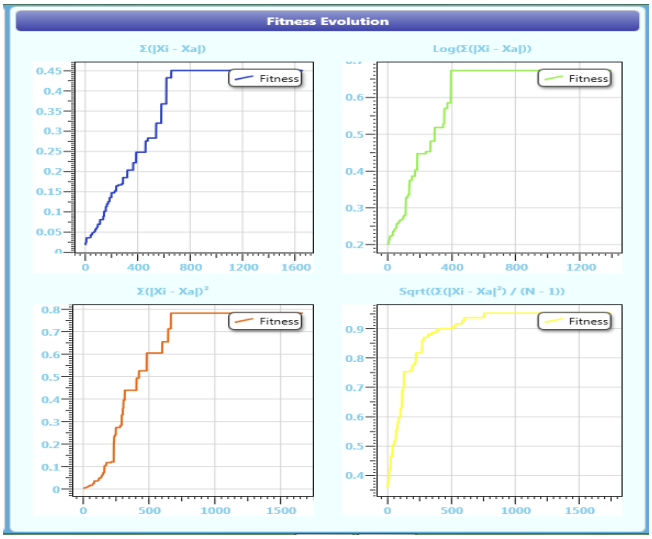
Please find below the different cases for illustrating the adaptation.

GA Stage	Parameters	Case 1	Case 2	Case 3
Initialization	**Mutation Rate**	0.1	0.1	0.1
	Cross Over Rate	0.6	0.6	0.85
Adapted at Pre-converge	**Mutation Rate**	0.1	0.25	0.1
	Crossover Rate	0.6	0.3	0.85
Generations before Converging	**Σ(|Xi - Xa|)**	~800	~700	~500
	Log(Σ(|Xi - Xa|))	~400	~400	~600
	Σ(|Xi - Xa|)²	~600	~600	~500
	Standard Deviation	~600	~700	~700
Optimum of Fitness	**Σ(|Xi - Xa|)**	0.38	0.45	0.33
	Log(Σ(|Xi - Xa|))	0.84	0.67	0.58
	Σ(|Xi - Xa|)²	0.78	0.78	0.86
	Standard Deviation	0.94	0.96	0.96

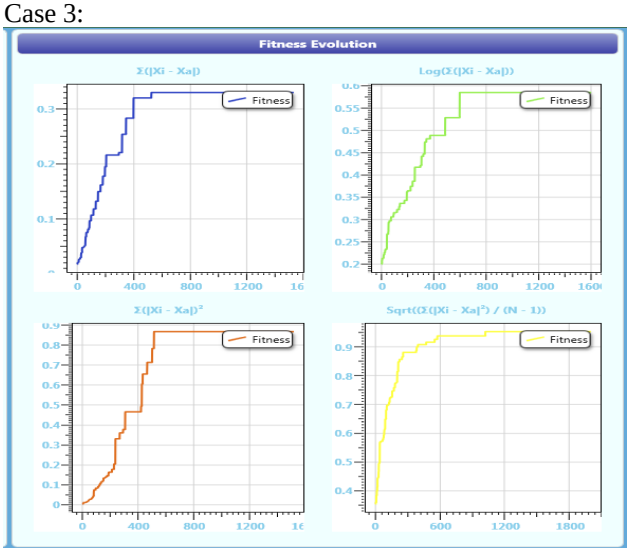
Case 1:



Case 2:



Case 3:



We could come up with the following findings as per the analysis/observations of above results:

1. Higher crossover rate at the beginning helps quick convergence but the fitness value may be stuck with a local optimum.

2. Medium crossover rate at the beginning tends to take more generations to converge.

3. Decreasing the crossover rate and increasing the mutation rate at the pre-converge stage tends to produce fitness optimum.

V. FURTHER EXTENSION

- Automate Adaptation: Based on the progress of evolvement, the genetic algorithm should have the ability to change the Mutation rate and crossover rate in order to maximize the fitness value instead of manual adjustment.

- Optimize the weight of the components in the fitness calculation.

- Currently the weights are hard coded based on the heuristic. In the future it can be fine-tuned by using other AI technologies such as Neural Network.

- Extend the system to handle packages of different shape and size thus utilize the space efficiently

REFERENCES

[1] Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430[1]W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.
[2] Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature:
[3] An Introduction to Genetic Algorithms (Complex Adaptive Systems) - Melanie Mitchell
[4] Genetic Algorithms - Goldberg
[5] http://en.wikipedia.org/wiki/Genetic_algorithm
[6] http://www.geneticprogramming.com/ga/index.htm
[7] http://ai.iit.nrc.ca/subjects/Evolutionary.html
[8] http://www.aforgenet.com/framework/features/genetic_algorithms.html

Shriranga Kulkarni is currently pursuing his part time Master's degree in Knowledge Engineering at Institute of System Sciences, National University of Singapore. Shriranga is a member of Singapore Computer Society, IEEE Singapore chapter and also a member of association of statisticians and historians of cricket, England.

Sathasivaiyer Nadarajasarma is currently pursuing his part time Master's degree in Knowledge Engineering at Institute of System Sciences, National University of Singapore.

George Gao Yong is currently pursuing his part time Master's degree in Knowledge Engineering at Institute of System Sciences, National University of Singapore.

Wang Yingwei is currently pursuing his part time Master's degree in Knowledge Engineering at Institute of System Sciences, National University of Singapore.