*Jurnal Teknologi*, 38(A) Jun. 2003: 61–74
© Universiti Teknologi Malaysia

# CONTAINER STACKING AND RETRIEVAL PROTOTYPE SIMULATION USING GENETIC ALGORITHMS

AZIZI AB. AZIZ[1] & AZIZI ZAKARIA[2]

**Abstract.** Container stacking problem is an NP (Non-Polynomial)-Hard problem. This type of problem requires robust and adaptive algorithms, such as the Constraints Programming technique. Rule-based programming is ineffective due to the complexity of container stacking problem. The container stacking processes are restricted to certain heuristic rules, which is difficult to be coded in algorithmic ways. Improper stacking will cause non-economical container forklift movement. As a result it will increase the operational costs. This paper proposed a genetic algorithms technique to solve container-stacking problem, where a prototype has been developed. The scope of the problem is based on Kontena Nasional (KONNAS Malaysia) case study, which is restricted for one-year operational basis. The container stacking mechanism is operated under several Genetic Algorithms operators such as Selection, Mutation and Crossover. The prototype is coded using C++, ORACLE 7.0 as database and runs under UNIX platform. The average optimal stacking result obtained ranged between 78-83 percent. This prototype was developed at Artificial Intelligence System Development Laboratory (AISDEL), Sirim Berhad.

*Keywords*: Genetic algorithms, constraint programming, container stacking, heuristic rules, port management system.

## 1.0 INTRODUCTION

Container Stacking and Retrieval System (CSRS) is a prototype system that recommends the optimum stacking and allocation for a given number of containers. The systems uses Genetic Algorithm technique, a constraint satisfactory solution that generates the appropriate container location in response to a given constraint. The allocation mechanism will search a suitable container placing, while the stacking engine will generate a suitable stack level according to the optimization requirements [1]. The user is required to provide relevant containers data as input. This system addresses some major logistic management in Port Management System (PMS) Kontena National.

## 2.0 PROBLEM DEFINITION

The three key elements of the logistic chain at container terminal are the quay cranes, the intra-terminal transport and the container stack. The scope of this paper is on the

[1&2]Artificial Intelligence Special Interest Group, School of Information Technology, Universiti Utara Malaysia, 06010 Sintok, Kedah. E-mail: azizi_ab_aziz@hotmail.com

container stack, the place where containers are stored prior to further transport. A container stack consists of an area where containers can be placed, and one or more cranes or forklift to handle the containers. Containers must be stacked in such away that the stacking capacity is maximised and the retrieval task is minimised [1,2].

Stack capacity is basically the product of length, width and stacking height (expressed as Tone Equivalent Unit/TEU). The real capacity is influenced by different container sizes (20, 40, 45 ft and off-standard). For this stacking simulation purposes, only twenty footer and forty footer containers were used. The main problem is the retrieval cost. This cost is incurred when the forklift makes unnecessary movement that can be eliminated if the containers location resides accordingly. Therefore, proper container stacking is needed to overcome this problem [1,3].

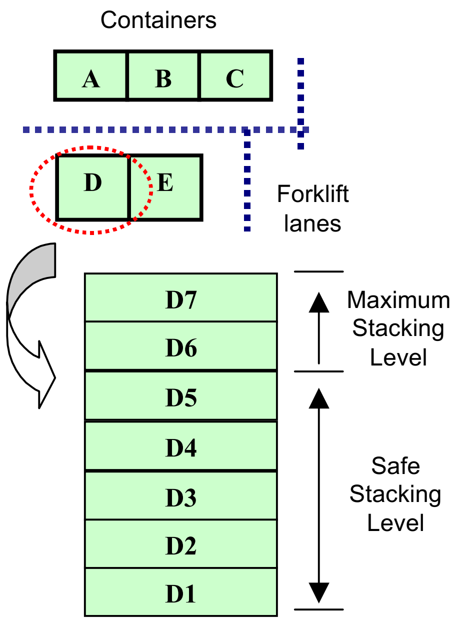

**Figure 1** General layout for container cluster and stack

Figure 1 depicts the general layout of containers placement. The containers A, B, C, D and E represent each respective containers cluster. For example, container cluster D contains stack of containers D1, D2,...,D7. The safety stacking level at Kontena National is restricted to five containers for each cluster. For certain cases, the maximum stacking is expanded up to seven containers per location. These containers are stacked on side-by-side basis; not exceeding four groups in each location clusters.

## 2.1 Stacking Method and Strategy

A distinction is made between stacking method and stacking strategy. The stacking method relates to the choice of a stacking lane, while the stacking strategy relates to the choice of position within stacking lane. Both choices are made separately, and not necessarily at the same time. Firstly, the stacking method determines the stacking lane, and only when the container is about to be moved inbound, the stack position is chosen [2].

Typically, there are two methods to determine stack lane [4]. First, the Random Assignment, and second, the Dedicated Stacking Lane. Random Assignment is a simplest way to choose stacking lane. The only criterion is the availability of at least one stack position (inbound), or at least *x* positions inbound (where *x* is the maximum stacking height). The latter is chosen because extra positions are needed in case it is necessary to reach the lowest container of a stack pile. Meanwhile, Dedicated Stacking Lanes is the assignment of stacking lanes to the crane (especially Quay Cranes). Each crane is serviced by one or more pre-determined stacking cranes. This method requires that the load plan must be known at the moment stacking starts. But, in practice, this rule is not used due to this limitation [2,4].

There are four main stacking strategies presented [2,4]. These strategies are:

- *Random*: uses no information about container or load plan. A random position is drawn, until a position is found where container pile has not reached its maximum heights.
- *Levelling*: uses no information about container and load plan. The stack is filled layer by layer, therefore the maximum actual stack height will be minimised.
- *Closest Position*: also uses no information about container or load plan. The closest position in which the pile, which is not maximum is chosen.
- *Maximum Remaining Stack Capacity*: This strategy needs information on the load category of a container. Containers with no information are placed in a separate part of the stack.

Kontena Nasional, used the random stacking method. For stacking strategy, the levelling technique is chosen. Other methods are not applicable at Kontena Nasional because of main constraint in quay cranes assignment. Furthermore, Kontena Nasional uses only forklift to lift and to move the containers for retrieval and re-stacking purposes. Due to the limitation of these techniques, a genetic algorithm is implemented in our prototype.

## 2.2 GENETIC ALGORITHMS

The idea of applying the biological principle of natural evolution to artificial systems was introduced three decades ago with an impressive growth. Genetic algorithms (GA) were proposed by John Holland where he has written a book "*Adaptation in Natural and Artificial Systems*" published in 1975. GA has been successfully applied to wide range of problems, such as optimisation, automatic programming, machine learning and social systems [5].

Basically, GA is an iterative, directive and random search procedure that consists of constant-size population of individuals, each one represented by a finite string of symbols known as *genome* (encoding possible solution in a given problem space)[6]. Solution to a problem solved by genetic algorithms is revolving in nature. The algorithm started with a set of solutions, which is represented as *chromosomes*. These chromosomes create a set of population. The generation of possible population can produce new population that can improve the old one (through gene formation – *crossover*, *mutation* and *selection*). Figure 2 depicts the flowchart of genetic algorithms process.

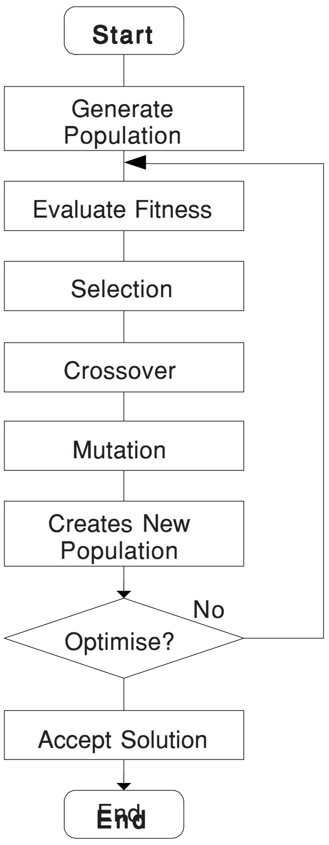

**Figure 2** The flowchart of GA

A new solutions (*offerings*) are selected according to their fitness. This process is repeated until some stopping conditions are satisfied. Generally, genetic algorithm is applied to a search space, which is normally too large to be exhaustively searched.

## 3.0 EXPERIMENTAL PARAMETERS AND DESIGN

### 3.1 Data Structures (Chromosomes Presentation)

This prototype system was developed using Hitachi C++ compiler, which runs under UNIX operating system. ORACLE 7 serves as a database mechanism that holds data for the system (refers Figure 3).

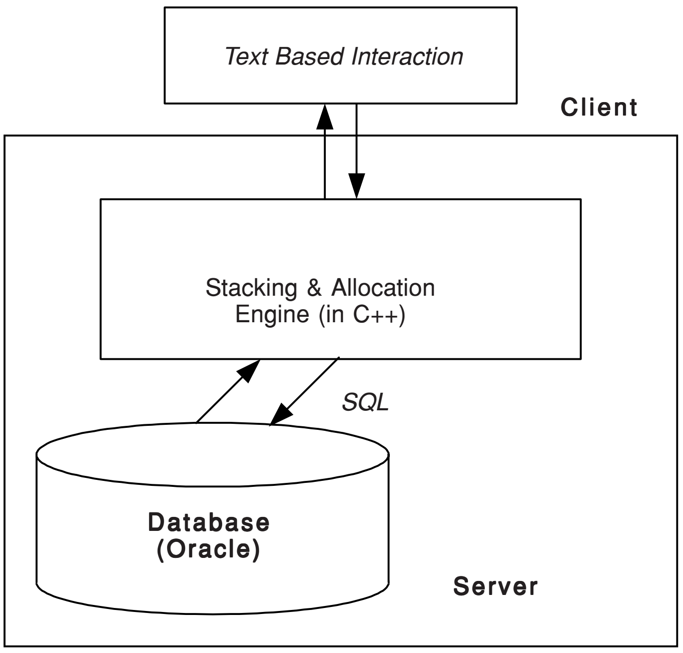

**Figure 3** The system architecture

The *permutation encoding* is used in creating chromosomes structures [3]. The important attributes in constructing data chromosomes are[2,4]:

- Container Identification
- Date In
- Types of Containers
- Stacking Priority
- Lane Priority
- Predicted Date Out

Table 1 shows the encoding representation for each attribute.

**Table 1** Representation of attributes

| **Attributes** | **Representation** |
|---|---|
| Container ID (Unique) | String (eg: A01) |
| Date In | Integer (eg: 001) |
| Types of Container | String (eg: E) |
| Stacking Priority | Integer (eg: 6) |
| Lane Priority | String (eg: W) |
| Predicted Date Out | Integer (eg:01) |

Permutation encoding represents the chromosome using sequence or order of number or alphanumeric value [7]. This representation is widely applied in Travelling Salesman Problem (TSP) problem. Graphically, the chromosome structure for CSRS is represented in Figure 4.

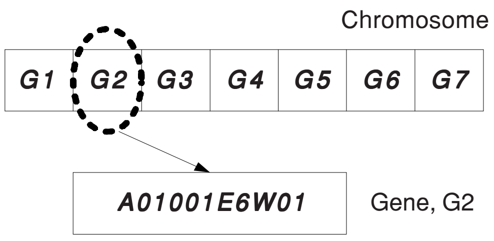

**Figure 4** The example of chromosome representation

Chromosome structures are noted as G1,...,G7. Each of this structure contains gene, which carry the information of domain problem. This gene was formed through the combination of respective attributes depicted in Table 1.

## 3.2 Pseudo Code For Container Stacking and Allocation Strategy

A skeleton of the evolutionary method used in CSRS is shown in Figure 5.

```
Start:

Input: Container Information from database
Output: Proposed Container Stacking &
        Allocation.
Set generation ← 0
  Initialise crossover and mutation parameters
  Load container information into container array
  Generate chromosomes from container array
  Initialise population

     For each chromosomes structures
       Calculate initial fitness
       Stored initial fitness
     End for

While Not Termination Condition = True
  Assign random number to Roulette Wheel
  Evaluate the fitness value
  Select chromosome from initial population
  Perform Single Point Crossover
  Perform Mutation (Order Changing)
  Evaluate New Generation
    generation ← generation + 1
  End While
End:
```

**Figure 5** Pseudo code for GA implementation

## 3.3 Genetic Algorithms Operator For CSRS

### 3.3.1 *Fitness Function*

The initial population was derived from the information given by Kontena Nasional. This population was then structured into chromosomes representation. In this case, 4704 genes were produced during initial population phase. The fitness calculation module evaluates the effectiveness and suitability of chromosome and genes sequence. For CSRS implementation, three types of fitness evaluation were used.

- *Fitness Value*

$$f(x) = optimum\ value - \sum x, \tag{1}$$

where $x$ refers to penalty value assigned. Optimum value is the value assigned if a set of stacked and allocated container is ideal. Through study, the value assigned is 50,000.

- *Total Fitness*
  Calculate the accumulate value of individual fitness value. Calculated as

$$tf = \sum f(x) \qquad (2)$$

- *Ratio Fitness*
  Evaluates corresponding individual fitness over population. Calculated as depicted in Eq.(3).

$$s_f(x) = \frac{f(x)}{\sum_{j=1}^{n} f\left(d_{ij}\right)} \qquad (3)$$

where $n$ refers to container population size, $j$ refers to iteration size and $d_{ij}$ refers to fitness for population.

The penalty value is assigned after several evaluation based on constraints given. If the chromosome or genes structures violate the given constraint, some penalty value will be given [8,9]. Thus, it decreases the fitness value for respective chromosomes. The major constraints that contribute to penalty assignment are:

- Missing genes (containers).
- Distance between genes.
- Improper container sequence.
- Operational cost for possible movement during retrieval.
- Similar Genes Redundancy.

### 3.3.2 *Selection Process*

The selection (reproduction operator) selects chromosomes according to their fitness function values. In this process, the well-fitted individuals have more chances to be selected. CSRS used *canonical type* selection process. Using this process the parents' $p$ produce offspring $c$ using crossover. Each of the $c$ children is then assigned a fitness value, depending on its quality considering the problem specific environment [5]. Later *Roulette Wheel Selection* technique is chosen to select potential chromosomes.

*Roulette Wheel* method selects the individual by mean of roulette style [5]. This method requires fitness value $f_i$ to be positive ($f_i > 0$) and each chromosome occupies

$f_i$ out of a total size $\sum_{j=1}^{n} f_i$, where $i$ refers to the chromosome, $j$ refers to iteration size and *n* refers to the population size. Therefore, the probability $p(i)$ of chromosome *i* to be selected is given by

$$p(i) = \frac{f_i}{\sum_{j=1}^{n} f_i} \tag{4}$$

A uniformly distributed random number *R* is generated, $R \in U[0,1]$. If *R* is between cumulative probabilities of the *i*th and $(i+1)$th chromosomes, *i* is selected for the next generation. This is repeated for the required number of *N* replacements for the next steps [3].

### 3.3.3 *Crossover*

Selection alone cannot introduce any new individuals into the population. These are generated by genetically inspired operators. Crossover is performed with probability of $P_{crossover}$ (crossover rate) between two selected chromosomes (parents), by exchanging part of their genomes (encoding) to form new individuals (offspring). In its simplest form, sub strings are exchanged after a randomly selected crossover point. This operator tends to enable the evolutionary process to move towards promising regions of the search space state[10].

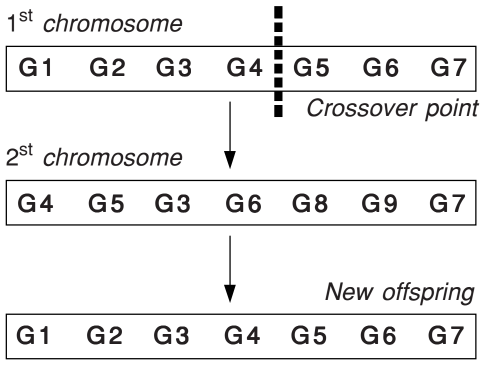

**Figure 6** Single point crossover for permutation encoding

The CSRS allows single-point crossover technique. This operator will select one crossover point at a specified chromosome location. At this point, the permutation encoding is copied from the first chromosome and the second chromosome is scanned. If the gene from second chromosome is not yet in that chromosome structure, it will be added into the first chromosome. A crossover process is shown in Figure 6.

### 3.3.4 *Mutation*

Mutation is another important operator in genetic algorithms. It is a random change in genetic material of a single individual (chromosome) where it is applied to genes by changing them with a very low probability [5]. Mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space [10].

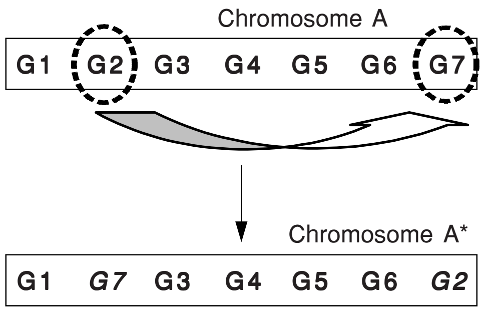

**Figure 7** Single conversion mutation

In CSRS, the mutation probability rate ranges from 0.001 to 0.010. Example of a mutation process is shown in Figure 7.

## 4.0 EXPERIMENTAL RESULTS

In this prototype, the overall results show the accepted stacking condition has achieved up to 85.6 percent optimisation rate. The optimisation rate or Opt, was measured using.

$$\text{Opt}\,(\%) = \frac{\text{correct stacking by GA}}{\text{correct stacking from record}} \times 100 \tag{5}$$

The data consist of 4,704 containers with 672 possible conditions. The initial population size was set to 672 chromosomes and the generations were alternated for 200 times. The generation with best optimisation rate was taken as the final output. Average run time is between 30 and 45 minutes. There are two steps involved in this prototype. The first step is an initial stacking step and the second step is an improvement step. During the improvement step, the initial condition will go through several genetic algorithm operators.

From the experiment, it was shown that some changes in genetic operator will affect certain degree of optimisation performances. The initial mutation rate started with 0.001 and has been increased up to 0.050 [5,9,10]. The crossover rate cutting point ranges from 0.10 to 0.90 [5,9,10]. The random number is generated to produce the crossover rate. Table 2 depicts the optimisation rate over several chosen parameters.

**Table 2** Experiment results for optimisation rate over different parameters

| **Num** | **Optimization Rate (%)** | **Crossover Rate** | **Mutation Rate** |
|---|---|---|---|
| 1 | 68.52 | 0.10 | 0.001 |
| 2 | 71.33 | 0.25 | 0.005 |
| 3 | 73.14 | 0.35 | 0.008 |
| 4 | 79.45 | 0.45 | 0.013 |
| 5 | 83.55 | 0.50 | 0.015 |
| ***6*** | ***85.60*** | ***0.55*** | ***0.020*** |
| 7 | 81.32 | 0.75 | 0.030 |
| 8 | 75.31 | 0.80 | 0.035 |
| 9 | 69.55 | 0.80 | 0.050 |
| 10 | 65.12 | 0.90 | 0.050 |
| 11 | 68.13 | 0.90 | 0.040 |
| 12 | 71.00 | 0.70 | 0.045 |
| 13 | 70.33 | 0.50 | 0.045 |
| 14 | 64.33 | 0.40 | 0.050 |
| 15 | 61.45 | 0.10 | 0.050 |

From Table 2, the highest optimisation rate 85.60% was achieved when the crossover rate is 0.55 and the mutation rate is 0.020. After several attempts, at certain threshold point, any increment of mutation rate will decrease the possibility of getting better optimisation rate.

Experimention on optimisation rates over generated population iteration (generation) was also investigated. During this process, we have chosen five best optimisation rates were chosen with respective parameters. There are two procedures involved:

- To estimate appropriate generation needed.
- To measure optimisation rate over generation produced.

Figure 8 illustrates the optimisation rate over produced generation. From Table 2, Set A refers to parameter in Num 6 , Set B for Num 5, Set C for Num 7, Set D for Num 4 and Set E for num 8.

From the experiment, the optimisation rate will decline or remain unchanged after certain period of reproduction process. It shows that the algorithm has achieved their optimum result [3, 6]. Any attempt to enhance optimisation value will fail since the overall possibility of probed searched space is high [6, 8]. Table 3 shows the optimal generation level for respective parameters.

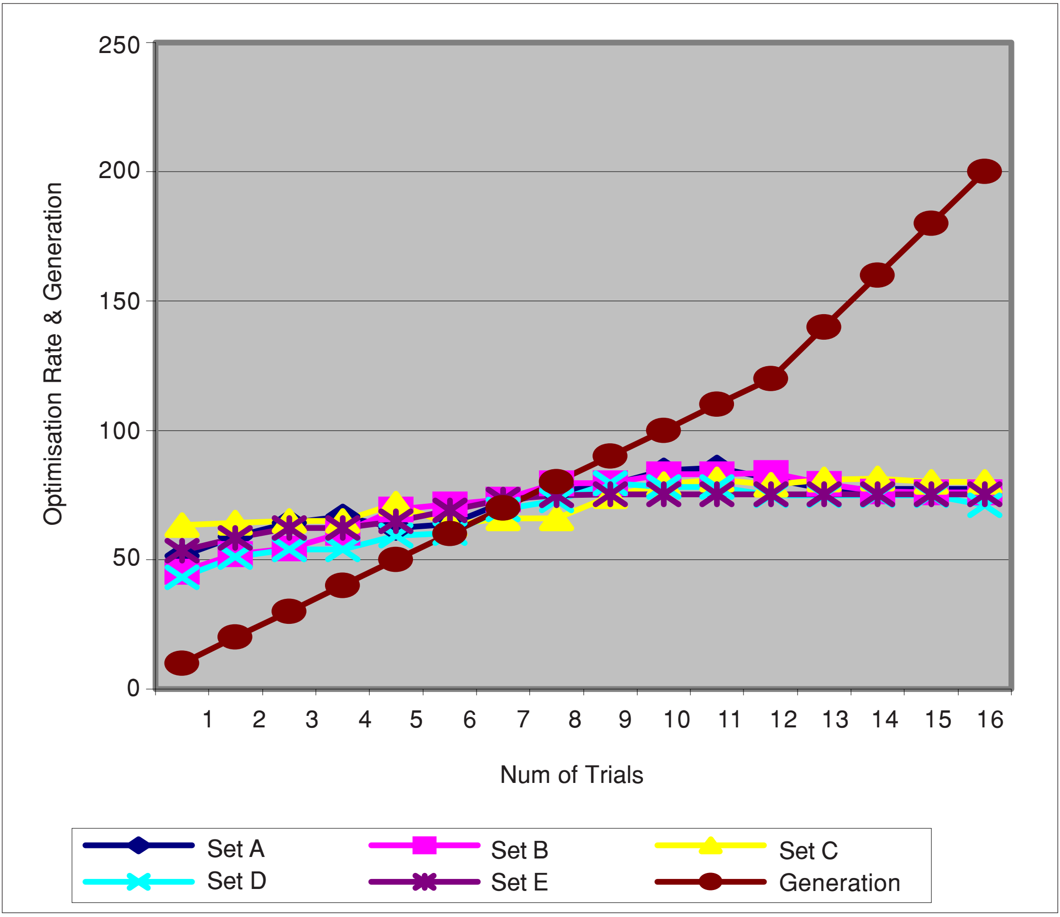

**Figure 8** Comparison between generation reproduction and optimisation rate

**Table 3** Generation level and optimisation rate

| Set | Optimisation Rate% | Generation (Reproduction) |
|---|---|---|
| A | 85.60 | 110 |
| B | 83.55 | 120 |
| C | 81.32 | 160 |
| D | 79.45 | 90 |
| E | 75.31 | 90 |

Based from the experimental results, the final property for highest optimisation rate is depicted in Table 4.

**Table 4** Final result property

| **Properties** | **Final Result** |
|---|---|
| Optimisation Rate | 85.60 % |
| Crossover Rate | 0.55 |
| Mutation Rate | 0.020 |
| Processing Time (Avr) | 35 minutes |
| Crossover Method | Single Point Crossover |
| Mutation Method | Single Conversion |
| Encoding | Permutation Encoding |
| Selection | Roulette Wheel |

## 5.0 CONCLUSIONS

A technique to optimise container stacking and allocation based on the given data has been described. Generally, our prototype (CSRS) shows the capability of genetic algorithms implementation in logistic management. Nature adaptation in genetic algorithms gives a better way to solve container stacking and allocation problem. Since the results were achieved through simulation process, it will provide better analysis in container management. The simulation results will provide further insight in predicting possible container arrangement and movement. Thus, it will allow analysis on forklift flow and tasks. The best optimisation rate achieved is 85.6 percent using Single Point Crossover, Single Conversion Mutation and Canonical-Roulette Wheel Selection. In the future, a combination of general approximation techniques such as neural networks and machine learning can provide a more significant results due to its ability in discerning hidden patterns in huge data warehouse.

## ACKNOWLEDGEMENT

The authors wish to express their sincere thanks to researchers at Artificial Intelligence System Development Laboratory (AISDEL) SIRIM Berhad for their help and fruitful discussion throughout this work.

## REFERENCES

[1] Aziz, A. and A. Zakaria. 2001. Container Stacking and Retrieval System using Evolutionary Strategy. *Persidangan Kebangsaan Penyelidikan dan Pembangunan Institut Pengajian Tinggi Awam*. 811-815.

[2] Mark, B, J. Joseph and J. Ottjes. 2001. A Simulation Model for Integrating Quay Transport and Stacking Policies in Automated Container Terminals. *Proceedings of the 15th European Simulation Multi-conference (ESM 2001).*

[3] Aziz, A. 1999. *Sistem Capaian dan Penyusunan Kontena Menerusi Kaedah Algoritma Genetik*. Industrial Attachment Final Report. SIRIM Berhad,1999.

[4] Duinkerken, M. and J. Evers. 2000. A Simulation Model for Automated Containers Terminals. *Proceedings of the Business and Industry Simulation Symposium*. 134-139.

[5] Koza, J. 1992. *Genetic Programming: On The Programming of Computer by Means of Natural Selection*. Cambridge : MIT Press.

[6] Gen, M. and R. Cheng. 2000. *Genetic Algorithms and Engineering Optimization*. John Willey and Sons.

[7] Khalid, M. 2002. Artificial Intelligence: From Biology to Industry. *Proceedings of the International Conference on Artificial Intelligence in Engineering and Technology*. 1-7.

[8] Lim, A. and Y. Singh. 2002. Function Learning with Numeric Constants via Genetic Programming. *Proceedings of the International Conference on Artificial Intelligence in Engineering and Technology*. 658-662.

[9] Tezuka, M. 1999. Genetic Algorithm to Optimize Production Scheduling. *Artificial Intelligence Application in Industry*. 78-83, 2000.

[10] L. S. Teo, M. Khalid and R.Yusof. 1999. Tuning a Neuro-Fuzzy Controller using Genetic Algorithms. *IEEE Trans on Systems, Man and Cybernetics*. April 1999.