Designing Optimal Binary Search Tree Using Parallel Genetic Algorithms

K. Zamanifar *M. Koorangi*

Department of Computer, Faculty of Engineering, University of Isfahan, Iran

Summary

Evolutionary algorithms (EAs) are modern techniques for searching complex spaces for on optimum [11]. Genetic algorithms (GAs) are developed as random search methods, which have not so sensitivity on primary data of the problems. They can be used in estimation of system parameters in order to obtain the best result. This can be achieved by optimization of an objective function. Genetic programming is a collection of methods for the automatic generation of computer programs that solve carefully specified problems, via the core, but highly abstracted principles of natural selection [12]. In this paper, genetic algorithms and parallel genetic algorithms have been discussed as one of the best solutions for optimization of the systems. Genetic and parallel genetic algorithms have been investigated in parallel programming environment called Multi-Pascal. Then an optimal binary search tree has been selected as a case study for decree sing of searching time. Also a dynamic programming method has been accelerated by using of a parallel genetic algorithm. In this case, by increasing the size of data, speed-up index will be increased.

Key words:

Optimization, Genetic Algorithm, Parallel Genetic Algorithm, Optimal Binary Search Tree

1. Introduction

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Their basic working mechanism is as follows: the algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one [13, 14, 15].

Everything around us is part of some system. Researchers have tried to model it into the system computer. The models were not complex enough to solve interesting problems. Thus the models were not practical [4]. A system is a black box with a set of input parameters. The system developers measure the parameters of each subsystem separately, and exhibit all them as a set of the system's parameters, but ignore the effect of sub-systems on each other and disorders signals. In addition, the parameters should be set so that the system conclude the best. For doing of this matter, it is needed to optimize the output function of the system. It means that we should minimize or maximize it, and consequently increase its performance. The goal of this research is achieving a solution that these values are obtained faster without involving in internal properties of the system. The optimal binary search tree has been considered as a case study. Generally, there are three general methods for optimization and searching of these optimal points [2] : The Calculus Based Searching method, Enumerative Searching method and Random Searching method. The calculus based searching method is divided to two branches : Direct and Indirect. In direct way, the optimal points are obtained by solution of some linear equations or non linear ones. In indirect way, a limited of optimal pointes are obtained, then they are optimized by Hill Climbing methods. In the enumerative searching method, the searching space of the problem is processed and the value of objective function of the system is obtained for each point, and finally optimal points are selected. Dynamic programming method is of these cases. In random searching method, the space of searching problem is searched by random for finding of optimal points. Genetic algorithm is a guided random algorithm [2].

The two first methods aren't cost effective and they don't effect if searching space of the problem is expanded. Parallel algorithms are used to increase the speed and performance of the optimization methods. The genetic algorithms are appropriate for this purpose because of : 1) Independency to primary values of the parameters 2) Independency to system's objective function properties (continuous, derivative, etc.) 3) Searching of greater space of the parameters values. The most important characteristics of these algorithms is parallelism. It causes the increasing of the speed and performance of the system and decrease the system's response time. Sometimes due to existing the several objective functions in the system, using of genetic algorithm will increase the system's speed and will decrease the system's response time.

2. Genetic Algorithms

Genetic algorithm can be viewed as a biological

metaphor of Darwinian evolution [4]. It is a random searching method which creates a new generation of the answers by selecting a collection of answers randomly, and improves them in each stage, until finally it achieves an acceptable answer between these answers. This algorithm have some components[1,2,6,9]. These components are : Chromosome, Genetic population, fitness function, genetic operations, and genetic algorithm parameters. By running of genetic algorithm, some chromosomes from genetic algorithm are selected as parents. Next generation of chromosomes are created by using the operators, and therefore the next genetic population is composed. This is done by Select operator [4,8,10].

Only selection of the parents is not enough for producing of the next generation of chromosomes, but we should search for some methods for returning of the produced chromosomes to the Genetic Population. This is also done by Replacement operator. To doing of this case, after selecting the parents from Current population, they are placed in the Intermediate population. The genetic operation will be done on them until a new population of the chromosomes will be created, then they will be placed in the Next population [4]. Permutation operator is used for recombination [4,6]. The permutation operator is also another operator which will cause innovation in the chromosomes of a genetic population. It also stops monotony in genetic population and stops involving the algorithm in the local minimize or maximize points.

3. Parallel Genetic Algorithms

For the first time, Holland, 1963, recognized the parallel nature of genetic algorithms, and in 1976 Bethke calculate the complexity of doing the Genetic algorithm on parallel machine, but he didn't simulate or implement it. Then in 1981, Grefenstette presented some parallel implementation of genetic algorithms[2].

The way in which GAs can be parallelized depends on the following elements[16]:

- How fitness is evaluated and mutation is applied
- If single or multiple subpopulations (demes) are used
- If multiple populations are used, how individuals are exchanged
- How selection is applied (globally or locally)

There have been some attempts to develop a unified taxonomy GAs which would greatly help addressing this issue[24].

There are several motivations for parallelism of the genetic algorithms. One of them is intending for increasing speed and performance of genetic algorithms using the parallel computers. The other one is able to apply genetic algorithms for solving of greater problems in a reasonable time and make it near to its own biologic structure in the nature. Also parallel genetic algorithms show a high performance for solving the problems with multi-objective functions.

3-1. Classes of parallel Genetic Algorithms

The parallel genetic algorithms are categorized to four classes : Global[4], Coarse-Grained [25], Fine-Grained[26], and Hybrid[4]. A global genetic algorithm considers all the population as a one. The population individuals are evaluated to obtaining their fitness. Also the genetic operations act in parallel. The goal in this class is parallelism of the genetic algorithm. These kinds of algorithms are implemented in two forms : shared memory machines and distributed memory machines. In implementation of the shared memory machines, the individuals of the genetic population will be stored in a common memory, and each processor can access this memory. These processors get some of individuals, and apply the genetic operators on them, and return them to the common memory. Synchronization is necessary between processors in starting of producing each generation. In the implementation of the distributed memory machines, the genetic population is stored in the memory of a processor called Master (or Farmer). This processor sends the individuals of the population to other processors called Workers (or Slaves). The workers evaluate individuals and collect the results. They also produce the next generations by using of genetic operators. This method has two problems : 1) A great time is consumed to evaluating and the master is unemployed. 2) If the master crash, the system will be stop. This model is presented in three forms : Synchronous, Asynchronous and Semi-Synchronous. In the synchronous model, the processors are synchronized in the starting and ending of each generation, therefore the master processor should wait for a slower processor. In asynchronous or semi-synchronous models, the master processor doesn't wait. In here, the master processor selects the individuals of the current population. Therefore the processors will work asynchronously.

The coarse-grained genetic algorithm divides the genetic population to separate sub-populations. The separate genetic algorithm is applied on the each sub-population. The individuals are exchanged between sub-populations in order to optimize the answers at special times. In other words, they migrate between sub-populations. In most of the times, the size of sub-populations will be taken equal. These kinds of algorithms usually are implemented on MIMD computers with distributed memories. Some samples of these machines are such as : *CM-5, NCUBE, Intel's paragon*, and etc.[1]. A point which should be noted is that in this class, the

communication between processors is very lower than the calculated work which each processor do on their own sub-population. A new operator called Migration operator, is presented here. This operator exchanges the individuals between the sub-populations[7]. The following actions is done by this operator :

- Selecting the emigrants: In this stage, the emigrants of each sub-population are selected.
- Sending the emigrants: In this stage, the emigrants of a sub-population are sent to the other one.
- Receiving the emigrants: In this stage, the emigrants are received from a sub-population.
- Merging the emigrants: In this stage, the emigrants are merged in a sub-population.

By this operator, sending and receiving of the individuals can be done in parallel message passing way. In this way, selecting and merging of the emigrants cause a population of the best answers in each sub-population. Migration models are presented in two forms: Island model and Stepping-Stone model. In island model, the individuals are allowed to migrate to each sub-population while in stepping-stone model, the migration limited to the neighborhood sub-populations. In Island model, the individuals have freedom to migrate, but the overhead of communication and delay are too much, while in stepping-stone model, the freedom of migration is limited but the overhead of communication is decreased.

The fine-grained genetic algorithm divides the genetic population into several small sub-population (Deme), and sometimes it behaves with each individual separately. In this algorithm, each one of the demes or individuals can place on a separate processor and each individual can mates with its neighborhoods. These kinds of algorithms also can be implemented on the parallel computers. The first attempt in this field was done by Robertson in 1987 on SIMD computers, and this algorithm was named ASPARAGOS [1, 5]. In these kinds of algorithms, against of the coarse-grained genetic algorithms, the communication between processors is more than the calculation work of each processor. Also using these algorithms prevents from soon dominant of super individuals on population.

The hybrid genetic algorithm is a combination of two previous algorithms. In here, two levels are considered for execution of algorithm which in each level, a class of parallel genetic algorithms is applied. In 1994, Gruau presented the hybrid genetic algorithm for the first time, and used it for Neural Networks [3].

3-2. Parallel population Models

Parallel population models state the following things:

- How a population is divided to different sub-populations?
- How information is exchanged between sub-populations?

These models are divided into three general parts[7] : Global, Regional, Local. In the global model, the population is not structured, the select operation is general, the fitness of each individual is calculated related to all the individuals, and each one of individuals can be selected as a parent for reproduction. In regional model, the population is divided to several sub-populations (Region). The fitness of each individual is calculated related to the individuals of its sub-population, and the parents are selected from that region. In the local model, the population has a neighborhood structure. The fitness of each individual is determined related to its local neighborhood, and parents are selected from the same neighborhood. Table1 shows the summary of related works with these three models [7].

Table 1: *Summary of Related Works with three Models*

Reference	Global Model	Regional Model	Local Model
GREFENSTETTE(1981)	Master-Slave	Network	-
MANDERICK et al.(1989)	R-Algorithm	Coarse Grain	Fine Grain
MACFARLANE et al. (1990)	Farming	Migration	Diffusion Model
GORGES-SCHLEUTER(1992)	Panmixia	Model	Diffusion Model
DORIGO et al. (1993)	-	Migration	Neighborhood
WHITLEY(1993)	Global Pop.	Model	Model
CANTU-PAZ(1995)	Global Par.	Island Model	Cellular GA
		Island Model	Fine Grain
		Coarse Grain	

- They do less functional evaluation for finding the optimized solutions.
- They are able to find several solutions.
- They can be synchronous or asynchronous.
- Their implementations accommodate with parallel architectures.
- They are fault tolerant.
- They are nearer to biological simile of evolution.

4. Designing of Optimal Binary Search Tree using the Parallel Genetic Algorithms

Binary trees and binary search trees are covered in many standard texts such as [17,18,19,20,21,22,23].

- Definition: A binary tree T is the structure defined on a finite set of nodes that either contains no nodes, or is composed of three disjoint sets of nodes: a root node, a binary tree called its left sub-tree, and a binary tree called its right sub-tree [21].
- Definition: A binary search tree (BST) is a binary tree whose nodes are organized according to the binary search tree property: keys in the left sub-

tree are all less than the key at the root; keys in the right sub-tree are all greater than the key at the root; and both sub-trees are themselves BSTs.

Optimal Binary Search Trees are covered in many algorithms texts [17,18,19,21,22,23]. For any set of keys, there are many different binary search trees. The time required to seek a given key can vary from tree to tree depending on the depth of the node where the key is found, or the length of the branch searched if the key is not present. An optimal binary search tree is a binary search tree with minimum expected comparisons for special set of keys and their possibilities. The number of comparisons is called Searching Time.

Suppose key_1, key_2, key_3, … key_n are n keys, P_i is the possibility of key_i and C_i is number of the comparisons for finding of key_i, then optimization of Binary Search Tree is minimizing of the following relation. It calculates the average searching time for n keys in a binary search tree.

$$\sum_{i=1}^{n} C_i P_i$$

$$(\sum_{i=1}^{n} P_i = 1 \quad , \quad 0 \le P_i \le 1 \quad , \quad 1 \le C_i \le n) \qquad (1)$$

4-1. Dynamic Programming for Solving Problems

Suppose tree t_1 is an optimal one for a case that key_1 is in the tree's root, tree t_2 is an optimal one for a case that key_2 is in the tree's root, and tree t_n is an optimal one for a case that key_n is in the tree's root. Then we should search for a k, so that key_k is in the tree's root and searching time the tree for it be minimum. The same work is repeated in left and right sub-trees, until an Optimal Binary Search Tree is formed. This is shown with the following equation :

$$A[i][j] = \min(A[i][k-1] + A[k+1][j] + \sum_{m=i}^{j} P_m) \quad , \quad \forall k = i, i+1, ..., j \quad (i < j)$$

$$A[i][i] = P_i \ , \ A[i][i-1] = 0 \quad , \quad A[j][j+1] = 0 \quad , \qquad (2)$$

$$R[i][j] = k \ , \ R[i][i-1] = 0 \quad , \quad R[j][j+1] = 0$$

This equation calculates the minimum searching time. It means we should find a k so make the time minimized. *A* is the cost function of the problem or in other words is the minimum searching time, and *R* shows the tree's root in each stage.

The execution cost of this algorithm is equal with the cost of filling $2(m + n(n+1)/2) + 2$ memory fields, because without considering the main diameter of matrixes, we should fill n fields in the first line, $n-1$ fields in the second line, …, and 1 field in the last line of each matrix. And if the elements of main diameters are considered, so we can reach to above equation. In addition, we should choose the minimum values between different k s which has cost of $O(n)$ in the worst case. Then in total, the cost of *this* algorithm is $O(n^2) \times O(n) = O(n^3)$. Of course, we can optimize this way, and in result decrease time cost to $O(n^2)$ [11]. The above way was presented in 1959 by Gilbert and was obtained in 1982 by Yaeu [11].

4-2. Genetic and Parallel Genetic Algorithms for solving the Problems

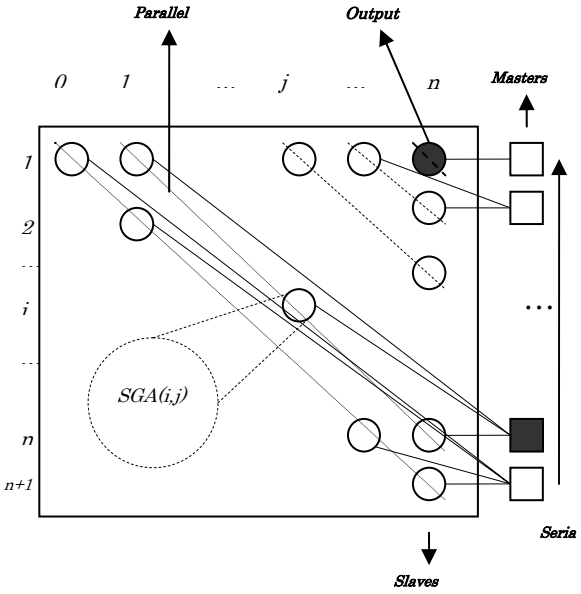
In the genetic algorithm, a collection of possible answers are considered as C_i s, and it is tried to find an optimal answer from them. Each array is considered as an answer (or a chromosome). Each gene of this chromosome, determine value of C for a key. And the internal number of each gene is value of C_i for key_i. More over, we should consider the following case for each gene.

- $1 < C_i < n$
- maximum number of 1s are 1, and maximum number of 2 s are 2, maximum number of 3 s are 4, and in general, maximum number of k s are 2^k (regarding number of leaves in a level of complete binary tree).
- If the number of nodes for one level is m, so the number of next level can not be more than $2m$.

If this procedure is used, then coding of the problem will be difficult, and in each stage of the algorithm, the above conditions should be controlled, that leads to lose the time. On the other hand, in this kind of coding, the solution depends to specifications of the problem. For removing this fault, instead using the objective function directly, we can use another evaluation function, and change the genetic population chromosomes (equation 2).

For this purpose, by using genetic and parallel genetic algorithms, it is tried to make the dynamic programming of this problem faster and better. We use this solution,

because of easy implementing the dynamic programming of this problem in parallel form. And the other hand, the introduced cost function can be used as the fitness function, and the chromosomes of the genetic population can be used as k s. Therefore instead using directly the objective function and obtaining C_i s, we try to obtain suitable k by using of genetic algorithm in each stage of the dynamic programming. We can decrease the algorithm execution cost by parallelism of this algorithm. Fig. 2 and Fig. 3 state a summary of this method.



Input : the number of keys (n) and set of their possibilities (Array p)
Output : the optimal search time (t) and the set of multipliers (Array c)
Parameters :

- genetic population size (popsize)
- length of the chromosome (lchrom)
- maximum of the generation number (maxgen)
- possibility of Crossover operator (pcross)
- possibility of the Mutation operator (pmutate)
- primary value of producing of random number (seed)

Properties :

- type of coding: Binary coding
- type of the Select operator: Roulette wheel
- type of the Crossover operator: Single point
- type of the Mutation operator: Inverting of selected bits
- Condition of terminating of genetic algorithm: maximum number of generations
- Objective function: $a[i,k-1]+a[k+1,j]$
- Fitness function: $1/(a[i,k-1]+a[k+1,j])$
- Model of parallel population: Global
- Parallel architecture : Shared Memory
- Class of Parallel Algorithm : Global (Semi-Synchronous Master-Slaves)

Fig. 2 Properties of Genetic Algorithm for the solution of Optimal Binary Search Tree

```
input(n,p)
input(popsizepercent,maxgen,pcross,pmutate,seed)
  fork forall i:=1 to n do
   fork a[i,i-1]:=0
   fork r[t,i-1]:=0
  fork w[i,i]:=p[i]
    fork a[i,i]:=p[i]
    r[i,i]:=i;
  fork a[n,n+1]:=0
  r[n,n+1]:=0
  join
  join
  for d:=1 to n-1 do
    forall i:=1 to n-d do
    j:=i+d
    k:=SGA(i,j)
    fork r[i,j]:=k
    w[i,j]:=w[i,j-1]+w[j,j]
    a[i,j]:=a[i,k-1]+a[k+1,j]+w[i,j]
  SGA(i,j):
    popsize:=(j-i+1) * popsizepercent / 100
    lchrom:=log j +1
    gen:=0
    initialize(gen)
    repeat
      gen:=gen+1
      generate(gen)
      k:=min(individuals)
      oldpop:=newpop
    until gen=maxgen
    return(k)
generate(gen):
  j:=1
  repeat
    fork mate1:=fitness_and_select(oldpop)
    mate2:=fitness_and_select(oldpop)
    join
    fork newpop[j]:=crossover_and_mutation
                (mate1,mate2,pcross,pmutate)
    newpop[j+1]:=crossover_and_mutation
                (mate1,mate2,pcross,pmutate)
    join
    j:=j+2
  until j > popsize
make_c(i,j,l):
```

```
k:=r[i,j]
if k<>0 then{
  fork c[k]:=1
  fork make_c(i,k-1,l+1)
  make_c(k+1,j,l+1)}
t:=a[1,n]
output(c,t)
```

Fig. 3 The Genetic Algorithm for the solution of Optimal Binary Search Tree

In the mentioned algorithm, first the number of keys, array P include possibilities of keys, and the genetic algorithm parameters are received from input, and then some entries of matrixes such as a, r, w are initiated. This work is done according to second part of equation 2. The matrix w stores the set of keys possibilities. In next part of the algorithm, matrixes entries are filled in the form of diagonal and parallel with the main diameter. This work is also done according to the first part of equation2. As it is obvious from Figure1, a master process is created for each secondary diameter and also main one. These processes create a worker for each element of diameters. Each worker process executes a simple genetic algorithm until the value of optimal k is calculated, and the obtained key is placed in matrix r. The master processes also are created as serial and according to the figure2. Finally, output array C is created according to matrix r, and the optimal searching time, $a[1,n]$, is sent to the output.

In executing of the genetic algorithm, the binary coding has been used. Each chromosome shows a value for k, and because of this, it's length shouldn't be more than $Log\ \ j+1$ for each process. On the other hand, the maximum of the genetic population for each process will be $j-i+1$. But for increasing the speed of the algorithm execution, we only apply a percent of the population. Here, the type of crossover is the common single point method. For creating the intermediate population and selecting the parents, the Roulette wheel has been used. Since the minimum value should be selected from the current population, the inverted objective function has been used. Therefore in the Roulette wheel, the optimal value allocates itself most of share, and in result the possibility of its selection will be more. Regarding to the problem structure, the global method is used. And because the processes work independently, and only they are synchronized in end of each stage, so class of this algorithm is global and semi-synchronous.

Ignoring the part related to the genetic algorithm in the main body of the program, the algorithm execution has time cost O(n), because the internal loop is executed in parallel. Notice in here, a processor is allocated to each process. If the number of processes is N, so this cost will be $O(n^2/N)$. Also in the genetic algorithm, there is a main loop which creates some generation of chromosomes, that it's the execution cost is $O(\max gen)$. Creating the primary population and making the chromosomes depends on the length of chromosomes. The time cost of creating a chromosome is $O(lchrom)$, and it's maximum will be happened in last stage of the algorithm. This value approximately is equal with $Log_2 n$. Due to the number of these chromosomes are equal with the genetic population, so time cost of creating a population is $O(popsize \times lchrom)$. The maximum size of genetic population is also $n \times popsizepercent$. If the chromosomes are created in parallel, so this value will be $O(popsize)$. In general, the executing cost of the genetic algorithm will be $O(\max gen \times popsize)$, and finally the total time cost will be $O(n^2 \times \max gen \times popsize/N)$. Now if there are enough processors, and operations are done in parallel, so the above cost will be simplified as follows :

$$O(n \times \max gen \times popsizepercent)$$

If above parameters are suitable, then the algorithm cost will be decreased, else the cost will be increased. It means that : $\max gen \times popsize$ percent $< n$.

Of course we should note that suitable performance of the algorithm depend on communication delay in the Network, time of creating and synchronizing of the processors, and the random distribution function used in the algorithm.

Also the cost of consumed memory will be as follow :

$$O(n^2 + \max(popsize) \times \max(lchrom))$$

Because the algorithm has applied three matrix of $(n+1)\times(n+1)$ and one record conclude a multi-chromosomes population.

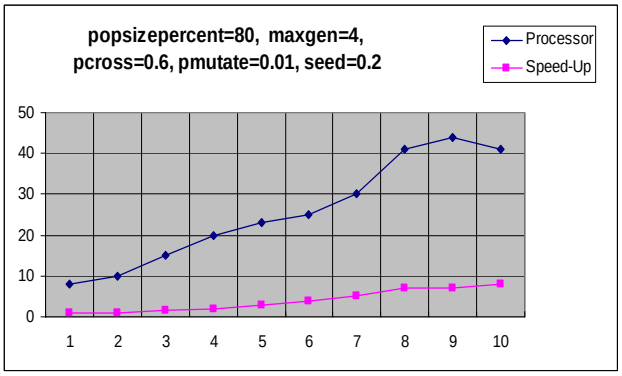
5. Experimental Results

A Multi-Pascal programming language has been used to implement the presented genetic algorithm. Multi-Pascal was designed by Bruce P. Lester as a research work in the computer group of International University of Maharishi (MIU). The goal of this design was creation a language with a set of high level instructions, so that we can simulate parallel algorithms on the multi-computers and multi-processors. This language consists many of standard parallel programming instructions. On the other hand, the instructions of this language are based on the standard Pascal language, which parallel capabilities added to it, and therefore it presents a simple high level language to

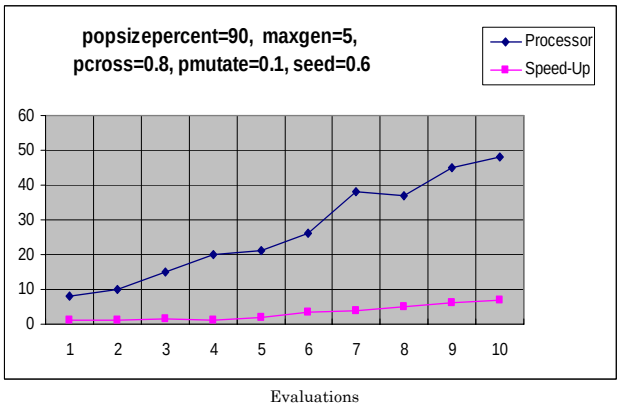
user. Multi-Pascal is a machine independent language, and we can simulate different parallel architectures by it. This language with two instructions FORK and FORALL, creates different processes as parallel. Instruction of JOIN along with FORK cause to join two processes of father and child in a point. Multi-Pascal uses the global variables as common, and it uses the internal variables of block (or a subroutine or a function) as a local case. It uses CHANNEL variable for sending a message. Also this language done the calculations related to run-time, processors number, speed-up and processors utilizations.

Here, results of different implementations of the algorithm are presented. Fig.4 shows four different implementations of the program. These diagrams show the speed-up and number of processors for different values of the genetic parameters. As it is seen, for less values of n, speed-up is small, and when it is increased, so the number of processors will be increased.

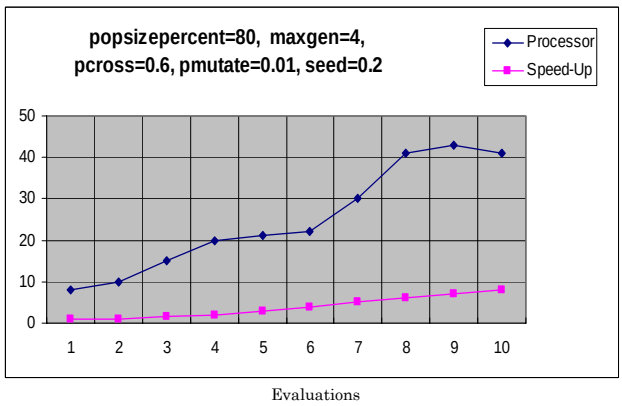
A



B



C



D

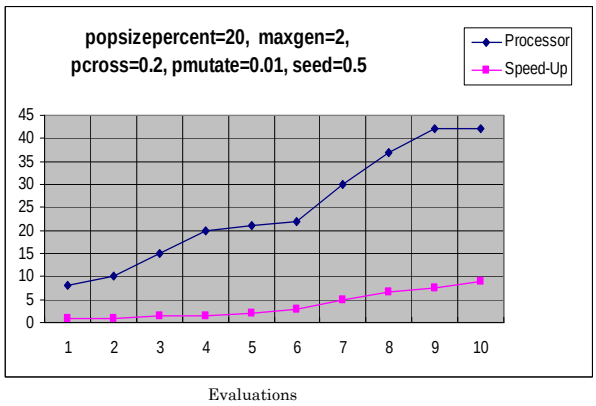


Fig. 4 Four different implementations of the Program

Fig.5 also shows speed-up of these four samples with each other. As it is seen, when n grows, the speed-up of sample 4 increase which it's genetic parameters are low, while the speed-up of sample 2 decrease which it's genetic parameters values are high.

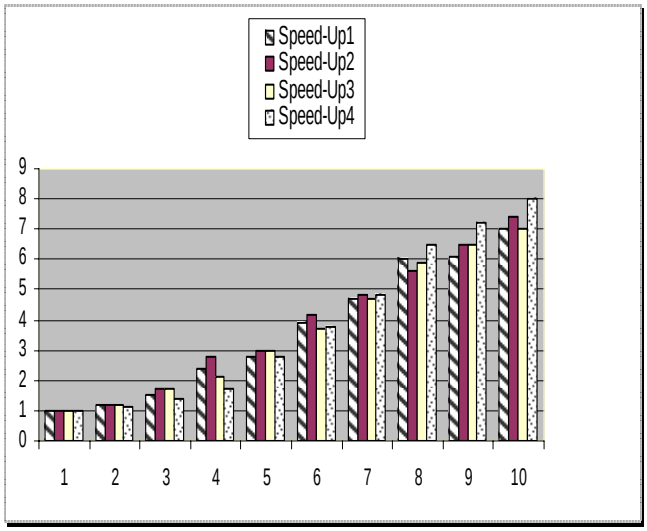


Fig. 5 Speed-up of the four Samples of Program

Fig.6 shows outputs of the algorithm execution with the same input values and different genetic parameters (fig.1 and fig.2). As it is seen, the different executions of this algorithm with the same inputs produce different outputs. And the output value decrease when greater genetic parameters are selected.

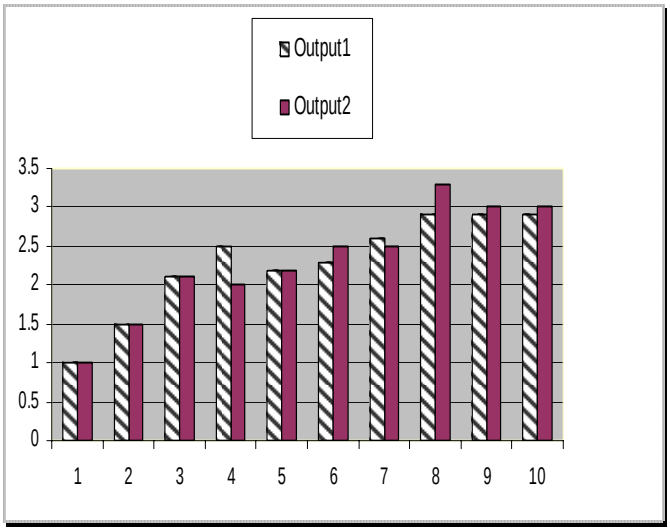


Fig.6 *Outputs of the algorithm execution*

6. Conclusion and future works

In this paper, a parallel genetic algorithm was presented for solution of the optimal binary search tree. To implement of this algorithm and obtaining the experimental results, a parallel programming environment called Multi-Pascal was used. First, a dynamic programming method was implemented, and then a genetic algorithm was added to it. Finally, by using of its parallel model and creating a semi-synchronous architecture of master/slave shared memory global genetic algorithm, the problem of optimal binary search tree was solved.

In general, we can refer to the following results:

- The simulation results show that speed-up will be increased when the number of inputs grows.
- The simulation results show that overhead of the algorithm execution is high when data is less, and their execution time is equal with usual methods.
- The simulation results show that executing of the genetic algorithm with the greater input has high speed.
- Optimizing the dynamic programming method using parallel genetic algorithms, increases the speed of the algorithm execution.

The works which can be done in this way are:

- Implementing other parallel genetic algorithms such as coarse grained and hybrid one.
- Implementing the parallel genetic algorithms on the different topologies.
- Finding the suitable values for parallel genetic algorithm parameters so that we reach to an answer with a high speed.
- Presenting a method for adjustment of population with parallel hardware.

References

[1] Chong Fuey Sian,"A Java Based Distributed Approach to Genetic Programming on the Internet", University of Birmingham School of Computer Science, 1999.

[2] David E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley,1989.

[3] Erick Cantu-Paz, "A Summary of Research on Parallel Genetic Algorithms", University of Illinois at Urbana-Champaign, Pages 1-12, 1995.

[4] Laurrens Jan Pit, "Parallel Genetic Algorithms", Master Thesis, Leiden University, 1995.

[5] M. Gorges-Schleuter, "ASPARAGOS : an Asynchronous Parallel Genetic Optimization Strategy", in Schaffer, Pages 422-427, 1985.

[6] Marek Obirko, "Genetic Algorithms", 1998.

[7] Markus Schwehm, "Parallel Population Models for Genetic Algorithms", University of Erlangen-Nurnberg, Pages 2-8, 1996.

[8] P. C. Chu and J. E. Beasley, "A Genetic Algorithm for the Set Partitioning Problem", The Management School Imperial College, Page 1,4-5, 1995.

[9] Ricardo Bianchini and Christopher Brown, "Parallel Genetic Algorithms on Distributed Memory Architecture", Prentice-Hall, 1993.

[10] Thomas Bak, "Evolutionary Algorithms in Theory and Practice", Oxford University, 1996.

[11] John H. Holland, “Genetic Algorithms”, 2005, http://www.econ.iastate.edu/tesfatsi/holland.GAIntro.htm

[12] A. Robinson, “Genetic Programming: Theory, Implementation, and the Evolution of Unconstrained Solutions”, Division III Thesis, Hampshire College, 2001.

[13] A. Kalynpur, M. Simon, “Pacman using Genetic Programming and Neural Networks”, Project Report for ENEE 459E, 2001.

[14] M. Obitco, “Introduction to Genetic Algorithms”, Czech Technical University, http://cs.felk.cvut.cz/~xobitko/ga/, visited in 2006.
[15] J. Fernandez, “Genetic Programming Network”, http://www.geneticprogramming.com/, visited in 2006.
[16] M. Nowostawski, R. Poli, “Parallel Genetic Algorithm Taxonomy”, Submitted to Publication to: KES’99, 1999.
[17] S. Hansen, and L. I. McCann, “Optimal Binary Search Tree Meet Object-Oriented Programming”, Computer Science Department, University of Wisconsin, WI53141, 2002.
[18] M. Schaefer, "Optimal Binary Search Tree" Department of Computer Science, DePaul University, Chicago, Illinois 60604, USA, 2006.
[19] S. Baase, and A. Gelder, “Computer Algorithms”, Introduction to Design and Analysis, Addison-Wesley, 2000.
[20] T. Budd, “Classic Data Structures in Java”, Addison-Wesley, 2001.
[21] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, “Introduction to Algorithms”, 2nd ed, MIT Press, 2001.
[22] M. Goodrich, and R. Tamassia, “Data Structures and Algorithms in Java”, Wiley, 1998.
[23] R. Sedgewick, “Algorithms”, 2nd ed. Addison-Wesley, 1998.
[24] R. Bianchini, and C. Brown, “Parallel Genetic Algorithms on Distributed-Memory Architectures”, Technical Reports 436, The University of Rochester, New York 14627,1993.
[25] S. Lin, W. F. Punch, and E. D. Goodman, “Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach”, in Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing, 1994.
[26] T. Maruyama, T. Hirose, and A. Konagaya, “A Fine-Grained Parallel Genetic Algorithm for Distributed Parallel Systems”, in Proceedings of the Fifth International Conference on Genetic Algorithms, Stephanie Forrest, Ed., San Mateo, CA, 1993.