Solving Graphical Steiner Tree Problem Using Parallel Genetic Algorithm

Nguyen Viet Huy
huynv-fit@mail.hut.edu.vn

Nguyen Duc Nghia
nghiand@it-hut.edu.vn

Department of Computer Science
Faculty of Information Technology, Hanoi University of Technology
No.1, Dai Co Viet road, Hanoi, Vietnam
Tel: (+84) 4 8696121 – Fax: (+84) 4 8692906

***Abstract*—The graphical Steiner tree problem is a classical combinatorial optimization problem that appears in many practically important applications. This paper presents a new parallel genetic algorithm for solving the problem. The presented algorithm is based on binary encoding, used the Distance Network Heuristic for evaluating fitness of individuals and is implemented in parallel using global population model. The results of experiments on the OR-Library tests are reported to show the algorithm's performance in comparison with other metaheuristics for the given problem. The speed-up of the parallel implementation is also discussed.**

Keywords: ***Steiner tree problem, parallel genetic algorithm.***

I. INTRODUCTION

The **G**raphical **S**teiner tree **P**roblem (GSP) is a classical combinatorial optimization problem which asks for the minimum weighted tree spanning some designated vertices of an undirected weighted graph.

Formally, the GSP can be defined as follows. Let $G=(V, E)$ be an undirected connected graph with a positive cost function $c: E \to R^+$ on the edges. Given a subset $N \subseteq V$ that is called terminal nodes set. A tree which is a subgraph of G is called Steiner tree if it spans all the terminal nodes in N. The graphical Steiner tree problem is to find the minimum weighted Steiner tree. The optimal solution is called the Steiner minimum tree (SMT). The nonterminal nodes in a Steiner tree are called Steiner nodes.

Fig. 1 shows an example of Steiner tree in graph: terminal nodes and edges of the Steiner tree are highlighted.

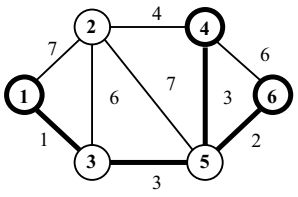

Figure 1. An example of Steiner tree in graph.

The GSP has many important applications in scientific and technology fields, for examples, the routing problem in VLSI layout, the designing problem for communication network, the phylogeny problem in biology, etc.

Two special cases of the problem: $N = V$ and $|N| = 2$ can be solved by polynomial time algorithms. When $N = V$, the optimal solution of GSP is obviously the spanning tree of G and thus the problem can be solved by polynomial time algorithms such as Prim's algorithm. When $|N| = 2$, the shortest path between two terminal nodes, which can be found by Dijkstra's algorithm, is exactly the Steiner minimum tree. However, the problem in general case, even with unit costs on the edges, has been shown to be NP-hard [12], so it cannot be solved in polynomial time, unless P = NP.

A survey of Steiner tree problem was given by Hwang and Richards [10]. Several exact algorithms have been proposed, such as dynamic programming technique given by Dreyfuss and Wagner [6], Lagrangean relaxation approach presented by Beasley [1], brand-and-cut algorithm used by Koch and Martin [13]. Duin and Volgenant presented some techniques to reduce the size of the graphs for the GSP [7].

Another approach for the GSP is using approximation algorithms to find near-optimal solution in reasonable time. Some heuristic algorithms have been developed, such as Shortest Path Heuristic (SPH) given by Takahashi and Matsuyama [27], Distance Network Heuristic (DNH) presented by Kou, Markowsky and Berman [15], Average Distance Heuristic (ADH) proposed by Rayward-Smith and Clare [23] and Path-Distance Heuristic (PDH) presented by Winter and MacGregor Smith [28]. Mehlhorn modified the DNH to make the algorithm faster [20]. Robins and Zelikovsky proposed algorithms improving the performance ratio [25, 26].

Recently, metaheuristics have been considered better methods for finding solutions closer to the optimum. Some metaheuristics can be enumerated such as Genetic algorithm (GA) [8, 11], GRASP [17], Tabu search [24]. Although these metaheuristics are polynomial time algorithms in general, they cost much time on large input sets. To deal with this issue, some parallel metaheuristic algorithms have been proposed,

such as Parallel GRASP [19], Parallel GRASP using hybrid local search [18], Parallel GA [5].

In this paper, we present a parallel genetic algorithm for the graphical Steiner tree problem. Our genetic algorithm is based on binary encoding idea [8, 11], uses the Distance Network Heuristics [15] as the fitness evaluation function and is parallelized by using global population model. The hill-climbing is applied to improve the quality of obtained solution. The hill-climbing routine is also implemented in parallel. To our knowledge, just only one parallel GA has been published previously [5], and it is based on ideas different from the ideas used in the algorithm given here.

The experiments have been carried out on OR-Library tests [2] to validate the efficiency of the algorithm and to compare with other metaheuristics in quality of solutions. The run-time of the proposed algorithm, when number of processes is changed, is also shown to verify its speed-up properties.

The paper is organized as follows. In the next section, the Distance Network Heuristics (DNH) is described. Section 3 presents the parallel genetic algorithm for the GSP. In section 4, results of the computational experiments are discussed. In section 5, we draw some conclusions and give an outlook to future work.

II. The Distance Network Heuristics for Graphical Steiner Tree Problem

Given graph $G = (V, E)$ with c and N as defined in last section, the DNH [15] first builds a complete graph G' on the terminal nodes set N with edge cost equal to length of the corresponding shortest path in G. Spanning tree M of G' is found, then each edge in M is replaced by the corresponding shortest path in G to obtain subgraph G''. Finally, the approximate solution of the GSP is the spanning tree of G'' after deleting repeatedly all Steiner nodes having degree 1.

For further presentation, the definition of distance network is given. For a given undirected connected graph G, let D_G denote the distance network of G which can be defined as follows:

- D_G is a complete graph having the same set of vertices as G.
- The cost of each edge connecting two vertices in D_G equals the cost of shortest path in G between these two vertices.

An example of the DNH is shown in Fig. 2. The algorithm DNH consists of five steps:

Input: $G = (V, E)$, positive cost function c, terminal nodes set N.

Output: Steiner tree T_{DNH}.

Step 1. Construct the distance network D_N with G, N, and c.

Step 2. Find minimum spanning tree M of D_N.

Step 3. Replace each edge in M by corresponding shortest path in G to get subgraph T.

Step 4. Find minimum spanning tree of T, called T_{DNH}.

Step 5. Delete repeatedly nonterminal nodes of T_{DNH} having degree 1.

Clearly, step 1 in the DNH costs the most time. It requires computing shortest paths for each couple vertices in N. This step runs in time $O\left(|N||V|^2\right)$, and is also the time bound for the remaining steps.

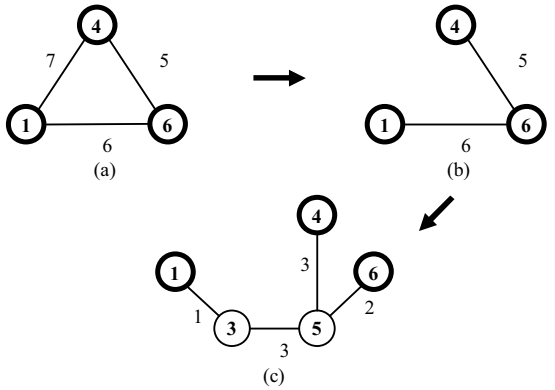

Figure 2. (a) Distance network for terminal nodes set in Fig. 1, (b) spanning tree of distance network, (c) Steiner tree T_{DNH}.

Beside the DNH, two other well-known polynomial time heuristics are Shortest Path Heuristics (SPH) [27] and Average Distance Heuristics (ADH) [23]. When compared to these heuristics, the solutions of DNH do not have better quality, but the DNH is chosen because it is relatively fast and easy to implement.

III. The Parallel Genetic algorithm for Graphical Steiner Tree Problem

In this section, we present a parallel genetic algorithm for graphical Steiner tree problem. Genetic algorithm (GA) is a class of adaptive search techniques, which gets ideas from principles of natural selection and evolution. In this paper, we do not describe GA in detail. For an introduction to GA, interested readers can refer to [22]. A survey of parallel GA was given by Cantú-Paz [3].

Basing on simple GA strategy, applying global population model, we develop a **P**arallel **G**enetic algorithm for graphical **S**teiner tree problem (PGS) using DNH [15] for fitness evaluation. According to global population model, fitness evaluation is implemented in parallel. The master process first divides the whole population into subsets and sends to each slave process a corresponding subset. Then, each slave process computes fitness for its part. During the fitness evaluation, there is no communication among processes. When the evaluation finishes, all slave processes return the fitness values to the master. The selection, crossover and mutation are executed only in the master process. A hill-climbing routine, which is also implemented in parallel, is executed after genetic algorithm finishes. By using local-search technique to improve quality of final solution, we can reduce number of generations

in genetic algorithm. Details of the PGS are described in sub-sections below.

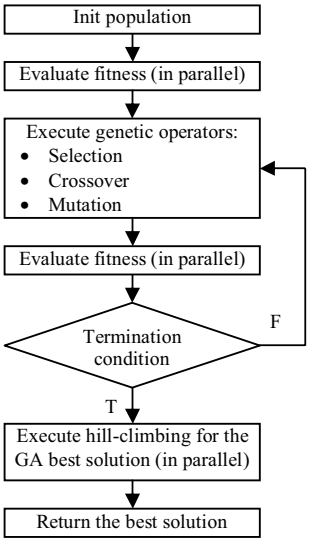

Figure 3. The parallel genetic algorithm

A. Genetic encoding and fitness evaluation

Given an instance of the GSP as defined in section 1, where c is the positive cost function, V is the vertices set, N is the terminal nodes set. Let V/N be the nonterminal nodes set, and $r = |V/N|$. Given a fixed numbering 0, 1 …, $r-1$ specifies the order of nonterminal vertices. A genotype is encoded as a string: $\{(B_1, I_1), (B_2, I_2) \ldots (B_{r-1}, I_{r-1})\}$. Each pair in genotype has two members, B_i is the bit value and I_i is the index of vertex in nonterminal nodes set, $I_i \in \{0,1,...,r-1\}$. This is called binary encoding; each bit in the string corresponds to a specific nonterminal vertex in the graph. By adding the vertex indices, genotype can be decoded independently of the ordering of its pairs. For example, two strings below present only one genotype,

{(0, 0), (0, 1), (1, 2), (1, 3), (0, 4)} and

{(0, 1), (1, 3), (0, 0), (1, 2), (0, 4)}.

The value of each bit decides whether corresponding vertex is selected or not. If the bit value is 1, the vertex is selected. Steiner tree will be built from the union of terminal nodes and set of selected vertices. Given a genotype s, let W_s be a set of all vertices in s having bit value 1. The Steiner tree corresponding to s, denoted by T_s, is computed by using the DNH on the set $N \cup W_s$, instead of N. Let $c(T_s)$ be the weight of T_s, so we define $\frac{1}{c(T_s)}$ as the fitness value of genotype s.

Steiner tree in D_G, if it exists, has no more than $|N|-2$ Steiner nodes [16]. Hence, it is sufficient to consider only genotypes s that $|W_s| < \min(|N|-2, r)$. All the other genotypes, which have set bits more than $\min(|N|-2, r)$, would be passed into a filter that clears all redundant set bits.

B. Selection

Basing on the fitness of individuals, the selection will chooses the individuals in the population that will be used to create offspring for the next generation. The purpose of selection is to increase the fitter individuals so that the next generations even have higher average fitness. Our genetic algorithm uses linear ranking method, which assigns an expected value presenting selection probability to each individual. First, each individual in the population is ranked in increasing order of fitness. Let Max be the expected value of the fittest individual. If P is the population size, then expected value for ith individual, denoted by e_i, is computed as follows:

$$e_i = 2 - Max + (2Max - 2)\frac{i}{P-1} \quad (1)$$

Once the expected values have been assigned, the Stochastic Universal Selection (SUS) [22] is used to select individuals.

C. Crossover

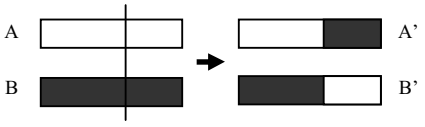

Figure 4. One cut-point crossover

After the selection, the crossover is executed to create new offspring. Two parent individuals are chosen randomly from selected individuals and are recombined with a probability p_c. A uniformly distributed random number k is generated, if $k \leq p_c$, two parent individuals undergo recombination to create two offspring. Otherwise, two offspring are simply the copies of their parents. The algorithm applies one cut-point crossover in which a cut-point is randomly selected in the genotype. Then two parts after cut-point of parents are exchanged to form two offspring.

D. Mutation

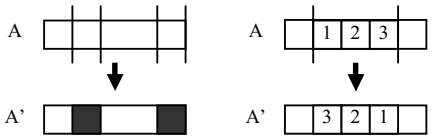

Figure 5. Bit-flip mutation (left) and inversion (right)

While the crossover forms hopefully better individuals, mutation adds diversity to the population and enlarges search space. In the algorithm, only two types of mutation: bit-flip mutation and inversion are applied. In bitwise mutation, each bit value in a genotype is changed with a certain probability p_m. After that, inversion works by choosing two points in the genotype and reversing the order of the bits between them with probability p_v.

The PGS starts with randomly generated population, and uses number of generations as termination condition. For the parallel implementation, the termination condition is broadcast

to all processes before the PGS is executed. The parallel fitness evaluation algorithm can be written in pseudo-code as follows:

Parallel fitness evaluation

//The master process
1. Divide the population in number of sub-sets which equals to number of processes,
2. Send to each process one corresponding subset called sub-population,
3. Get fitness values from slave processes.
4. END. *//of fitness evaluation*

//The slave process
1. If stopping condition is true,
2. Terminate,
3. Else
4. Receive corresponding sub-population from the master process,
5. Compute fitness values of the sub-population received,
6. Send fitness values to the master process
7. Go to 1,
8. END.

After the GA finishes, a hill-climbing routine runs to improve the solution founded by GA. To reduce run-time the hill-climbing is also implemented in parallel as follows. Let p be the number of processes which are numbered from 0 through $p-1$, and r be the length of genotype. Each genotype has r neighbors by flip systematically each bit, one at a time. Hence, each process computes fitness for $\left\lceil \frac{r}{p} \right\rceil$ neighbors. The ith process flips bits from $i\left\lceil \frac{r}{p} \right\rceil$ through $(i+1)\left\lceil \frac{r}{p} \right\rceil - 1$ to get its own neighbors; the last process flips bits from $(p-1)\left\lceil \frac{r}{p} \right\rceil$ to end. Pseudo-code for the parallel hill-climbing is given below:

Parallel hill-climbing
Input: the current_top which is the best solution founded by the PGS.

//The master process
01. Broadcast the current top to all processes,
02. Compute fitness values for corresponding genotypes to the process,
03. Get the best genotype found over all processes, called the new_best,
04. If the new_best is better than the current_top then
05. current_top = new_best,
06. Go to 01,
07. Else
08. Terminate all slave processes,
09. Return the current top.
10. END.

//The slave process
01. Get the current_top,
02. Compute fitness values for corresponding genotypes to the process,
03. Send the best genotype found to the master process,
04. Go to 01,
05. END. *//Terminated by the master*

IV. EXPERIMENTAL RESULTS

The proposed algorithm has been tested on all 78 graphs on four classes B, C, D, E of OR-Library [2] and the best founded solutions are compared with the optimal solutions. The test sets of OR-Library are chosen because they were solved exactly to find the optimal solutions. In addition, these test sets are also used to evaluate the performance of other algorithms for the graphical Steiner tree problem. Some important characteristics of these test sets are given in Tab I. Then, performance of the algorithm has also been compared to other metaheuristics. Last, the speed-up of parallel implementation has been discovered with the number of used processes varying from 2 to 10.

TABLE I. SIZE OF GRAPHS IN OR-LIBRARY.

	Class B	Class C	Class D	Class E
$\|V\|$	50-100	500	1000	2500
$\|E\|$	63-200	625-12500	1250-25000	3125-62500
$\|N\|$	9-50	5-250	5-500	5-1250

The following parameters of genetic operators of algorithm are fixed for all test problems:

- Linear ranking selection using SUS with expected value for the best individual $Max = 2$.
- One cut-point crossover with probability $p_c = 0.9$.
- Bitwise mutation with probability $p_m = 0.001$.
- Inversion with probability $p_v = 0.15$.

The PGS executed 10 times for each problem on classes B and C, 5 for times class D, and 2 times for class E. Size of population and number of generations are changed depending on each class. These parameters are given in Tab. II.

TABLE II. POPULATION SIZE AND NUMBER OF GENERATIONS FOR EACH CLASS OF THE OR-LIBRARY.

	Class B	Class C	Class D	Class E
Population size	40	100	150	250
No. of generations	50	100	200	300
No. of runs	10	10	5	2

Our algorithm is implemented on the local network with 9 computers having the same system configuration: Intel Pentium IV 3.06GHz, 480MB of RAM, 100Mbps Ethernet card, Windows XP Professional SP2. Computers are connected by a 2Gbps Gigabit Ethernet switch. The parallel program has been implemented using MPI [21] environment, which is established by installing DeinoMPI [4] on all computers for communication interface among processes. The PGS is written in C++ language using programming environment Microsoft Visual C++ 6.0.

A. Performance of the algorithm

First, the PGS is tested on all graphs of the OR-Library [2]. For each test graph, after the PGS is executed in determined times, the best, average and the worst results over all executions are recorded. All these results are listed in the end of the paper, from Tab. V to VIII. In each table, the *Opt*, *Best*, *Avrg*, *Worst* indicate the optimal solutions, the best, average and the worst results, respectively. The relative error in percentage of the best result in comparison with the optimal solution, denoted by $\Delta Best$, is computed as in (2). The relative error of the average results, $\Delta Avrg$, is similarly computed.

$$\Delta Best = \left(\frac{Best - Opt}{Opt} \right) \times 100\% \qquad (2)$$

On the classes B and C, the PGS finds the optimal solutions for all tests. On 28 over 38 graphs on classes B and C, the PGS finds the optimal solution in all 10 executions. The relative errors of average results are less than 1.9%. On class D, the PGS finds optimal solution for 17 graphs, and in addition, for 10 of them the optimal solution is found in all executions. For class D, relative errors of average results are less than 1.7%, and the maximum relative error of best result is 0.9%. On class E, the PGS finds optimum for 13 over 20 graphs. For 7 remaining graphs on class E, relative errors of best results are no more than 1.42% and the errors of average results are no more than 1.7% with respect to optimal solution. Thus for all 78 graphs in OR-Library test set, the PGS finds the optimal solution for 68 graphs, relative errors of the best and average results are less than 1.5% and 1.9%, respectively.

TABLE III. PERFORMANCE COMPARISON OF THE PGS WITH OTHER METAHEURISTICS.

	=0	**< 0.05**	**< 0.1**	**< 0.5**	**< 1**
PGS	50	51	53	57	59
GRASP	49	51	51	55	56
EGA	48	49	49	53	56
PGRASP	46	47	49	53	56
Tabu	33	35	38	46	49

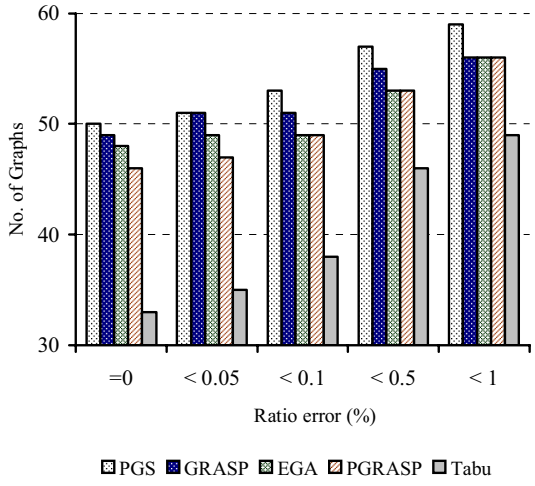

Figure 6. Performance comparison of the PGS with other metaheuristics.

Quality of solutions obtained by our PGS on classes C, D and E is also compared with other metaheuristics: Esbensen's GA with graph reduction (EGA) [8], GRASP [17], Parallel GRASP (PGRASP) [19], Tabu search [24]. For each problem, the best results of the PGS and other metaheuristics are recorded. The experimental results of these metaheuristics are recorded from mentioned papers. Then, the ratio error for each result is computed similarly to the $\Delta Best$.

Performance of algorithms is evaluated by comparing number of problems which have the best results with ratio errors less than a predefined value. 5 determined values of ratio error are chosen, from 0 through 1%. Only problems in classes C, D and E are used because all metaheuristics and our PGS find the optimal solution for all 18 graphs on class B. The comparison is shown in Tab. III and Fig. 6. According to Tab. III, the PGS has high performance in comparison to other metaheuristics. There are 50 over 60 graphs in which the PGS finds optimal solution. On 57 cases, the PGS finds the best result with ratio error less than 0.5%. There is only one case that ratio error is more than 1%. The statistic numbers of the PGS is higher than other metaheuristics in the test. Hence, the PGS could be considered a highly competitive solution for graphical Steiner tree problems.

B. Speed-up of the parallel implementation

The speed-up of the PGS is evaluated by comparing run-time when number of processes is changed. Graphs used for the experiment are on classes C and D. Number of processes is changed from 2 through 10; each process runs on a computer. However, when number of processes is 10, there is a computer running two processes, one of which is the master.

TABLE IV. AVERAGE RUN-TIME, RUN-TIME REDUCTION AND AVERAGE SPEED-UP.

	Class C			**Class D**		
No. of P	*Avrg run-time (s)*	*Reduc-tion*	*Speed-up*	*Avrg run-time (s)*	*Reduc-tion*	*Speed-up*
2	196.2	-	1	477	-	1
3	107.2	89.00	1.86	237.5	239.5	2.01
4	72.2	35.00	2.37	174.88	62.62	2.73
5	55.8	16.40	3.55	134.5	30.38	3.55
6	46.88	8.92	4.29	116.5	18	4.09
7	41.1	5.78	4.69	99.38	17.12	4.80
8	38.2	2.90	5.58	86.88	12.5	5.49
9	34.4	3.80	5.94	82.38	4.5	5.79
10	32.7	1.70	6.61	76.5	5.88	6.24

Tab. IV lists experimental results for problems on classes C and D. On class C, just problems from C16 through C20 are used for the test. On class D, only such problems: D9, D14, D18, D19 are chosen. Due to time limitation of experiment, we did not test all problems in OR-Library. *Avrg-run-time* is the average time of all executions on all determined problems at a specific number of processes. Each value in *Reduction* column is the difference of two adjacent *Avrg-run-time* rows, presenting the reduction of run-time whenever one process is added. *Speed-up* gained at a specific number of processes is computed as ratio of run-time when using two processes to run-time when using that number of processes. In the PGS, the master process does not take part in fitness evaluation, so there must be at least 2 processes when the PGS runs.

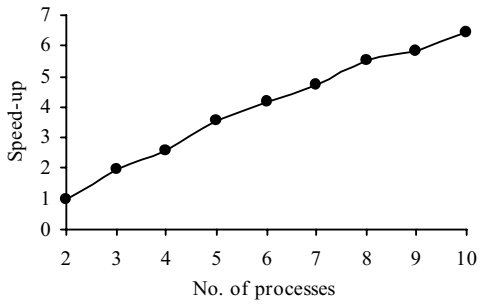

Figure 7. Average speed-up using up to 10 processes for classes C and D.

Fig. 7 shows the speed-up for the set of problems on classes C and D. According to the experiment, speed-up is gained when using parallel implementation. However, the run-time reduction decreases when we increases number of processes. As data in Tab. III, when number of processes increases from 2 to 3, run-time reduces nearly a half, but from 9 to 10 processes, run-time reduction is less than 6s. This is due to communication-time increases. Having more processes means the master has to spend more time on sending sub-populations and receiving fitness values. Although fitness-evaluation routine is implemented in parallel, send-receive are serial routine. This is a disadvantage of global population model when being implemented in group of sequential computers.

V. CONCLUSION

In this work, we present a parallel genetic algorithm applying global population model for solving graphical Steiner tree problem. According to the results, our algorithm finds the optimal solution for 68 over 78 graphs of OR-Library. For graphs that the algorithm does not find optimum, relative errors of average results do not exceed 1.9%. Small relative errors show that the algorithm is highly stable on many different classes of graph. Our algorithm also has better performance in comparison with other metaheuristics for graphical Steiner tree problem. Beside high performance of the algorithm, parallel implementation obtains speed-up. Average run-time decreases 6 times when the number of processes increases from 2 to 10. Nevertheless, graphs in OR-Library are relatively sparse; the average vertex degree is at most 50. Although this paper has been completed, experiments on dense and complete graphs have been being implemented by using test problems in class MC, X, I80, I160 and I320 in [14]. First experimental results are satisfactory; our algorithm find optimum for almost all test graphs.

However, one disadvantage of global population model is that the master process has to execute genetic operators for whole population and run a send-receive routine with each slave process. Hence, run-time in the master rapidly increases when population is large or there are many processes. These disadvantages make the global population model almost impossible to be applied in large computer systems to solve graphs having tens of thousands to millions vertices. To deal with this issue, our research direction in future is to apply more sophisticated parallel models to genetic algorithm. One of such models is multi-population GA. This model is also called island model parallel GA which uses relatively isolated sub-populations controlled by the same GA strategy. The communication among populations is done by the migration in which numbers of individuals are exchanged between each couple of sub-populations. By using this model, we hope performance of the GA will be improved.

COMPUTATIONAL RESULTS

This section gives a full report of the experimental results for all graphs of OR-Library. There are totally 78 problems, which consists of 18 problems on class B and 20 problems on each class C, D and E.

TABLE V. CLASS B

Problem	Opt	Best	Avrg	Worst	ΔBest	ΔAvrg
b01	82	82	82	82	0	0
b02	83	83	83	83	0	0
b03	138	138	138	138	0	0
b04	59	59	59	59	0	0
b05	61	61	61	61	0	0
b06	122	122	122	122	0	0
b07	111	111	111	111	0	0
b08	104	104	104	104	0	0
b09	220	220	220	220	0	0
b10	86	86	86	86	0	0
b11	88	88	88	88	0	0
b12	174	174	174	174	0	0
b13	165	165	165	165	0	0
b14	235	235	235.1	236	0	0.04
b15	318	318	318	318	0	0
b16	127	127	127	127	0	0
b17	131	131	131	131	0	0
b18	218	218	218	218	0	0

TABLE VI. CLASS C

Problem	Opt	Best	Avrg	Worst	ΔBest	ΔAvrg
c01	85	85	85	85	0	0
c02	144	144	144	144	0	0
c03	754	754	754	754	0	0
c04	1079	1079	1080.5	1083	0	0.14
c05	1579	1579	1579	1579	0	0
c06	55	55	55	55	0	0
c07	102	102	102	102	0	0
c08	509	509	509.8	511	0	0.16
c09	707	707	707.9	709	0	0.13
c10	1093	1093	1093.1	1094	0	0.01
c11	32	32	32	32	0	0
c12	46	46	46	46	0	0
c13	258	258	259.4	260	0	0.54
c14	323	323	323.4	324	0	0.12
c15	556	556	556	556	0	0
c16	11	11	11.2	12	0	1.82
c17	18	18	18	18	0	0
c18	113	113	114.7	116	0	1.50
c19	146	146	147.7	149	0	1.16
c20	267	267	267	267	0	0

TABLE VII. CLASS D

Problem	Opt	Best	Avrg	Worst	ΔBest	ΔAvrg
d01	106	106	106	106	0	0
d02	220	220	220	220	0	0
d03	1565	1565	1566.6	1569	0	0.10
d04	1935	1935	1935	1935	0	0
d05	3250	3250	3250.6	3251	0	0.02
d06	67	67	67	67	0	0
d07	103	103	103	103	0	0
d08	1072	1072	1073	1074	0	0.09
d09	1448	1448	1449.6	1451	0	0.11
d10	2110	2110	2110.6	2111	0	0.03
d11	29	29	29	29	0	0
d12	42	42	42	42	0	0
d13	500	500	500.6	501	0	0.12
d14	667	669	669.4	670	0.30	0.36
d15	1116	1116	1116.4	1117	0	0.04
d16	13	13	13	13	0	0
d17	23	23	23	23	0	0
d18	223	225	226.8	228	0.90	1.70
d19	310	312	313	314	0.65	0.97
d20	537	537	537	537	0	0

TABLE VIII. CLASS E

Problem	Opt	Best	Avrg	Worst	ΔBest	ΔAvrg
e01	111	111	111	111	0	0
e02	214	214	214	214	0	0
e03	4013	4015	4018	4021	0.05	0.12
e04	5101	5101	5102	5103	0	0.02
e05	8128	8128	8128	8128	0	0
e06	73	73	73	73	0	0
e07	145	145	145	145	0	0
e08	2640	2645	2646.5	2648	0.19	0.25
e09	3604	3607	3607.5	3608	0.08	0.10
e10	5600	5600	5600.5	5601	0	0.01
e11	34	34	34	34	0	0
e12	67	67	67	67	0	0
e13	1280	1286	1288.5	1290	0.47	0.66
e14	1732	1733	1733.5	1734	0.06	0.09
e15	2784	2784	2784.5	2785	0	0.02
e16	15	15	15	15	0	0
e17	25	25	25	25	0	0
e18	564	572	573.5	575	1.42	1.68
e19	758	761	761.5	762	0.40	0.46
e20	1342	1342	1342	1342	0	0

REFERENCES

[1] J. E. Beasley, An SST-Based Algorithm for the Steiner Problem in Graphs, Networks, Vol.19, 1-16, l989.

[2] J. E. Beasley, OR-Library: distributing test problems by electronic mail, URL http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html

[3] E. Cantú-Paz, A Survey of Parallel Genetic Algorithms, Technical Report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1997.

[4] DeinoMPI, Deino Software © 2006, URL http://mpi.deino.net

[5] G. Di Fatta, G. Lo Presti, G. Lo Re, A Parallel Genetic Algorithm for the Steiner Problem in Networks, 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003), Marina del Rey, CA, USA, 569-573, 2003.

[6] S. E. Dreyfuss, R. A. Wagner, The Steiner Problem in Graphs, Networks, Vol.1, 195-207, 1971.

[7] C. W. Duin, A. Volgenant, Reduction Tests for the Steiner Problem in Graphs, Networks, Vol.19, 549-567, 1989.

[8] H. Esbensen, Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm, Networks, Vol.26, 173-185, 1995.

[9] P. J. B. Hancock, An Empirical Comparison of Selection Methods in Evolutionary Algorithms, Lecture Notes in Computer Science, Springer-Verlag, 80-94, 1994.

[10] F. K. Hwang, Dana S. Richards, Steiner Tree Problems, Networks, Vol.22, 55-89, 1992.

[11] A. Kapsalis, V. J. Rayward-Smith, G. D. Smith, Solving the Graphical Steiner Tree Problem Using Genetic Algorithms, Journal of the Operational Research Society, Vol.44, No.4, 397-406, 1993.

[12] R. M. Karp, Reducibility among Combinatorial Problems, Complexity of Computer Computations, (R. E. Miller, J. W. Thatcher Eds.) Plenum Press, New York, 85-103, 1972.

[13] T. Koch, A. Martin, Solving Steiner Tree Problems in Graphs to Optimality, Networks, Vol.32, 207-232, 1998.

[14] T. Koch, A. Martin, S. Voss, SteinLib: An Updated Library on Steiner Tree Problems in Graphs, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2000, URL http://elib.zib.de/steinlib/steinlib.php

[15] L. Kou, G. Markowsky, L. Berman, A Fast Algorithm for Steiner Trees, Acta Informatica, Vol.15, 141-145, 1981.

[16] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[17] S. L. Martins, P. Pardalos, M.G. Resende, and C.C. Ribeiro, Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol.43, 133-146, 1999.

[18] S. L. Martins, M. G. C. Resende, C.C. Ribeiro, and P.M. Pardalos. A Parallel GRASP for the Steiner Tree Problem in Graphs Using a Hybrid Local Search Strategy, Journal of Global Optimization, 267-283, 2000.

[19] S. L. Martins, C. C. Ribeiro, M. C. Souza, A Parallel GRASP for the Steiner Problem in Graphs, Lecture Notes in Computer Science, Springer-Verlag, Vol.1457, 310-331, 1998.

[20] K. Mehlhorn, A Faster Approximation Algorithm for the Steiner Problem in Graphs, Information Processing Letters archive, Vol.27, 125-128, 1988.

[21] MPI: A Message-Passing Interface Standard (Version 1.1), Message Passing Interface Forum, Technical report, University of Tennessee, 1995.

[22] M. Mitchell, An Introduction to Genetic Algorithms, The MIT Press, 1998.

[23] V. J. Rayward-Smith, A. Clare, On Finding Steiner Vertices, Networks, Vol.16, 283-294, 1986.

[24] C. C. Ribeiro, M.C. Souza, Tabu Search for the Steiner Problem in Graphs, Networks, Vol.36, 138-146, 2000.

[25] G. Robins, A. Zelikovsky, Improved Steiner Tree Approximation in Graphs, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, ACM, New York, 770-779, 2000.

[26] G. Robins, A. Zelikovsky, Tighter Bounds for Graph Steiner Tree Approximation, SIAM Journal on Discrete Mathematics Vol.19, No.1, 122-134, 2005.

[27] H. Takahashi, A. Matsuyama, An Approximate Solution for the Steiner Problem in Graphs, Mathematica Japonica, Vol.24, No.6, 573-577, 1980.

[28] P. Winter, J. MacGregor Smith, Path-Distance Heuristics for the Steiner Problem in Undirected Networks, Algorithmica, Vol.7, 309-327, 1992.