Discussions on LGA with Parallel System

PENG Gang
Oita National College of Technology
1666, Maki
Oita, Japan
peng@oita.ac-ct.ac.jp

Takeshi NAKATSURU
Oita National College of Technology
1666, Maki
Oita, Japan
nakatsuru@oita.ac-ct.ac.jp

Shigeru NAKAYAMA
Kagoshima University
1-21-40, Korimoto
Kagoshima,Japan
shignaka@ics.kagoshima-u.ac.jp

ABSTRACT

This paper discusses a parallel genetic algorithm (PGA) which focuses on the local operator for Traveling salesman problem (TSP). The local operator is a simple GA named as Local Genetic Algorithm (LGA). The LGA is combined to another GA named as Global Genetic Algorithm (GGA). It increases the computational time running a GA as a local operator in another one. To solve this problem, we build a parallel system based on our previous works for running the LGA to speed up the process. The results show that LGA improve the search quality significantly and it is more efficient running LGA with parallel system than single CPU.

Keywords

genetic algorithm (GA), parallel GA, global GA, local GA, object shared space, Traveling Salesman Problem

1. INTRODUCTION

Local search acts as an important role in most current algorithms for optimal combinatorial problems. The local operators are usually repeated many times in the search algorithms. It is a key work how to design the local operators and determine their parameters in the algorithms. For TSP, many excellent local operators have been proposed in the past years.

GA requires an enormous amount of calculation because the genetic operations of selection, crossover, mutation and the fitness calculation of each individual are processed over many generations. The calculation time will be significantly different with the different local operations. GA has high parallelism because the genetic operations and the fitness calculation can be processed simultaneously in every individual. Various parallelization types have been proposed to speed up the computation of GA such as single group type and multiple group type[5].

```
Initialization
  Global GA {
      Deletion
      Crossover
      Mutation
      Smallest Square
      Complete 2-Opt
      Local GA{
          Selection
          Crossover
          Mutation
      }
  }
End
```

Figure 1: Algorithm

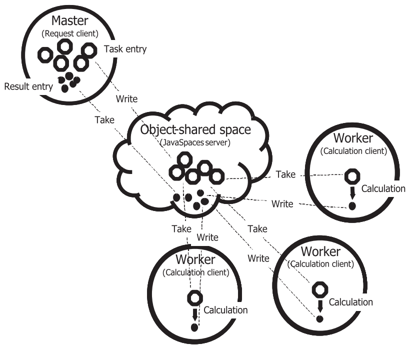

Figure 2: Parallel Implementation

We presented two algorithms at conference GECCO'03[4] and GECCO'04[3] as late-breaking papers. The first one is a multiple heuristic search algorithm which consists of a few local search operators such as Deletion, Best part collector (BPC), Smallest square (SS) and Complete 2-opt (C2Opt). Another one is combined with two simple Genetic Algorithms focus on the local operation. A detailed discussion was made on how a simple GA performs as an local operator in the second algorithm. The local operator is named as LGA. The result has shown that it is computationally expensive running LGA in another algorithm with a single CPU. This paper discusses a parallel genetic algorithm (PGA) which focuses on the local operation for TSP based on our previous works. LGA is used as an operator in the

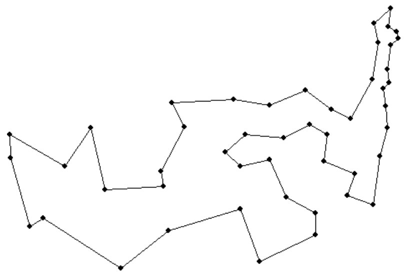

Figure 3: att48

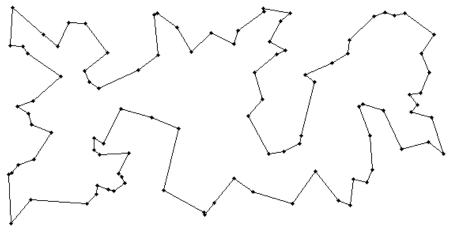

Figure 4: kroc100

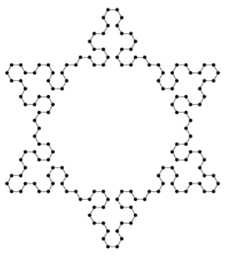

Figure 5: David162

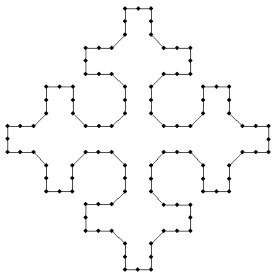

Figure 6: MPeano108

algorithm called as GGA. We build a parallel system for running the LGA to test its efficiency with different cases. The implementation of parallel GA is based on a multiple group type island model, that uses object shared space. The results show that LGA improve the search quality significantly with parallel system than single CPU.

2. ALGORITHM

Figure 1. shows the algorithm designed for our experiment. The LGA which consists of basic genetic operators only is implemented to the GGA. The LGA performs as a local search operator in the GGA. The GGA is applied to the main tours of the TSP instances for searching the global optimal solutions. Because both the GGA and the LGA are complete genetic algorithms, all genetic operators for solving the TSP are probably applicable to them. To obtain the global optimal solutions in the process of the TSP, the GGA may just need to improve the order of a set of cities in the best tour with evolution of the process, especially in the latter stages of the process. We chose a set of continuous cities to create the sub tour in the experiment. The LGA was applied to the sub tour for finding the local optimal solution to replace the original part chosen from the main tour. The genetic operations used in the experiment are as follows: selection is elitism; crossover is two-point crossover to the individuals chosen by the roulette wheel rule; and mutation is one-point mutation. Other operators in the algorithm have been described in detail at conference ’03 [4].

3. IMPLEMENTATION OF PARALLEL SYSTEM

The implementation for running the LGA adopts the object shared space "JavaSpaces" [2, 6] which is a Linda model in the Java programming language, and employs a replicated worker pattern (The Linda model enables a flexible distributed parallel processing). Figure 2. shows the conceptual diagram of the replicated worker pattern on the object shared space. The system consists of one master and three workers. The master creates an main tours (main population) for GGA and writes the entries of sub tours to the object shared space. The workers take the entries and create sub tours (sub populations) for LGA. The master watches the object shared space, and takes the entries of the elite individuals of the LGAs from the workers in each generation. When the number of generations reached to the number of generations defined beforehand, the parallel processing of LGA is completed. The object shared space is also used to temporarily store the elite individuals. In the object shared space built in the Java programming language, it is possible to communicate without specifying objects with an IP address. The parallel processing of the LGA with extendibility and generality can be achieved using this implementation.

4. RESULTS AND DISCUSSIONS

In the LGA, the sub tour is an open tour. The length of the tour is calculated from the start city to the end city, not including the distance between the end city and the start city. The main tour which contains n cities $c = \{c_0, ..., c_i, c_{i+1}, ..., c_{n-1}\}$ is given and the distance between two cities c_i and c_{i+1} is $d(c_i, c_{i+1})$. A sub tour which contains m cities from main tour is $c_s = \{c_0, ..., c_k, c_{k+1}, ..., c_{m-1}\}$ and the distance between two cities c_k and c_{k+1} is $d(c_k, c_{k+1})$. Total distances of the main tour and the sub tour are D_{mt} and D_{st} respectively:

$$D_{mt}=\sum_{i=0}^{n-1} d(c_i, c_{i+1}) \tag{1}$$

When $i=n-1$, $c_{i+1}=c_0$.

Table 1: Results with Single CPU

LGA	Shortest Time(s)	Longest Time(s)	Mean Time(s)	Total Time(s)	Optimums
0	25.4	25.4	25.4	3406.3	1
1	6.6	102.3	51.0	6040.6	9
5	9.5	327.4	80.1	10577.3	15
10	19.9	120.6	55.1	173078.0	15

Notation Short Time: Shortest time for obtaining the optimal solution. Long Time: Longest time for obtaining the optimal solution. Mean Time: Mean time for obtaining the optimal solution. Total Time: Time for running the total cycle. Optimums: Number of the optimal solutions.

Table 2: Results with Parallel system

LGA	Shortest Time(s)	Longest Time(s)	Mean Time(s)	Total Time(s)	Optimums
5	15.7	248.3	61.8	7675.43	14
10	18.9	176.8	40.5	10334.6	14

Notation Shortest Time: Shortest time for obtaining the optimal solution. Longest Time: Longest time for obtaining the optimal solution. Mean Time: Mean time for obtaining the optimal solution. Total Time: Time for running the total cycle. Optimums: Number of the optimal solutions/Number of the runs.

$$D_{st}=\sum_{k=0}^{m-2} d(c_k,\, c_{k+1}) \tag{2}$$

Here are the genetic parameters used in the experiment: crossover is set at 75% and mutation is set at 0.5% to all individuals in both GGA and LGA. Other parameters are the same as our previous works. The source code is written in the Java and the specs of computers in the experiment are CPU Pentium III 2.40GHz and 256 MB memory with Windows XP Operating System.

Four instances are used in the experiments. Two of them are from TSPLIB (Figure 3. and Figure 4.) and another two are fractal TSP instances which are generated by L-System[1](Figure 5. and Figure 6.). On single CPU, four cases of LGA were set in the GGA: a, 0 LGA. b, 1 LGA. c, 5 LGAs. d, 10 LGAs. For the parallel system, 2 cases of LGA were set in the GGA: a, 5 LGAs. b, 10 LGAs. Table 1 and table 2 show the total time for running the algorithms and the time for reaching to the optimal solutions. The number of optimal solutions obtained in all cases is investigated and also shown in the two tables. The data in the tables are average values of the four instances. On single CPU, the number of optimal solutions increased with the increase of LGAs in the algorithm. On the other hand, the total time for running the algorithm increased significantly, but the time for reaching to the optimal solutions did not increase rapidly. On the parallel system, the number of optimal solutions are almost the same as single CPU, but the total time for running the algorithm decreased significantly. It shows that the LGA is an efficient method for the parallel system.

5. CONCLUSIONS

This paper discussed an algorithm focusing on the local operation which is a simple GA named as LGA. The results show that the LGA can improve the search quality on single CPU and parallel system. It is more efficient running LGA on the parallel system than on single CPU.

6. ADDITIONAL AUTHORS

Additional authors: Ichiro Iimura (Department of Administration, Faculty of Administration, Prefectural University of Kumamoto, email: `iiimura@pu-kumamoto.ac.jp`).

7. REFERENCES

[1] A.Marian, P.Moscato, and M. Norman. *Using L-Systems to generate arbitrarily large instances of the Euclidean Traveling Salesman Problem with known optimal tours.* Nov 1995.

[2] E. Freeman, S. Hupfer, and K. Arnold. Javaspaces principles, patterns, and practice. 1999.

[3] P. Gang, I. Iimura, H. Tsurusawa, and S. Nakayama. A local search algorithm based on genetic recombination for traveling salesman problem. In *Genetic and Evolutionary Computation Conference (GECCO-2004).*, pages CD–ROM. GECCO, June 2004.

[4] P. Gang and S. Nakayama. Multiple heuristic search in genetic algorithm for traveling salesman problem. pages 88–93, July 2003.

[5] N. Sannomiya, H. Kita, H. Tamaki, and T. Iwamoto. Genetic algorithms and optimization. 1998.

[6] T. Yuizono and S. Nakayama. *Trial Experiments of Distributed Parallel Processing with JavaSpaces: Linda-based Object Shared Space*, volume 99.