A Parallel Hybrid Heuristic for the TSP

Ranieri Baraglia[1], José Ignacio Hidalgo[2], and Raffaele Perego[1]

[1] CNUCE - Institute of the Italian National Research Council
Via Alfieri 1, 56010 S. Giuliano Terme, Pisa (Italy)
{Ranieri.Baraglia, Raffaele.Perego}@cnuce.cnr.it
[2] Dpto. Arquitectura Computadores y Automatica
Universidad Complutense, Madrid 28040
hidalgo@dacya.ucm.es

Abstract. In this paper we investigate the design of a coarse-grained parallel implementation of *Cga-LK*, a hybrid heuristic for the Traveling Salesman Problem (TSP). Cga-LK exploits a compact genetic algorithm in order to generate high-quality tours which are then refined by means of an efficient implementation of the Lin-Kernighan local search heuristic. The results of several experiments conducted on a cluster of workstations with different TSP instances show the efficacy of the parallelism exploitation.

Keywords: Parallel algorithms, TSP, compact genetic algorithm, Lin-Kernighan algorithm, hybrid GA.

1 Introduction

The Traveling Salesman Problem (TSP) is the problem of finding the shortest closed tour through a given set of cities visiting each city exactly once. Thus, given a set of cities $C = \{c_1, c_2, ..., c_k\}$, for each pair (c_i, c_j), $i \neq j$, let $d(c_i, c_j)$ be the distance between city c_i and c_j. Solving the TSP entails finding a permutation $\pi\prime$ of the cities $(c_{\pi'(1)}, ..., c_{\pi'(k)})$, such that

$$\sum_{i=1}^{k} d(c_{\pi'(i)}, c_{\pi'(i+1)}) \leq \sum_{i=1}^{k} d(c_{\pi(i)}, c_{\pi(i+1)}) \qquad \forall \pi \neq \pi', (k+1) \equiv 1 \qquad (1)$$

In the *symmetric* TSP $d(c_i, c_j) = d(c_j, c_i), \forall i, j$, while in the *asymmetric* TSP this condition is not satisfied. In this work we consider the symmetric TSP.

Since the TSP is probably the most well-known NP-hard combinatorial optimization problem, researchers have proposed many heuristics for searching the space of all permutations π. Problem-independent heuristics such as simulated annealing (SA) [1] and genetic algorithms (GA) [2,3] perform quite poorly with this particular problem. They require high execution times for solutions whose quality is not comparable with those achieved in much less time by their domain-specific local search counterparts. Domain-specific heuristics such as *2-Opt* [4],

E.J.W. Boers et al. (Eds.) EvoWorkshop 2001, LNCS 2037, pp. 193–202, 2001.

3-Opt [5], and Lin-Kernighan (LK) [6] are effective. In particular LK is considered to be the heuristic that leads to the best solutions. Efficient implementations have been devised for LK which take just a few seconds to compute a high-quality solution for problems with hundreds of cities [7,8].

Several published results demonstrate that combining a problem-independent heuristic with a local search method is a viable and effective approach for finding high-quality solutions of large TSPs. The problem-independent part of the hybrid algorithm drives the exploration of the search space focusing on the global optimization task, while the local search algorithm allows to search in depth the subregions of the solution space.

In [9] the *Chained local optimization* algorithm is proposed. It exploits a special type of *4-opt* moves under the control of a SA mechanism to escape from the local optima found with LK. In [10] genetic operators to search the space of the local optima determined with LK are proposed.

Martina Georges-Schleuter experimented with the exploitation of simple k-Opt moves within her *Asparagos96* parallel genetic algorithm [11]. She concluded that, for large problem instances, the strategy of producing many fast solutions might be almost as effective as using powerful local search methods with fewer solutions.

In this paper we propose a coarse-grained parallel implementation of Cga-LK, a previously proposed hybrid heuristic for the TSP which exploits a compact genetic algorithm (Cga) in order to generate high-quality tours which are then refined by means of the Lin-Kernighan local search heuristic. The parallel implementation allowed us to enhance the encouraging results obtained with the sequential version of Cga-LK [12,13].

The rest of the paper is organized as follows: Section 2 gives a brief introduction to the Cga. Section 3 describes the parallel implementation of Cga-LK, and discusses the experimental results. Finally, Section 4 outlines some conclusions.

2 The Compact Genetic Algorithm

The Cga does not manage a population of solutions but only mimics its existence [14]. The Cga represents the population by means of a vector of values $p_i \in [0,1], \forall i = 1,\ldots,l$, where l is the number of alleles needed to represent the solutions. Each value p_i indicates the proportion of individuals in the simulated population which has a 0 or 1 in the i^{th} locus of their representation. By treating these values as probabilities, new individuals can be generated and, based on their fitness, the probability vector can be updated in order to favor the generation of better individuals.

The values for probabilities p_i are initially set to 0.5 to represent a randomly generated population in which the value for each allele has equal probability. At each iteration the Cga generates two individuals on the basis of the current probability vector and compares their fitness. Let W be the representation of the individual with a better fitness and L the individual whose fitness is worse.

The two representations are used to update the probability vector at step $k+1$ in the following way:

$$p_i^{k+1} = \begin{cases} p_i^k + 1/n & \text{if } w_i = 1 \wedge l_i = 0 \\ p_i^k - 1/n & \text{if } w_i = 0 \wedge l_i = 1 \\ p_i^k & \text{if } w_i = l_i \end{cases} \quad (2)$$

where n is the dimension of the population simulated, and w_i (l_i) is the value of the i^{th} allele of W (L). The Cga ends when the values of the probability vector are all equal to 0 or 1. At this point vector p represents the solution obtained.

In order to represent a population of n individuals, the Cga updates the probability vector by a constant value equal to $1/n$. Only $\log_2 n$ bits are thus needed to store the finite set of values for each p_i. The Cga therefore requires $l \cdot \log_2 n$ bits with respect to the $n \cdot l$ bits needed by a classic GA. Larger populations can be thus exploited without significantly increasing memory requirements, but only slowing Cga convergence. This peculiarity makes the use of Cga attractive to address problems for which the huge memory requirements of GAs is a constraint.

On order-one problems the Cga and the simple GA with uniform crossover are approximately equivalent in terms of solution quality and in the number of function evaluations needed. To solve higher than order-one problems GAs with both higher selection rates and larger population sizes have to be used [15]. The Cga selection rate can be increased by adopting the following mechanism: (I) generate at each iteration s individuals from the probability vector; (II) choose among the s individuals the one with best fitness; (III) compare the individual with best fitness with the other $s-1$ individuals and update the probability vector according to (2). Such an increase on the selection rate helps the Cga to converge to better solutions since it increases the survival probability of higher-order building blocks.

3 A Parallel Hybrid Heuristic for the TSP

Cga-LK combines a Cga with *Chained LK* [1], an efficient implementation of LK proposed by Applegate Bixby, Chvatal, and Cook [8]. In Cga-LK the Cga is used to explore the more promising part of the TSP solution space in order to generate "good" initial solutions which are refined with Chained LK. The refined solutions are also exploited to improve the quality of the simulated population as the execution progresses. A detailed description of the sequential implementation of Cga-LK can be found in [12]. Here we will concentrate the attention on the parallelism exploitation.

The coarse-grained parallelization model [16,17] was used to design the parallel version of Cga-LK. According to this model the whole population is divided into a few demes, which evolve in parallel. Each deme is assigned to a different

[1] This routine is available in the CONCORDE library at url http://www.keck.caam.rice.edu/concorde.

processor and the evolution process takes place only among individuals belonging to the same deme. This feature means that a greater genetic diversity can be maintained with respect to the exploitation of a panmitic population, thus improving the solution space exploration. Since the size of the demes is smaller than the population used by the correspondent serial GA, in general, a parallel GA converges faster. Moreover, it is also true that the quality of the solution might be poorer than that of the sequential case. Therefore, in order to improve the deme genotypes, a migration operator that periodically exchanges the best solutions among different demes is used. Depending on the migration operator chosen we can distinguish between the *island* model and the *stepping stone* model. In the island model migration occurs among every deme, while in stepping stone model the migration occurs only between neighboring demes. Studies have shown that there are two critical factors [18]: the number of solutions migrated each time and the interval time between two consecutive migrations. A large number of migrants leads to the behavior of the island model similar to the behavior of a panmitic model. A few migrants prevent the GA from mixing the genotypes, and thus reduce the possibility to bypass the local optima inside the islands. Implementations of coarse grained parallel GAs can be found in [18, 19,20,21,22,23].

To implement Cga-LK the island model was adopted. Moreover, Cga-LK exploits the MPI message-passing library [24], and the SPMD programming model [25]. According to this programming model all the processing nodes run the same code which simulates a different population. We can consider each simulated population elaborated in parallel an island of the same larger population. To improve the sub-population genotypes, a migration operator that periodically exchanges the best solution among different islands was adopted.

To extend the Cga to the TSP, we considered the frequencies of the edges occurring in the simulated population. A $k \times k$ triangular matrix of probabilities P was used to store these frequencies. Each element $p_{i,j}, i > j$, of P represents the proportion of individuals whose tour contains edge (c_i, c_j). If n is the population dimension, our Cga thus requires $(k^2/2) \cdot \log_2 n$ bits to represent the population, compared with the $k \cdot n \log_2 k$ bits required by a classical GA. Figure 1 shows the pseudo-code of our parallel Cga-LK. Its main functions are discussed in the following.

After setting the MPI environment, the matrix P is initialized. To this end, first we randomly generate a tour to which the Chained LK routine is applied to carry out a local optimum. Then the probability values associated to all the edges belonging to the local optimum are increased by $1/n$. This procedure is iteratively applied n times to represent in P the whole simulated population.

To differentiate its behavior each parallel process uses a different seed (`local_seed`) for the pseudo-random generation. It is obtained by adding to the same seed (`global_seed`) the process identifier.

At each generation k of Cga-LK, a single individual L (`current_tour`) is generated from the probability matrix. To this end a *greedy* algorithm is used. A starting city c_a is randomly selected and inserted in the tour $\mathcal{V}$. Then, another

```
Program Par-Cga-LK
  begin
    /* Setting of the MPI environment, me is the process identifier */
    MPI_Init(...);
    MPI_Comm_rank(MPI_COMM_WORLD,&me);
    /* Initialization of the probability matrix P */
    local_seed := global_seed + me;
    Initialize(P, local_seed);
    best_tour_length = MAX_INT;
    generations := 0;
    repeat
       generations := generations + 1;
       current_tour := Generate(P);
       /* Apply Chained LK algorithm */
       optimized_tour := Chained_LK(current_tour);
       optimized_tour_length := Tour_length(optimized_tour);
       Update(P,optimized_tour,current_tour);

       /* Store the best tour found so far */
       if (optimized_tour_length < best_tour_length) then
          count := 0;
          best_tour_length := optimized_tour_length;
          best_tour := optimized_tour;
       end if
       count := count + 1;

       if (mod(generations, F_mig) = 0) then
          /* Perform migration */
          MPI_Allreduce(best_tour,best_global_tour,.......)
          if (best_global_tour = termination_signal) then
             Output_Results();
             exit;
          else
             Update(P,best_global_tour, best_tour);
          end if
       end if
    until (Local_Termination());

    MPI_Allreduce(termination_signal,.....);
    Output_Results();
end
```

Fig. 1. Pseudo-code of the parallel Cga-LK.

city $c_b \notin \mathcal{V}$ is randomly chosen. City c_b is inserted in $\mathcal{V}$ as successor of c_a with probability $p_{a,b}$ (i.e. the probability associated to edge (c_a, c_b)). Otherwise c_b is discarded and the process is repeated by choosing another city not belonging to $\mathcal{V}$. Clearly, this process may fail to find the successor of some city $c_{\overline{a}}$. This takes place when all the cities not already inserted in the current tour have been analyzed, but the probabilistic selection criterion failed to choose one of them. In this case the city $c_{\overline{b}}$ successor of $c_{\overline{a}}$ is selected according to the following formula:

$$\overline{b} = \text{argmax}\{p_{\overline{a},j} : c_j \in \{c_1, c_2, \ldots, c_k\} \setminus \mathcal{V}\} \tag{3}$$

When (3) is satisfied by several cities, i.e. edges $(c_{\overline{a}}, c_j)$ have the same probability for different cities $c_j \notin \mathcal{V}$, the city which minimizes the distance $d(c_{\overline{a}}, c_j)$ is selected. The generation process ends when all the cities have been inserted in $\mathcal{V}$, and a feasible tour has been thus generated.

The `current_tour` is then used as the starting solution for the Chained LK routine which produces an individual W (`optimized_tour`). Then the probability matrix is updated comparing `current_tour` with `optimized_tour` as follows:

$$p_{i,j}^{k+1} = \begin{cases} p_{i,j}^{k} + \frac{1}{n} & \text{if } (c_i, c_j) \vee (c_j, c_i) \in W \text{ and } (c_i, c_j) \vee (c_j, c_i) \notin L \\ p_{i,j}^{k} - \frac{1}{n} & \text{if } (c_i, c_j) \vee (c_j, c_i) \in L \text{ and } (c_i, c_j) \vee (c_j, c_i) \notin W \\ p_{i,j}^{k} & \text{otherwise} \end{cases} \tag{4}$$

Every `F_mig` generations a migration of individuals among different islands takes place to improve sub-population genotypes. The function `MPI_Allreduce` provides an efficient way to perform migration. It implements an all-to-all reduction operation where the tour achieving minimal length (`best_global_tour`) is broadcast to all the processes. The global optimum found so far is then used to update the probability matrix local to all the processes. The migration mechanism is used also to manage distributed termination. To this purpose, when a process reaches the termination condition, it broadcasts a `termination_signal` which is intercepted by the other processes that accordingly terminate their execution. Local termination is decided when a threshold is reached on the number of generations performed without an improvement of the best solution achieved, or on the elapsed execution time.

3.1 Experimental Results

The parallel Cga-LK algorithm was tested on some TSP instances defined in TSPLIB [26]. We used instances: `att532, gr666, rat783, pr1002` which have optimal solutions equal to $27686, 294358, 8806, 259045$, respectively. The experiments were conducted on a cluster of three Linux Pentium II 200 MHz PCs with 128 Mbytes of memory, and each test was repeated ten times to obtain an average behavior. Each Cga-LK process simulates a population of 128 individuals. In all the tests performed the parallel Cga-LK algorithm carried out

solutions with optimal length, independently from the parallelism degree and the migration frequency exploited. To make the algorithm performance evaluation independent from the computer architecture used, the comparison of the results obtained on each test was based on the number of generations performed. Figure 2 shows the average number of generations required to get the optimal tour as a function of the number of parallel processes used, for different values of the migration parameter. As it can be seen from the plots, the average number of generations required by the parallel algorithm to get the optimal tour is always lower than in the sequential case. Moreover, such number, in general decreases when the number of processes increases. With regard to the migration parameter, on smaller TSP instances a low value seems to work better than a large one, while on larger instances it affects slightly the results achieved. For the same tests, Figure 3 shows the minimal number of generations needed to achieve optimal solutions. Also in this case the benefits of parallelism exploitation are clear. The parallel algorithm always needs a lower number of iterations to obtain the optimal solution than those needed by the sequential version of the algorithm.

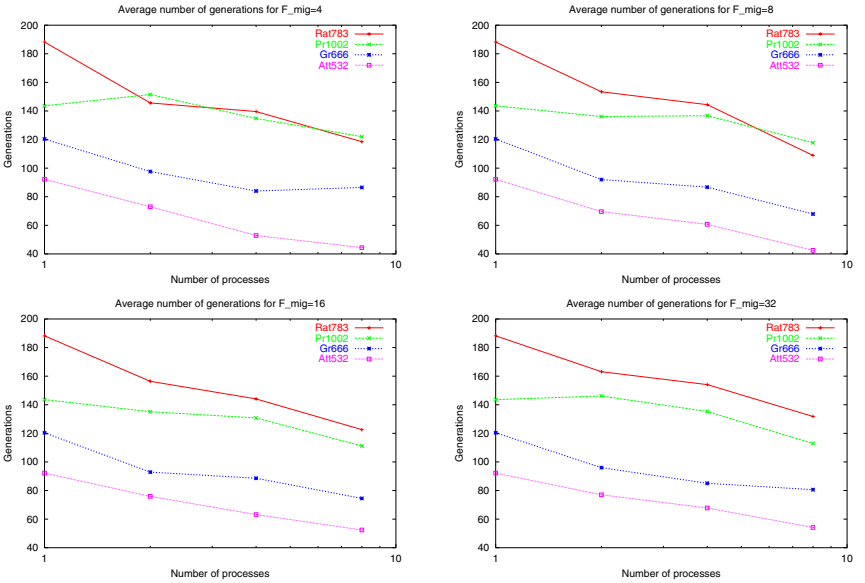

Fig. 2. Average number of generations required to carry out the optimal tour of various TSP instances as a function of the migration parameter and the number of parallel processes used.

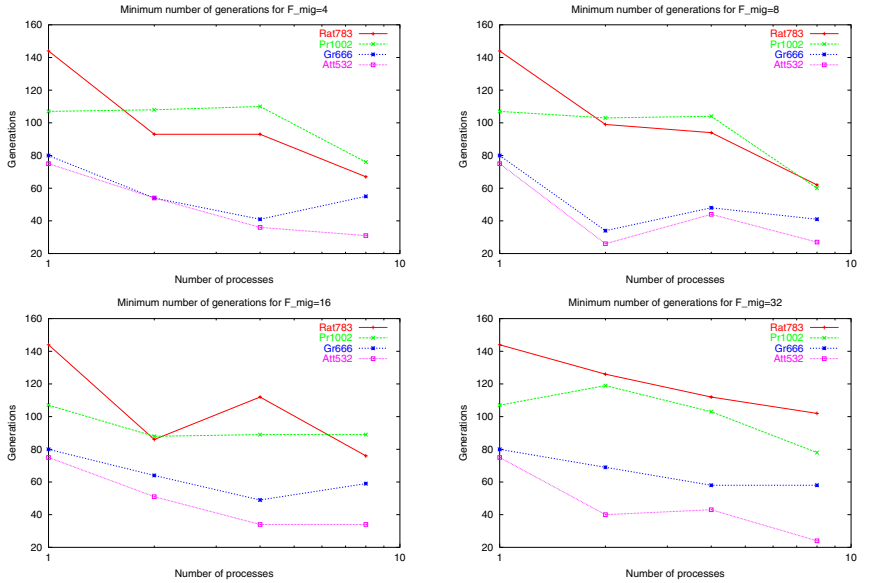

Fig. 3. Minimum number of generations required to carry out the optimal tour of various TSP instances as a function of the migration parameter and the number of parallel processes used.

4 Conclusions

In this paper we proposed a coarse-grained parallel hybrid heuristic to solve TSP. It combines a compact genetic algorithm to generate high-quality tours which are then refined by means of the Lin-Kernighan local search heuristic. The refined solutions are also exploited to improve the quality of the simulated population as the execution progresses. The parallel algorithm was implemented according to the SPMD programming paradigm by using the MPI message-passing library. Our parallel algorithm was evaluated on medium TSP instances. The results achieved were satisfactory if compared to those obtained by the sequential version of the algorithm on the same instances. The average number of generations required by the parallel algorithm to get the optimal tour was always lower than in the sequential case. As future work we plan to investigate either the behavior of the parallel algorithm on large TSP instances and extensions of the hybrid approach to solve other optimization problems.

Acknowledgments. This work was supported by the Italian National Research Council and the Spanish Government, Grant TYC 1999-0474.

References

1. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
2. J. Grefenstette, R. Gopal, B. Rosimaita, and D. van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the International Conference on Genetics Algorithms and their Applications*, pages 160–168, 1985.
3. H. C. Braun. On solving traveling salesman problems by genetic algorithm. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 129–133, Berlin, 1991. Springer-Verlag.
4. G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.
5. S. Lin. Computer solution of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.
6. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
7. D. S. Johnson and L. A. McGeoch. *Local Search in Combinatorial Optimization*, chapter The Traveling Salesman Problem: A Case Study in Local Optimization. John Wiley and Sons, New York, 1996.
8. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the tsp. Preliminary chapter of a planned monograph on the TSP, available at URL: `http://www.caam.rice.edu/~keck/reports/lk_report.ps`, 1999.
9. O. Martin and S.W.Otto. Combining simulated annealing with local search heuristic. *To appear on Annals of Operation Research*.
10. P. Merz and B. Freisleben. Genetic local search for the TSP: New results. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 159–163, Indianapolis, USA, 1997. IEEE press.
11. M. Gorges-Schleuter. Asparagos96 and the travelling salesman problem. In T. Bäck, editor, *Proceedings of the Fourth International Conference on Evolutionary Computation*, pages 171–174, New York, IEEE Press, 1997.
12. R. Perego R. Baraglia, J. I. Hidalgo. A hybrid approach for the TSP combining genetics and the Lin-Kerninghan local search. Technical Report CNUCE-B4-2000-007, CNUCE - Institute of the Italian National Research Council, 2000.
13. R. Perego R. Baraglia, J. I. Hidalgo. A hybrid approach for the Traveling Salesman Problem. *submitted paper*.
14. G. Harik, F. Lobo, and D. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.
15. D. Thierens and D. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45, San Mateo, CA, 1993. Morgan Kaufmann.
16. E. Cantu-Paz. A survey of parallel genetic algoritms. Technical Report 97003, University of Illinois at Urbana-Champaign, Genetic Algoritms Lab. (IlliGAL), http://gal4.ge.uiuc.edu/illigal.home.html, July 1997.
17. M. Tomassini. A survey of genetic algorithms. Technical Report 95/137, Department of Computer Science, Swiss Federal Institute of Technology, Lausanne, Switzerland, July 1995.
18. P. Grosso. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. PhD thesis, University of Michigan, 1985.

19. R. Tanese. Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 177–183. L. Erlbaum Associates, 1987.
20. R. Tanese. Distribuited genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–440. M. Kaufmann, 1989.
21. S. Cohoon, J. Hedge, S. Martin, and D. Richards. Punctuated equilibria: a parallel genetic algorithm. *IEEE Transaction on CAD*, 10(4):483–491, April 1991.
22. C. Pettey, M. Lenze, and J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 155–161. L. Erlbaum Associates, 1987.
23. H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.
24. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. Massachusetts Institute of Technology, 1999.
25. D. E. Culler and J. P. Singh. *Parallel Computer Architecture a Harware/Sotware Approach*. Morgan Kaufmann Publishers, Inc., 1999.
26. G. Reinelt. TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.